Reescritura de términos y sustituciones explícitas

Bonelli, Eduardo

2003

Tesis presentada para obtener el grado de Doctor en Ciencias de la Computación de la Universidad de Buenos Aires

Universidad de Buenos Aires

Facultad de Ciencas Exactas y Naturales

Reescritura de Términos y Sustituciones Explícitas

Autor: Eduardo Bonelli

Directores: Delia Kesner y Alejandro Ríos

Departamento de Computación

Tesis presentada para acceder al título de Doctor de la Universidad de Buenos Aires

Abstract

Substitution spans many areas in programming language theory. It plays a central role in the lambda calculus (hence functional programming), in first and higher-order unification (hence logic programming), in parameter passing methods (hence imperative programming), etc. Recently researchers became interested in shifting from the usual atomic, coarse grained view of substitution to a more refined, fine grained one. Substitution is promoted from the metalevel (our language of discourse) to the object-level (our language of study). This is interesting when studying the operational interpretation of the formalisms in question. Calculi of object-level or explicit substitution is the concern of this thesis. The following three study axes are developed.

First we consider perpetual rewrite strategies in lambda calculi of explicit substitutions. These are rewrite strategies that preserve the possibility of infinite derivations. Also, we study how to characterize inductively the set of terms that do not possess infinite derivations (the strongly normalizing terms). Polymorphic lambda calculus with explicit substitutions shall receive our attention too, including properties such as subject reduction and strong normalization.

Secondly, we put the ς-calculus of M.Abadi and L.Cardelli augmented with explicit substitutions under the microscope. This calculus is at the level of the lambda calculus but is based on objects instead of functions. Properties such as simulation of the lambda calculus, confluence and preservation of strong normalization (terms which are strongly normalizing in ς are also strongly normalizing in ς with explicit substitutions) are considered.

Finally, we address the task of reducing higher-order rewriting to first-order rewriting. We fix a variant of Z.Khasidashvili's *ERS* (dubbed SERS_{db}) as our departing formalism and provide a conversion procedure to encode any *ERS* as a first-order rewrite system in which a rewrite step takes place modulo an equational theory determined by a calculus of explicit substitutions. The latter is achieved with the aid of a macro-based presentation of calculi of explicit substitutions, thus parametrizing the conversion procedure over any calculus of explicit substitutions in compliance with the aforementioned presentation. The conversion procedure is in charge of encoding higher-order pattern matching and substitution in the first-order framework. Properties relating the rewrite relation in the higher-order framework and that of the resulting first-order system are studied in detail. We then identify a class of SERS_{db} for which the resulting first-order system does not require the equational theory to implement higher-order pattern matching, thus contenting itself with syntactic matching. It is argued that this class of systems is appropriate for transferring results from the first-order framework to the higher-order one. As a non-trivial example we study the transfer of the (strong) standardization theorem.

Resumen

La operación de sustitución constituye un engranaje básico en los fundamentos de la teoría de lenguajes de programación. Juega un rol central en el lambda cálculo (por ende, en lenguajes de programación funcional), en unificacón de primer orden y de orden superior (por ende, en lenguajes de programación basados en el paradigma lógico), en modalidades de pasaje de parámetros (por ende, en lenguajes de programación imperativos), etc. Recientemente, investigadores en informática se han interesado en el pasaje de la noción usual de la sustitución, atómica y de gruesa granularidad, hacia una noción más refinada, de más fina granularidad. La noción de sustitución es transportada del metalenguaje (nuestro lenguaje de discurso) al lenguaje objeto (nuestro lenguaje de estudio). Como consecuencia de ello se obtienen los llamados ***cálculos de sustituciones explícitas***. Estos son de sumo interés a la hora de estudiar la interpretación operacional de los formalismos en cuestión y constituyen los objetos de interés de esta tesis. Se desarrollan los siguientes tres ejes de estudio:

Primero, se consideran estrategias de reescritura perpetuas en lambda cálculos con sustituciones explícitas. Estas son estrategias de reescritura que preservan la posibilidad de reducciones infinitas. Se propone una caracterización inductiva del conjunto de términos que no poseen reducciones infinitas (los llamados fuertemente normalizantes). Un lambda cálculo polimórfico con sustituciones explícitas también es analizado, incluyendo propiedades tales como subject reduction y normalización fuerte.

Segundo, colocamos el ς-cálculo de M. Abadi and L. Cardelli enriquecido con sustituciones explícitas bajo el microscopio. Este cálculo se encuentra en un nivel semejante de abstracción al lambda cálculo pero se basa en objetos en lugar de funciones. Propiedades tales como simulación del lambda cálculo, confiuencia y preservación de la normalización fuerte (aquellos términos que son fuertemente normalizantes en ς también lo son en ς con sustituciones explícitas) son consideradas.

Finalmente, dirigimos nuestra atención a la tarea de relacionar la reescritura de orden superior con aquella de primer orden. Fijamos una variante de los *ERS* (apodados $SERS_{db}$) de Z. Khasidashvili como nuestro formalismo de orden superior de partida y definimos un proceso de conversión que permite codificar cualquier $SERS_{db}$ como un sistema de reescritura de primer orden. En este último, cada paso de reescritura se lleva a cabo módulo una teoría ecuacional determinada por un cálculo de sustituciones explícitas. La misma se formula de manera genérica a través de una presentación de cálculos de sustituciones explícitas basada en macros y axiomas sobre estas macros, parametrizando de esta manera al procedimiento de conversión sobre cualquier cálculo de sustituciones explícitas que obedece la presentación basada en macros. El procedimiento de conversión se encarga de codificar pattern matching de orden superior y sustitución en el entorno de reescritura de primer orden. Asimismo, propiedades que relacionan la noción de reescritura en el orden superior con aquella de primer orden son analizadas en detalle. Se identifica una clase de $SERS_{db}$ para los cuales el sistema de primer orden resultante de su conversión no requiere una teoría ecuacional para implementar pattern matching de orden superior, bastando para ello matching sintáctico. También se argumenta que esta clase de sistemas de orden superior es apropiada para transferir resultados del entorno de reescritura de orden superior a aquella de primer orden. A modo de ejemplo no-trivial de ello, estudiamos la transferencia del teorema de standarización (fuerte).

Résume étendu

Le calcul peut être considéré comme la tâche consistant à la transformation d'un objet, la *donnée*, en un objet nouveau, le *résultat*, en utilisant certaines *règles de transformation*. Un système de réécriture de termes [Klo92] est un modèle calculatoire dans le sens où les objets sont spécifiés comme des séquences de symboles et les règles de transformations comme des ensembles de paires d'objets capturés par un *schéma* ou *règle de réécriture*. Un premier exemple de système de réécriture de termes est le λ-calculus [Chu41], créé par A. Church dans les années 1930. Les objets s'appellent λ-termes et représentent des fonctions ; la seule règle de transformation est la règle β qui code le processus d'application d'une fonction à son argument. Par exemple, le terme "$\lambda x.x$" représente la fonction identité, l'occurrence la plus à gauche de "x" joue le même rôle que le paramètre formel d'une fonction ou procédure dans un langage impératif de programmation (tel que Pascal).

Le λ-terme "$(\lambda x.x)$ 4" est un exemple d'application d'une fonction à son argument, il représente la fonction identité appliquée à la représentation de l'entier 4. On remarque que l'application est dénotée par la juxtaposition des termes. On peut alors calculer en appliquant effectivement la fonction à l'argument grâce à la règle β :

$$(\lambda x.M)N \quad \rightarrow_\beta \quad M\{x \leftarrow N\}$$

où M et N dénotent des λ-termes arbitraires et x une variable arbitraire. L'expression $(\lambda x.M)N$ est aussi nommée le *membre gauche* de la règle et abrégée *LHS*. Les symboles $\bullet\{\bullet \leftarrow \bullet\}$[1] qui apparaît dans le membre droit (abrégé désormais *RHS*) de la règle représente l'opération de substitution dans le méta-langage : $M\{x \leftarrow N\}$ représente donc le λ-terme qui résulte du remplacement des occurrences de la variable x (le paramètre formel) dans M par N (le paramètre actuel). En réalité, seules les occurrences dites libres de la variable x sont remplacées, mais nous reviendrons sur les détails techniques plus tard. Un λ-terme de la forme $(\lambda x.M)N$ est appelé un β-radical. L'exemple suivante illustre un calcul en un seul pas de la règle β :

$$(\lambda x.x)4 \quad \rightarrow_\beta \quad x\{x \leftarrow 4\} = 4$$

On dit que le λ-terme $(\lambda x.x)4$ se β-réduit au terme 4. Un λ-terme sans β-radical est une *β-forme normale* ou plus simplement *forme normale*. Le λ-terme 4 est en particulier une forme normale.

Il est possible de montrer que le λ-calculus avec la seule règle β peut représenter ou coder toutes les fonctions calculables (plus formellement les fonctions partielles récursives). Ce fait de même que sa formulation simple et concise justifie l'intérêt de la communauté informatique pour ce formalisme. Il est cependant difficile de concevoir des programmes concrets écrits sous forme de λ-termes. Les langages de programmation fonctionnels (comme, par exemple, ML [MTH90], Haskell [HW88] et CAML [WL93]) sont essentiellement des représentations utilisables en pratique du λ-calculus. Même s'ils ont considérablement évolué ces cinq dernières années, ils partagent tous comme fondement théorique le λ-calculus. En fait, pour tous ces langages, le λ-calculus fournit un scénario de preuve convenable pour l'étude de nouvelles constructions de langages.

Certaines questions auparavant négligées sont désormais étudiées par les informaticiens car le λ-calculus est considéré comme la base de certains langages de programmation :

- Le processus de calcul s'arrête-t-il à partir d'un λ-terme quelconque ?
- Étant donné un λ-terme avec au moins deux β-radicaux, ceux-ci peuvent-ils être réduits dans un ordre quelconque ?
- Peut-on considérer le nombre de pas de β-réduction comme une bonne mesurede la consommation des ressources de calcul ? Par exemple, les pas suivants de β-réduction ont-ils le même coût ?

$$\begin{array}{lll}(\lambda x.x)4 & \rightarrow_\beta & x\{x \leftarrow 4\} = 4 \\ (\lambda x.xx)4 & \rightarrow_\beta & (xx)\{x \leftarrow 4\} = 44\end{array}$$

[1] On utilisera $\bullet$ pour dénoter un trou qui sera remplacé dans la suite par des expressions.

Le premier de ces points, la "*terminaison*" ou "*normalisation forte*", a été étudiée en profondeur. Le λ-calculus, tel que nous l'avons présenté, ne jouit pas de la propriété de terminaison. L'exemple le plus simple[2] pour illustrer ceci est le λ-terme $\Delta\Delta$ où $\Delta = \lambda x.xx$.

$$(\lambda x.xx)\Delta \rightarrow_\beta (xx)\{x \leftarrow \Delta\} = \Delta\Delta \rightarrow_\beta \Delta\Delta \rightarrow_\beta \ldots$$

Cependant, pour des ensembles restreints de termes[3] on a montré que la propriété de terminaison est vraie.

Revenons au deuxième point. Le théorème de Church-Rosser, peut-être le premier théorème syntaxique important développé pour le λ-calcul, dit que si deux λ-termes M_1 et M_2 sont obtenus à partir d'un autre λ-terme M comme résultats de β-réductions, alors il existe un autre λ-terme M_3 tel que M_1 et M_2 se β-réduisent (en un nombre convenable de pas) à M_3. Ce fait est illustré ainsi :

$$\begin{tikzcd}
& M \arrow[dl, two heads, "\beta"'] \arrow[dr, two heads, "\beta"] & \\
M_1 \arrow[dr, two heads, "\beta"'] & & M_2 \arrow[dl, two heads, "\beta"] \\
& M_3 &
\end{tikzcd}$$

où la flèche $\twoheadrightarrow_\beta$ dénote zéro ou plusieurs pas de β-réduction (ce qu'on appelle aussi *β-dérivation*). Par conséquent, si l'on déduit différents radicaux dans M on trouve toujours un réduit commun (M_3 dans la figure). Une conséquence importante de ce fait est que si un λ-terme possède une forme normale, alors elle est unique.

Quant au troisième point, celui de l'analyse raffinée des propriétés des β-dérivations (que nous appellerons "techniques d'implantation"), comme par exemple les techniques de partage d'information sur les graphes et les machines abstraites. Une idée relativement récente est de donner à l'opération de substitution dans le méta-langage (le langage du discours) le droit de citoyenneté dans le langage-objet (le langage d'étude) en l'introduisant comme un nouvel opérateur. Ceci entraîne l'addition de nouvelles règles de réécriture, donnant lieu au calcul dit *calcul de substitutions* qui permet de décrire son mécanisme. De cette manière, la β-règle doit être modifiée en remplaçant la substitution du méta-langage par le nouvel opérateur de substitution explicite qui le spécifie. Comme résultat de cette transformation on obtient un *λ-calculus de substitutions explicites.* Avant de présenter un exemple, nous étudierons l'opération de substitution dans le méta-langage (celle qui est utilisée dans le membre droit de la règle β) sur laquelle sont fondés les calculs de substitutions explicites. Comme il a été dit plus haut, $M\{x \leftarrow N\}$ représente le λ-terme obtenu en remplaçant les occurrences de la variable x (le paramètre formel) dans M par N (le paramètre actuel). Cette notion est définie en considérant toutes les formes possibles de M, c'est-à-dire : une variable, une abstraction (un terme de la forme $\lambda y.P$, où P est un λ-terme et y une variable) ou bien une application d'un λ-terme à un autre λ-terme :

$$\begin{array}{llll}
(PQ)\{x \leftarrow N\} & \overset{\text{def}}{=} & P\{x \leftarrow N\}Q\{x \leftarrow N\} & \\
(\lambda y.P)\{x \leftarrow N\} & \overset{\text{def}}{=} & \lambda y.(P\{x \leftarrow N\}) & x \neq y \\
x\{x \leftarrow N\} & \overset{\text{def}}{=} & N & \\
y\{x \leftarrow N\} & \overset{\text{def}}{=} & y & x \neq y
\end{array}$$

La première clause peut être lue ainsi : le résultat de la substitution des occurrences de x par N dans PQ est l'application des termes résultant de la substitution des occurrences de x par N dans P et Q. Nous pouvons supposer qu'il n'y a pas de terme de la forme $\lambda x.M'$ dans M, grâce à la possibilité de renommer les variables dites liées, un rapport détaillé de ce fait se trouve au chapitre 2 de la thèse. La deuxième clause peut s'expliquer d'une manière semblable et les deux dernières sont évidentes. Par conséquent, $\{x \leftarrow N\}$ traverse M jusqu'aux variables et finalement les variables sont remplacées par une copie de N ou restent inaltérées en même temps que la copie de N est rejetée. Si les clauses de cette définition sont orientées de gauche à droite et les accolades remplacées par des crochets, en internalisant ainsi l'opérateur de substitution du méta-langage comme un nouvel

[2] En effet, on a montré que $\Delta\Delta$ est le λ-terme le plus petit (ayant le moins de symboles) qui admet une β-réduction infinie [RSSX99, Omega théorème de Sørensen].

[3] Par exemple, les termes simplement typés ou les termes avec des types polymorphes [GLT89].

opérateur dans le langage-objet, nous obtenons le λx-calcul [Ros92, Blo97] :

$$\begin{array}{llll}
(\lambda x.M)N & \rightarrow_{Beta} & M\langle x := N\rangle & \\
(PQ)\langle x := N\rangle & \rightarrow_{App} & P\langle x := N\rangle Q\langle x := N\rangle & \\
(\lambda y.P)\langle x := N\rangle & \rightarrow_{Lam} & \lambda y.(P\langle x := N\rangle) & x \neq y \\
x\langle x := N\rangle & \rightarrow_{Var} & N & \\
y\langle x := N\rangle & \rightarrow_{Varf} & y & x \neq y
\end{array}$$

Maintenant un terme est soit une variable, soit une application d'un terme à un autre terme (représenté, comme auparavant, par juxtaposition), soit une abstraction, soit un terme de la forme $P\langle x := Q\rangle$ appelé une "clôture". De la même manière que la portée de x (terme dans lequel les occurrences de x sont liées) dans $\lambda x.M$ est M, le terme P est la portée de x dans $P\langle x := Q\rangle$. Alors le *RHS* de la règle *Beta* est un terme nouveau dans le calcul et peut être considéré comme une substitution en attente qui devra être exécutée. Le calcul de substitutions de λx est obtenu à partir de λx en enlevant la *Beta*-règle et sera dénoté par x. Chaque pas de β-réduction peut être simulé dans le λx-calcul. Considérons, par exemple, le premier pas de la β-dérivation décrite plus haut. Il peut être simulé dans λx comme suit :

$$(\lambda x.xx)\Delta \rightarrow_{Beta} (xx)\langle x := \Delta\rangle \rightarrow_{App} x\langle x := \Delta\rangle x\langle x := \Delta\rangle \rightarrow_{Var} \Delta x\langle x := \Delta\rangle \rightarrow_{Var} \Delta\Delta$$

L'un des bénéfices des calculs de substitutions explicites est que la substitution peut maintenant être calculée d'une façon contrôlée. Par exemple, certaines substitutions en attente peuvent ne pas être exécutées, comme l'illustre l'exemple suivant :

$$\begin{array}{lll}
(\lambda y.(\lambda x.z)(yy))N & \rightarrow_{Beta} & ((\lambda x.z)(yy))\langle y := N\rangle \\
& \rightarrow_{App} & (\lambda x.z)\langle y := N\rangle(yy)\langle y := N\rangle \\
& \rightarrow_{Lam} & (\lambda x.z\langle y := N\rangle)(yy)\langle y := N\rangle \\
& \rightarrow_{Varf} & (\lambda x.z)(yy)\langle y := N\rangle \\
& \rightarrow_{Beta} & z\langle x := (yy)\langle y := N\rangle\rangle \\
& \rightarrow_{Varf} & z
\end{array}$$

Il faut remarquer qu'il n'a pas été nécessaire de calculer la substitution $(yy)\langle y := N\rangle$, en réduisant ainsi le temps de calcul et la duplication superflue du terme N. Les calculs de substitutions explicites seront le principal sujet d'étude de cette thèse.

En présence d'un calcul de substitutions explicites pour le λ-calculus il est naturel de se demander si les propriétés dynamiques (celles du λ-calculus) sont toujours valides. Nous énumérons trois exemples :

- Simulation : Si un λ-terme M se β-réduit à N alors M devrait aussi se réduire à N dans le calcul de substitutions explicites. Ceci est cohérent avec notre vue des calculs de substitutions explicites en tant qu'analyse plus précise de la β-réduction, et par la suite du processus de substitution.

- Church-Rosser : Le λ-calculus jouit de la propriété de Church-Rosser. Son calcul de substitutions explicites devrait en jouir aussi. Si l'on interprète le "sens" d'un terme comme sa forme normale (dans notre cadre simplifié des termes sans forme normales n'ayant pas de "sens"), alors l'absence de cette propriété pourrait rendre quelques termes "ambigus" (termes avec plus d'une forme normale). Ceci est bien sûr indésirable puisque, comme nous l'avons déjà remarque, il n'y a pas de λ-termes ambigus dans le λ-calculus.

- Préservation de la Normalisation Forte (PSN) : Si un λ-terme n'admet pas de dérivations infinies dans le λ-calculus, alors quand on réduit ce même terme dans le calcul de substitutions explicites on ne devrait pas engendrer une dérivation infinie non plus. Le processus d'augmenter le λ-calculus avec des substitutions explicites peut être vu comme un processus d'enrichissement de dérivations. N'importe quelle paire de λ-termes M et N tels que $M \twoheadrightarrow_\beta N$ bénéficie d'une riche provision de dérivations alternatives. PSN garantit que l'on enrichit avec précaution.

Il y a encore des autres propriétés telles que la normalisation forte du calcul de substitutions associé (qui dans notre exemple serait x).

Compte rendu de la thèse

Le corps principal de cette thèse est divisé en trois parties :

Partie I	Chapitre 3 : Perpétuité dans le calcul de substitutions explicites λx
	Chapitre 4 : Perpétuité dans le calcul de substitutions explicites λws
Partie II	Chapitre 5 : Enregistrements et substitutions explicites pour les objets et les fonctions
Partie III	Chapitre 6 : Une notation de De Bruijn pour la réécriture d'ordre supérieur
	Chapitre 7 : De la réécriture d'ordre supérieur à la réécriture du premier ordre
	Chapitre 8 : Transfert de la standardisation

Celles-ci sont précédées par une brève introduction aux théories de réécriture, λ-calculus et calculs de substitutions explicites (chapitre 2). Dans la suite nous présentons un résumé du contenu de ces parties en détaillant les principaux résultats obtenus dans chacune d'entre elles.

Partie I : Perpétuité dans les calculs de substitutions explicites

La première partie de la thèse étudie la perpétuité dans les calculs de substitutions explicites : la contraction de radicaux qui préserve la possibilité de dérivations infinies, ces radicaux sont appelés des radicaux perpétuels. Une stratégie de réécriture qui réduit toujours un radical perpétuel est appelée *stratégie de réécriture perpétuelle*. Par exemple, la stratégie "plus à gauche" (celle qui réduit le radical le plus à gauche) n'est pas perpétuelle pour le λ-calculus. En effet, il suffit de considérer le λ-terme $M = (\lambda x.y)(\Delta\Delta)$ où $\Delta = \lambda x.xx$. Alors M admet une β-dérivation infinie, tout simplement en réduisant le radical le plus à droite :

$$(\lambda x.y)(\Delta\Delta) \to_\beta (\lambda x.y)(\Delta\Delta) \to_\beta \dots$$

Mais si l'on réduit le radical le plus à gauche dans M on obtient le terme $N = y$ qui est une forme normale. Comme le lecteur peut observer, ceci est dû à la nature d'effacement du radical contracté puisque le sous-terme $\Delta\Delta$ n'apparaît plus dans N, il a été effacé. L'intérêt des stratégies perpétuelles est que, si elles normalisent un terme M, alors ce terme est fortement normalisant (c'est à dire, toutes les dérivations commençant par M sont finies). Par exemple, nous utiliserons nos études sur la perpétuité pour caractériser inductivement l'ensemble de tous les termes du λx-calcul qui sont fortement λx-normalisables. Par "inductivement" nous voulons dire que l'ensemble sera décrit comme le plus petit ensemble vérifiant certaines règles, de la même manière que l'on décrit l'ensemble des λ-termes ou l'ensemble des théorèmes d'un certain système logique [Acz77]. Nous utilisons aussi la perpétuité pour donner une preuve de normalisation forte d'un lambda calcul polymorphe avec des substitutions explicites.

Les stratégies de réécriture perpétuelles pour le λ-calculus ont été introduites dans [BBKV76], une étude d'ensemble récente est [RSSX99]. Un système de réécriture de termes (*TRS*) est dit *uniformément normalisable* si tous ses radicaux sont perpétuels. Un *TRS* est dit *orthogonal* ou *déterministe* si le membre gauche de chaque règle a au plus une occurrence de chaque variable et si il n'y a pas de superposition. En fin, on a les *TRS non-effaçants* : dans un terme quelconque les arguments déterminés par le *LHS* d'une règle apparaît aussi dans les arguments déterminés par le *RHS*. J.W. Klop [Klo80] a montré que tous les *CRS* orthogonaux non-effaçants (et alors tous les *TRS* du premier ordre orthogonaux non-effaçants) sont uniformément normalisables, en généralisant ainsi le Théorème de Church [CR36] qui établit que le λ_I-calcul[4] est uniformément normalisables. Quant à la caractérisation des radicaux perpétuels Z. Khasidashvili [Kha94, Kha01] a montré que tous les radicaux non-effaçants sont perpétuels dans les *ERS* (et donc que tous les radicaux non-effaçants sont perpétuels dans les *TRS* orthogonaux du premier ordre), en généralisant ainsi le Théorème de Conservation [BBKV76] qui établit que les β_I-radicaux (i.e. les β-radicaux dans λ_I) sont perpétuels dans le λ-calculus. Les abréviations *CRS* et *SERS* sont des noms de formalismes de réécriture d'ordre supérieur, le lecteur peut trouver plus de détails dans le chapitre 6 de la thèse. On peut se référer à [KOO01a] pour une étude exhaustive des stratégies de réécriture perpétuelles et une caractérisation des radicaux perpétuels dans les systèmes de réécriture de termes d'ordre supérieur.

Tous ces résultats sont formulés pour des systèmes orthogonaux, mais le calcul de substitutions explicites λx n'est pas orthogonal. Les résultats sur la perpétuité déjà développés ne peuvent donc pas s'appliquer. Nous étudierons perpétuité dans le λx-calcul en adaptant une technique introduite à l'origine pour montrer

[4] Le λ_I-calcul est obtenu à partir du λ-calculus en restreignant la formation des termes : toute abstraction $\lambda x.M$ doit contenir au moins une occurrence libre de la variable x dans M.

préservation de la normalisation forte des calculs de substitutions explicites [BBLRD96]. Les applications de cette étude sont :

- formulation d'une définition inductive de l'ensemble des termes fortement $\lambda\mathtt{x}$-normalisables,
- deux stratégies de réécriture perpétuelles pour $\lambda\mathtt{x}$, l'une d'elles est calculable,
- une preuve de normalisation forte d'un calcul typé polymorphe avec des substitutions explicites.

Nous étudions aussi $\lambda\mathtt{ws}$. Le $\lambda\mathtt{ws}$-calcul [DG99] est un calcul de substitutions explicites fondé sur la notation de De Bruijn (une notation pour le termes dans lequel les variables sont codés par des nombres) qui est considérablement plus compliqué que $\lambda\mathtt{x}$ à cause de la présence de la composition des substitutions. Nous développons pour $\lambda\mathtt{ws}$ un programme d'étude semblable à celui que nous venons de décrire pour $\lambda\mathtt{x}$, la seule différence étant que nous ne traitons pas un calcul polymorphe avec des substitutions explicites fondé sur $\lambda\mathtt{ws}$. Pour offrir au lecteur une idée sûr la manière de composer des substitutions nous donnons un exemple de règle qui fait cette tâche dans le $\lambda\mathtt{x}$-calcul :

$$M\langle x := N\rangle\langle y := O\rangle \quad \to_c \quad M\langle x := N\langle y := O\rangle\rangle \qquad \text{si } y \text{ n'est pas libre dans } M$$

Dans le chapitre 4 de la thèse nous développons une stratégie de réécriture perpétuelle pour $\lambda\mathtt{ws}$ et nous formulons une caractérisation inductive du $\lambda\mathtt{ws}$-termes qui terminent.

Récemment, des travaux supplémentaires sur la perpétuité pour des systèmes non-orthogonaux, avec des applications à des calculs de substitutions explicites, ont été développés [KOO01b].

Proposition de Perpétuité et quelques applications

Le chapitre 3 de la thèse commence par considérer les résultats fondamentaux de la technique de preuve développées par P. Lescanne et al dans [BBLRD96] afin de démontrer la préservation de la normalisation forte de λv, un autre calcul de substitutions explicites que nous introduisons brièvement dans le chapitre 2. Cette technique est fondée sur l'assignation d'une mesure aux dérivations et sur l'application d'un argument de minimalité, semblable à celui que Nash-Williams [NW63] a utilisé pour arriver à une contradiction dans la preuve du théorème de Kruskal. Nous appliquons cette même technique pour démontrer la *proposition de perpétuité*. Cette proposition dit que si M se réécrit dans N à l'aide du calcul de substitutions $\mathtt{x}$ et M possède une $\lambda\mathtt{x}$-dérivation infinie, alors N aussi. Dans le cas où la règle de réécriture *Varf* est utilisée pour obtenir N de M, nous devons demander additionnellement que le terme éliminé par cette règle soit fortement $\lambda\mathtt{x}$-normalisable.

Cette proposition est suffisante pour pouvoir formuler une caractérisation inductive de l'ensemble des $\lambda\mathtt{x}$-termes qui terminent, c'est à dire qui ne sont pas la source d'une dérivation infinie. En effet, sept schémas d'inférence sont proposés et à partir de la proposition mentionnée auparavant, nous montrons que ces schémas capturent exactement l'ensemble des $\lambda\mathtt{x}$-termes qui terminent.

Ensuite, nous considérons des stratégies perpétuelles pour le $\lambda\mathtt{x}$-calcul. Une $\lambda\mathtt{x}$-stratégie de réécriture est une fonction $\mathcal{F}(\bullet)$ de $\lambda\mathtt{x}$-termes en termes telle que pour tout $\lambda\mathtt{x}$-terme M on a $M \to_{\lambda\mathtt{x}} \mathcal{F}(M)$, à l'exception du cas où M soit une forme normale. Dans ce dernière cas nous avons $\mathcal{F}(M) = M$. On dit qu'une stratégie est perpétuelle si $\infty_{\lambda\mathtt{x}}(M)$ implique $\infty_{\lambda\mathtt{x}}(\mathcal{F}(M))$, où $\infty_{\lambda\mathtt{x}}(M)$ indique que M est la source d'une dérivation infinie dans $\lambda\mathtt{x}$.

Nous formulons deux stratégies de réécriture pour $\lambda\mathtt{x}$. Nous montrons qu'elles sont perpétuelles à l'aide de la proposition de perpétuité. Les deux stratégies consistent à réécrire l'un des $\lambda\mathtt{x}$ radicaux le plus à gauche, à exception du cas où ce radical est un *Varf*-radical $y\langle x := Q\rangle$. Dans ce dernière cas soit un radical dans Q est réduit soit le *Varf*-radical lui même est réduit selon des conditions sur Q.

Le problème suivant sur les stratégies maximales est resté ouvert. Soit $\mathcal{F}$ une $\lambda\mathtt{x}$ stratégie de réécriture. Définissons $L_{\mathcal{F}}(M) \overset{\text{def}}{=} \min\{n \mid \mathcal{F}^n(M) \text{ est une } \lambda\mathtt{x}\text{-forme normale}\}$ ou $\mathcal{F}^0(M) \overset{\text{def}}{=} M$ et $\mathcal{F}^{n+1}(M) \overset{\text{def}}{=} \mathcal{F}(\mathcal{F}^n(M))$. Alors on dit que $\mathcal{F}$ est maximal ssi M possède une $\lambda\mathtt{x}$-forme normale et $L_{\mathcal{F}}(M)$ coïncide avec la longueur maximale d'une dérivation qui mène M à sa forme normale. La stratégie $\mathcal{F}_\infty(\bullet)$ (Définition 3.16, Section 3.2.2 de la thèse) est elle maximale?

Lambda calcul polymorphe avec substitutions explicites

Le polymorphisme est une discipline de types qui permet aux fonctions de s'appliquer sur des arguments de différents types. Ceci aide à la réutilisation de code et, avec les fonctions d'ordre supérieur, est l'un des ingrédients essentiels de n'importe quel langage de programmation moderne. Un exemple classique de fonction polymorphe est la fonction identité $I = \lambda x.x$. Notons que pour un terme quelconque, cette fonction retourne une copie de ce terme, sans regarder la nature du terme en question. Ainsi, pour chaque type τ on dit que I possède type $\tau \rightarrow \tau$. Ceci peut être rendu interne dans le langage des types en assignant le type $\forall\tau.\tau \rightarrow \tau$ à I, où τ joue le rôle d'une variable de type, qui varie sur tous les éléments de l'univers de types. Le système F [Gir72, GLT89] est un ensemble de règles de typage pour typer des types polymorphes et deux règles de réécriture : le règle β et une règle pour instancier des variables de type par des types arbitraires. Nous augmentons le calcul F avec des substitutions explicites, en obtenant F_{es}. Comme nous pouvons abstraire pas seulement de variables de terme mais aussi de variables de type, deux notions distinctes de substitution seront introduites et étudiées : substitution de terme et substitution de type. On considère alors les propriétés suivantes :

- Préservation de types : la préservation de types est démontrée pour F_{es}. Ce résultat n'est pas vrai pour d'autres formulations de lambda calculs avec substitutions explicites basés sur λx [Blo97]. Cette situation a été ensuite inversée dans [Blo99, Blo01], elle peut donc être vue comme une solution indépendante à ce problème. Un travail en relation avec nos résultats est celui de C. Muñoz [Muñ97b] qui définit un lambda calcul typé avec substitutions explicites (basé sur une variante linéaire à gauche du calcul $\lambda\sigma$) et types dépendants. Il démontre la préservation de types pour ce calcul en introduisant des annotations de type dans le constructeur de substitutions $\bullet \cdot \bullet$ de $\lambda\sigma$. Notre solution consiste à considérer des contextes de typage dans lesquels une notion d'égalité de types modulo un calcul de substitutions explicites pour les substitutions de types sont pris en compte.

- Normalisation forte : nous démontrons que tous les termes polymorphes typables sont fortement normalisables. La preuve est obtenue en appliquant la technique des candidats de réductibilité de J-Y. Girard [Gir72], même si nous suivons plutôt la présentation donnée par J. Gallier [Gal90]. La réductibilité pour démontrer des propriétés de calculs avec substitutions explicites apparaît d'abord dans [Muñ97b] et [Rit93], néanmoins elle est seulement utilisée pour démontrer la normalisation faible. En ce qui concerne la normalisation forte, il y a eu aussi d'autres contributions indépendantes [Rit99, DL01, Her00]. L'idée que nous suivons consiste à définir une fonction d'effacement $\mathit{Erase}(\bullet)$ qui élimine toute l'information de types d'un terme typé dans F_{es} en produisant un λx-terme non typé.

  - Montrer que si un terme résultant de l'effacement de l'information de types est fortement λx-normalisable, alors le terme original est fortement F_{es}-normalisable.
  - Nous obtenons ainsi le résultat suivant : si M est F_{es}-typable alors $\mathit{Erase}(M)$ est fortement λx-normalisable. Nous généralisons ce but en montrant que si M est typable alors $\mathit{Erase}(M)$ suivi d'une séquence de substitutions explicites est fortement λx-normalisable, ceci implique clairement que lui même est fortement λx-normalisable.
  - Le point précédent est démontré à l'aide de la technique des candidats de réductibilité où les candidats sont des sous-ensembles de λx-termes qui terminent. La proposition de perpétuité est cruciale pour que cette technique puisse être appliquée.

Partie II: Subtitutions explicites pour un calcul à objets

Nous considérons ici un calcul avec substitutions explicites pour modéliser des langages orientés objets en enrichissant le ς-calcul de M. Abadi et L. Cardelli [AC96] avec des substitutions explicites. Le ς-calcul est un formalisme qui se trouve au même niveau d'abstraction que le lambda calcul, mais qui se base sur des objets à la place que sur des fonctions. On peut le considérer un calcul minimal dans le sens qu'il s'avère difficile de concevoir un calcul plus simple pour modéliser des constructions des langages orientés objets. Un objet dans le ς-calcul est une collection de *méthodes*, les seules structures calculatoires dans le formalisme.

Le ς-calcul est Turing complet dans le sens qu'il peut représenter toutes les fonctions calculables. En particulier, ce résultat est démontré [AC96] en proposant une traduction du lambda calcul dans le ς-calcul. La traduction simple et élégante qui achève ce codage est appelée la *traduction fonction-objet*. Maintenant, si on fixe un calcul de substitutions explicites, disons e, pour rendre la méta-substitution dans ς au niveau objet, il est naturelle d'espérer que le calcul de substitutions explicites résultant ςe soit capable de coder λe. Par exemple, si on considère des substitutions explicites à la v [BBLRD96] pour augmenter le ς-calcul, obtenant ainsi ςv, on voudrait vérifier que λv est codable dans ςv, et ainsi ςv serait au moins si expressif que le lambda calcul. Dans un contexte où les variables liées son dénotées par des noms et pas par des indices, D. Kesner and P.E. Martínez López [KML98] ont vérifié que λx peut être simulé dans ςx. Pour vérifier cette propriété de simulation ils ont adapté la traduction fonction-objet en introduisant une nouvelle notion de substitution appelée ***substitution*** *d'invocation.* Cette substitution se comporte différemment de la notion usuelle de substitution car elle représente plutôt un remplacement qu'une notion de substitution d'ordre supérieur. De plus, dans l'environnement avec indices et substitutions explicites présenté dans cette thèse, nous avons vérifié que les *enregistrements* sont aussi nécessaires pour coder le λe dans ςe via une traduction fonction-objet. Le chapitre 5 de la thèse étudie un calcul du premier-ordre avec explicit substitutions pour le ς-calcul avec enregistrements comme constructeurs primitifs. Simulation du $\lambda\upsilon$-calcul, confluence et préservation de la normalisation forte conforment le centre d'attention de ce chapitre.

L'étude des calculs avec substitutions explicites s'est initiée dans le domaine du λ-calculus, cependant il y a eu plusieurs travaux dans le domaine plus général de la réécriture d'ordre supérieur connue sous le nom des *CRS* [Klo80], en particulier les Explicit Combinatory Reduction Systems [BR96] et les eXplicit Reduction Systems (*XRS*) de Pagano [Pag98]. Ces formalismes, même s'ils sont définis dans un cadre d'ordre supérieur travaillent avec un calcul de substitutions explicites fixe (Σ dans le cas des Explicit *CRS* et $\sigma_{\Uparrow}$ dans le cas des *XRS*). Nous verrons que notre calcul de substitutions explicites proposé comme une implémentation du ς-calcul n'est une instance d'aucun de ces deux formalismes mentionnés auparavant. Au moment de la rédaction de cette thèse nous avons appris l'existence d'une formalisation indépendante du ς-calcul publiée par M-O. Stehr [Ste00] et basée sur une représentation alternative qui utilise les termes avec la notation de Berkling. Cette notation peut être vue comme le résultat de fusionner les indices de De Bruijn et les noms de variables. Ce notation serve pas pour résoudre nos problèmes.

Un autre travail mélangeant calculs avec substitutions explicites et calculs orientés objets est celui de F. Lang et al [LLL98]. Le mérite de ce travail est de donner un environnement unifié pour étudier la sémantique opérationnelle de plusieurs calculs à objets, donc il peut être vu comme une approche orthogonale à la notre. De plus, ce formalisme est basé sur une extension du $\lambda\mathcal{O}bj$-calcul [FHM94] plutôt que du ς-calcul. Comme le $\lambda\mathcal{O}bj$-calcul inclut le λ-calculus, les traductions fonction-objet ne sont pas nécessaires dans ce cadre.

L'incorporation de substitutions explicites à ς

Le ς-calcul avec substitutions explicites et indices de De Bruijn, que nous appellerons ς_{dbes}-calculus, est présenté dans le chapitre 5 de la thèse. Ce calcul introduit deux formes de substitution dans le langage objet : substitution ordinaire et substitution d'invocation. Les règles de réduction pour les substitutions ordinaires sont basés sur le calcul $\lambda\upsilon$ de P. Lescanne et al. Mais nous avons aussi des enregistrements (explicites) dans le langage objet de ς_{dbes}. La section 5.4.1 de la thèse explique pourquoi nous avons besoin des enregistrements comme constructeurs primitifs dans le langage.

Les substitutions de la forme $a/$ sont appelées *substitutions ordinaires* tandis que les substitutions contenant $@l$ sont appelées *substitutions d'invocation.* L'ajustement des indices sera différent pour ces deux types de substitutions. Une autre caractéristique intéressante de ς_{dbes} est qu'elle possède une forme limitée de composition de substitutions.

Comme nous l'avons dit auparavant les enregistrements peuvent être simulés dans le ς_{db}-calculus d'une manière très naturelle. Cette situation n'est plus valable lorsque les substitutions explicites sont introduites dans le formalisme et lorsque l'on veut coder le $\lambda\upsilon$-calcul dans le ς_{dbes}-calculus à l'aide d'une traduction fonction-objet. Les détails se trouvent dans la section 5.4.1 de la thèse.

Enfin, quelques propriétés de ς_{dbes} sont étudiées. La première d'entre elles dit que de la même manière que ς est capable de simuler λ-calculus, le ς_{dbes}-calcul peut simuler $\lambda\upsilon$. On regarde aussi quelques propriétés essentielles qui sont demandées pour n'importe quel calcul avec substitutions explicites qui implémente un autre calcul où la substitution est au niveau du langage du discours. La première de ces propriétés est la confluence; on utilise pour la démontrer la méthode d'intérpretation [Har89]. La deuxième est la préservation de la normalisation forte. On utilise pour la démontrer une technique due à Bloo et Geuvers [BH98]. Cette propriété est l'un des ingrédients essentiels dans n'importe quel implémentation via des substitutions explicites, surtout s'il y a quelque forme d'interaction entre les substitutions comme dans notre cas.

Partie III: de la réécriture d'ordre supérieur à la réécriture du premier ordre

La partie III de la thèse concerne la traduction de la réécriture d'ordre supérieur à la réécriture du premier ordre modulo une théorie équationnelle. La réécriture (de termes) d'ordre supérieur concerne la transformation de termes en présence de mécanismes de liaison pour les variables et des substitutions. Sa théorie a commencé avec le travail pionnier de J.W. Klop en 1980 dans sa thèse [Klo80]. L'exemple paradigmatique de système de réécriture d'ordre supérieur est le λ-calculus. La notion de substitution dans ce calcul est une opération méta qui peut être vue comme la conséquence de l'existence d'un symbole spécial appelé *symbole de liaison* qui a le pouvoir de lier des variables dans les termes. Ceci a pour conséquence que la substitution ne peut pas être considérée comme la notion usuelle de remplacement du premier ordre, mais plutôt comme une opération qui doit respecter le statut (libre ou lié) de chaque variable. Dans ce sens, il est juste de dire que la substitution est un 'remplacement respectueux'. Cependant, il serait erroné de considérer la substitution comme un concept trivial : la théorie de la réécriture d'ordre supérieur est considérablement plus compliqué que celle du premier ordre.

Plusieurs formalismes de réécriture d'ordre supérieur (HORS) existent, la recherche dans ce domaine est actuellement très active. Dans le travail fondateur de J.W. Klop [Klo80] les *Combinatory Reduction Systems* (*CRS*) ont été introduits. Plusieurs formalismes ont été définis plus tard : les *Expression Reduction Systems* (*ERS*) de Z. Khasidashvili [Kha90], les *Higher-Order Rewrite Systems* (*HRS*) de T. Nipkow [Nip91], les *Higher-Order Term Rewrite Systems* D.A. Wolfram [Wol93], les *Higher-Order term Rewriting Systems* de V. van Oostrom et F. van Raamsdonk [OR94] qui recouvrent plusieurs autres formalismes [Oos94, Raa96] et les *Explicit Reduction Systems* (*XRS*) de B. Pagano [Pag98] qui utilisent les indices de De Bruijn. La thèse de F. van Raamsdonk présente une étude d'ensemble dans ce domaine [Raa96].

Même si au niveau méta l'exécution d'une substitution est toujours atomique, le coût de son calcul dépend de la forme des termes, en particulier si la capture de variables doit être évitée en utilisant des renommage de variables liées (α-conversion). En conséquence, il y a un intérêt bien pratique pour essayer d'éviter l'α-conversion, car n'importe quelle implantation d'un système de réécriture d'ordre supérieur doit inclure des instructions concrètes pour appliquer des substitutions. Comme nous l'avons déjà mentionné, il y a une technique standard introduite par De Bruijn, appelée la notation des indices de De Bruijn, pour éviter la α-conversion. La représentation des variables via des indices élimine complètement la capture de variables. Cependant, les formalismes de De Bruijn ont été étudiés uniquement pour quelques systèmes particuliers (et uniquement au niveau des termes) et il n'y pas de formalisme général de réécriture d'ordre supérieur avec indices. Nous étudions ce problème dans cette thèse en ne nous focalisant pas uniquement sur les termes (comme il est usuellement fait dans la littérature pour le λ-calcul [KR98]) mais aussi sur les méta-termes, qui sont les objets syntaxiques utilisés pour exprimer un système de réécriture d'ordre supérieur. Plus précisément, nous introduisons une notation de De Bruijn pour les *ERS*, en obtenant ainsi la classe $SERS_{db}$. En effet, nous formulons une version simplifiée des *ERS* que nous appelons *Simplified ERS* (*SERS*), et ensuite nous considérons une notation de De Bruijn pour ce formalisme. La raison du choix du formalisme *ERS* est que sa syntaxe est plus proche du λ-calcul. Ainsi la règle de réécriture β s'écrit comme $app((\lambda x.M), N) \to M[x \leftarrow N]$ où M et N dénotent deux termes quelconques.

Le formalisme *SERS* peut être vu comme l'*interface* d'un langage de programmation fondé sur la réécriture d'ordre supérieur. Comme l'utilisation de formalismes fondés sur les variables avec noms sont nécessaires pour l'interaction des humains avec les ordinateurs d'une manière amicale, les ressources techniques tels que les indices de De Bruijn (et, plus tard, les substitutions explicites) ne doivent pas être visibles, autrement dit, elles doivent être considerées comme une décision d'implantation. Un point clé sera l'étude détaillée de la relation entre les *SERS* et les $SERS_{db}$. Les définitions dévelopées dans ce chapitre fournissen des traductions de la syntaxe d'ordre supérieure avec noms vers celui des indices et vice-versa. Ces traductions sont des extensions à l'ordre supérieur des traductions qui ont été présentées dans [Cur93], et aussi étudiées dans [KR98].

Quant aux formalismes de réécriture d'ordre supérieure basés sur des indices de De Bruijn, il y a au moins, à la connaissance de l'auteur, trois classes : les *Explicit CRS* [BR96], les *Explicit Reduction Systems* (*XRS*) [Pag98], et le *Calculus of Indexed Names and Named Indices* (*CINNI*) [Ste00]. Dans [BR96] des substitutions explicites à la λx [Ros92, Blo97] sont ajoutées au formalisme *CRS* comme un premier pas vers l'utilisation de la réécriture d'ordre supérieure avec des substitutions explicites pour la modélisation de l'exécution des programmes fonctionnels d'une façon fidèle. Comme ceci est fait dans un environnement de variables avec noms, l'α-conversion doit être prise en compte dans ces systèmes. La classe des *XRS* de B. Pagano constitue le premier formalisme de réécriture d'ordre supérieur qui utilise la notation des indices de De Bruijn et des substitutions explicites. Ils sont présentés comme une généralisation du $\lambda\sigma_{\Uparrow}$-calcul [CHL96] mais aucune connexion avec des formalismes bien établis comme les *CRS*, les *ERS* et les *HRS* á été établie. En effet, l'expression de règles naturelles des *SERS* dans le formalisme des *XRS* n'est pas triviale. On peut considérer comme exemple un système de réécriture pour des expressions logiques tel que si $imply(e_1, e_2)$ se réduit à une constante *true* alors e_1 implique logiquement à e_2. Une règle de réécriture possible serait :

$$imply(\exists x \forall y M, \forall y \exists x M) \to_{imp} true$$

Un essaie naïf pour représenter cette règle dans un *XRS* pourrait être :

$$imply(\exists\forall M, \forall\exists M) \to_{imp_{db}} true$$

Mais elle n'a pas l'effet désiré parce que $\exists\forall M$ et $\forall\exists M$ correspondent à $\exists x\forall y M$ et $\forall x\exists y M$ mais $\forall x\exists y M$ et $\forall y\exists x M$ ne sont pas équivalents. Observer que même si nous incorporons des substitutions explicites aux *XRS*, ce problème se manifeste déjà au niveau des indices de De Bruijn. Autre exemple qui peut être intéressant est la règle qui exprime l'extensionalité fonctionnelle η :

$$\lambda x.(app(M,x)) \to M \quad \text{si } x \text{ n'apparaît pas libre dans } M$$

qui est usuellement exprimée dans un système fondé sur des indices de De Bruijn et substitutions explicites par la règle η_{db} suivante :

$$\lambda(app(M,1)) \to N \quad \text{si } M =_{\mathcal{E}} N[\uparrow]$$

où $M =_{\mathcal{E}} N$ signifie que M et N sont équivalents modulo la théorie des substitutions explicites $\mathcal{E}$ (on peut prendre par exemple v). Ni la règle *imp* ni η_{db} peuvent être exprimées dans le formalisme *XRS*. Ils n'ont donc pas, en principe, le même pouvoir d'expression que les *ERS*. Quant aux systèmes de M-O.Stehr [Ste00] mentionnés plus haut le même problème apparaît : aucune relation n'est établie avec les systèmes de réécriture d'ordre supérieur.

Systèmes de réduction d'expressions simplifiées

Le chapitre 6 de la thèse introduit le formalisme de réécriture d'ordre supérieur de variables avec noms appelé *SERS*. Ce formalisme est une simplification convenable des *ERS* de Khasidashvili [Kha90] qui consiste à restreindre les symboles de liaison aux symboles qui lient une seule variable et à restreindre la substitution à la substitution simple (en opposition avec la substitution simultanée ou parallèle).

Un exemple d'un *SERS* est le λ-calculus, obtenu avec la signature qui contient le symbole fonctionnel *app* et le symbole lieur λ, aussi comme les *SERS*-règles de réécriture : $app(\lambda\alpha.X, Z) \to_\beta X[\alpha \leftarrow Z]$. Observer que nous avons des symboles de fonction (tel que *app*), des symboles de liaison (tel que λ), des méta-variables pour les termes (telles que X et Z) et des méta-variables pour les variables lié (telle que α). Nous avons aussi des méta-variables pour les variables libres. Tant $app(\lambda\alpha.X, Z)$ comme $X[\alpha \leftarrow Z]$ sont appelés ***méta-termes*** et sont utilisés pour définir des règles de réécriture. L'expression $\bullet[\bullet \leftarrow \bullet]$ est appelée l'***opérateur de méta-substitution*** et représente une substitution suspendue.

Un autre exemple est le λx-calcul [BR96, Ros92]. Il est définit en considérant la signature qui contient les symboles fonctionnels $\{app, subs\}$ et les symboles de liaison $\{\lambda, \sigma\}$, avec les *SERS*-règles de réécriture suivantes :

$$\begin{array}{lll}
app(\lambda\alpha.X, Z) & \to_{Beta} & subs(\sigma\alpha.X, Z) \\
subs(\sigma\alpha.(app(X,Y)), Z) & \to_{App} & app(subs(\sigma\alpha.X, Z), subs(\sigma\alpha.Y, Z)) \\
subs(\sigma\alpha.\lambda\beta.(X), Z) & \to_{Lam} & \lambda\beta.(subs(\sigma\alpha.X, Z)) \\
subs(\sigma\alpha.\alpha, Z) & \to_{Var} & Z \\
subs(\sigma\alpha.\widehat{\beta}, Z) & \to_{Varf} & \widehat{\beta}
\end{array}$$

Le $\widehat{\beta}$ dans la dernière règle représente une variable libre (c'est un exemple d'une méta-variable pour les variables libres), donc elle ne peut pas recevoir en assignation la même variable que celle assignée à α.

Les règles de réécriture sont instanciées à l'aide des ***valuations*** pour pouvoir obtenir la relation de réécriture sur les termes. Une valuation assigne tout simplement des termes aux méta-variables pour les termes, des variables aux méta-variables pour les variables et elle exécute les substitutions suspendues qui sont représentées par l'opérateur de méta-substitution. Bien sûr, tout cela doit être fait avec attention, l'ensemble des valuations ***admissibles*** est identifié et ce sont uniquement ces valuations qui peuvent être utilisées pour instancier des règles de réécriture. Par exemple, le $\lambda\eta$-calcul est obtenu en ajoutant la *SERS*-règle de réécriture suivante au λ-calculus : $\lambda\alpha.(app(X,\alpha)) \to_\eta X$. Une valuation admissible serait une valuation telle que la variable assignée à α n'apparaît pas libre dans le terme assigné à X.

Systèmes de réduction d'expressions simplifiées avec des indices

On introduit le système de réécriture d'ordre supérieur avec indices de De Bruijn, appelé *SERS*$_{db}$, en utilisant des exemples. En particulier, nous considérons d'abord le cas du lambda calcul avec des indices de De Bruijn : $app(\lambda X_\alpha, Z_\epsilon) \to X_\alpha\llbracket Z_\epsilon\rrbracket$. Les expressions $app(\lambda X_\alpha, Z_\epsilon)$ et $X_\alpha\llbracket Z_\epsilon\rrbracket$ sont appelés ***méta-termes de De Bruijn***. Bien qu'il y n'a pas plus de méta-variables pour les variables liés nous avons encore des méta-variables pour les termes (telle que X_α) et des méta-variables pour les variables libres (voir $\widehat{\beta}$ dans l'exemple dessous). Noter que maintenant les méta-variables pour les termes portent une étiquette, et que ces étiquettes forment une partie intégrale de ces méta-variables. Une méta-variable de la forme X_l indique qu'elle apparaît dessous un nombre d'opérateurs de liaison, notamment un pour chaque symbole dans l'étiquette l. L'opération $\bullet[\bullet]$ à droite est nommée *l'opérateur de méta-substitution de De Bruijn* et représente une substitution de De Bruijn suspendue.

Un deuxième exemple est le système suivant qui est obtenu en traduisant le λx-calcul à l'aide de la traduction que nous développons dans le chapitre 6 de la thèse :

$$\begin{array}{lcl}
app(\lambda X_\alpha, Z_\epsilon) & \to & subs(\sigma X_\alpha, Z_\epsilon) \\
subs(\sigma(app(X_\alpha, Y_\alpha)), Z_\epsilon) & \to & app(subs(\sigma X_\alpha, Z_\epsilon), subs(\sigma Y_\alpha, Z_\epsilon)) \\
subs(\sigma(\lambda(X_{\beta\alpha})), Z_\epsilon) & \to & \lambda(subs(\sigma(X_{\alpha\beta}), Z_\beta)) \\
subs(\sigma(1), Z_\epsilon) & \to & Z_\epsilon \\
subs(\sigma(\mathrm{S}(\widehat{\beta})), Z_\epsilon) & \to & \widehat{\beta}
\end{array}$$

La règle $subs(\sigma(\lambda(X_{\beta\alpha})), Z_\epsilon) \to \lambda(subs(\sigma(X_{\alpha\beta}), Z_\beta))$ est intéressante parce qu'elle montre l'utilisation de la commutation des symboles de liaison (confronter $X_{\beta\alpha}$ et $X_{\alpha\beta}$) et elle illustre en même temps comment quelque sorte d'ajustement sera nécessaire pour aller de Z_β vers Z_ϵ.

En effet, des valuations seront nécessaires pour instancier des règles de réécriture et obtenir de cette façon la relation de réduction sur les termes. Ces valuations doivent respecter les étiquettes des méta-variables. Considérons pour l'instant la *SERS*-règle de réécriture :

$$\xi\alpha.(\xi\beta.X) \to_r \xi\beta.(\xi\alpha.X)$$

et son traduction dans les formalisme $SERS_{db}$:

$$\xi(\xi(X_{\beta\alpha})) \to_{r_{db}} \xi(\xi(X_{\alpha\beta}))$$

Une valuation est dite *valide* si elle respecte les étiquettes des méta-variables. Par exemple, si $X_{\beta\alpha}$ est instanciée avec l'indice 1 alors $X_{\alpha\beta}$ doit être instanciée avec l'indice 2. Quand on fait de la'instantiation des règles de réécriture de De Bruijn, uniquement des valuations valides seront utilisées.

Propriétés

Une des propriétés élémentaires qui nous intéresse est le rapport entre les formalismes *SERS* et $SERS_{db}$. Nous montrons que la réécriture dans le formalisme *SERS* peut être simulée dans celle de $SERS_{db}$ et vice-versa. Pour le premier cas nous avons besoin de définir plusieurs traductions (que nous appelons $T(\bullet)$ sans distinction) : de termes vers termes de De Bruijn, de méta-termes vers méta-termes de De Bruijn et de valuations vers valuations de De Bruijn. Après nous étudions des propriétés de base de ces traductions. Comme nous l'avons déjà mentionné, le dernier exemple en haut est obtenu en traduisant le λx-calcul comme un *SERS*. Une fois que ces traductions sont fixées nous pouvons montrer que si s se réécrit en t dans le formalisme *SERS* en utilisant la règle de réécriture (G, D), alors $T(s)$ se réécrit en $T(t)$ dans le formalisme $SERS_{db}$ en utilisant la règle de réécriture De Bruijn $T(G, D)$.

Quant au deuxième point on procède d'une façon similaire pour obtenir les traductions $U(\bullet)$. Pourtant, ce point a besoin d'un travail plus technique que le précédent puisque la traduction d'un terme de De Bruijn (ou un mèta-terme de De Bruijn) peut ne pas donner un terme unique (ou méta-terme). Dans le cas où deux termes différents sont obtenus ils seront α-équivalents (ou v-équivalents - une notion définie dans le chapitre 6 - dans le cas des méta-termes). Une question additionnelle qui nous intéresse est la garantie que les valuations de De Bruijn valides soient traduites vers des valuations admissibles dans le formalisme *SERS*. En fin, nous montrons que si a se réécrit dans b dans le formalisme $SERS_{db}$ en utilisant la règle de réécriture (L, R), alors $U(a)$ se réécrit dans $U(b)$ dans le formalisme *SERS* en utilisent la règle $U(L, R)$.

La partie finale du chapitre 6 étudie la rapport entre les traductions mentionnées dans le paragraphe précédent. Celle ci donne lieu à deux résultats qui disent, respectivement, qu'étant donné un méta-terme M alors $U(T(M))$ est équivalent à M (dans un sens précis, voir la section 6.1 de la thèse), et qu'étant donné un méta-terme de De Bruijn A alors $T(U(A))$ est identique à A. Ces résultats sont utilisés pour montrer que la confluence est préservée en traduisant un système de réécriture *SERS* vers un système de réécriture $SERS_{db}$. Plus précisément nos montrons que, d'un coté, si $\mathcal{R}$ est un *SERS* qui est confluent alors $T(\mathcal{R})$ est un $SERS_{db}$ confluent aussi. D'autre côté, nos montrons que si $\mathcal{R}$ est un $SERS_{db}$ confluent alors $U(\mathcal{R})$ est un *SERS* confluent.

De la réécriture d'ordre supérieur vers la réécriture de premier ordre

Comme nous l'avons déjà dit ci-dessus, l'opération de substitution ne peut pas être cadrée comme une opération simple de remplacement telle que la substitution dans les théories du premier ordre. En conséquence, des chercheurs se sont intéressés par la formalisation de la substitution d'ordre supérieure à l'aide de ***substitutions explicites***, de telle façon que les formalismes/systèmes d'ordre supérieur soient exprimables dans des formalismes/systèmes du premier ordre : la notion de variable liée n'existe plus et la substitution devient du remplacement. Un exemple bien connu de la combinaison d'indices de De Bruijn et de substitutions explicites est la formulation des différents calculs du premier ordre pour le λ-calculus [ACCL91, BBLRD96, KR95, DG01]. D'autres exemples sont les traductions de l'unification d'ordre supérieur vers la unification du premier ordre modulo [DHK00], la logique d'ordre supérieur vers celle du premier ordre modulo [DHK01], la démonstration automatique d'ordre supérieur vers celle du premier ordre modulo [DHK98], etc.

Le cas du λ-calculus est intéressant mais en même temps pas toute á fait représentatif des problèmes que l'on peut trouver quand on fait du codage de systèmes d'ordre supérieur vers le premier ordre. La raison c'est que dans ce cas particulier il suffit de se débarrasser de l'α-conversion et de promouvoir la substitution du niveau méta au niveau objet. En effet, le remplacement des variables usuelles par des indices de De Bruijn et l'introduction de substitutions explicites suffisent pour rendre un système du premier ordre, tels que les exemples précédents le montrent. Pourtant, c'est ne pas le cas pour des systèmes de réécriture d'ordre supérieur arbitraires. Cette à dire, l'élimination de l'α-conversion et l'introduction de substitutions explicites n'est pas suffisante pour obtenir un système simple (dans le sens de la réécriture du premier ordre modulo une théorie équationelle vide). La raison est que dans la réécriture d'ordre supérieur[5] le *LHS* des règles de réécriture sont des motifs d'ordre supérieur [Nip91, Oos94]. En conséquence il faut coder *aussi* le filtrage des motifs d'ordre supérieur quand on se dirige vers le formalisme du premier ordre. Un exemple simple de cet fait est le cas de la η_{db}-règle de réécriture :

$$\lambda(app(X_\alpha, 1)) \rightarrow_{\eta_{db}} X_\epsilon$$

Noter que X_ϵ du membre droit de la règle, qui n'apparaît pas dans un contexte de liaison, est en relation avec l'occurrence de X_α à gauche, que cette fois ci apparaît sous un contexte de liaison. Ceci peut être vu comme la raison pour laquelle la règle η_{db} avait reçu tellement d'attention [Río93, Har92, Bri95, Kes96], filtrage syntaxique ne suffit donc pas. On peut bien dire que le test d'occurrence est une caractéristique du filtrage d'ordre supérieur qui ne peut pas être traité au premier ordre. Dans l'exemple de la règle η_{db} le lecteur peut vérifier que le terme $\lambda(app(3, 1))$ se réécrit en 2. Dans un formalisme de réécriture du premier ordre avec des substitutions explicites on a la formulation alternative :

$$\lambda(app(X[\uparrow], 1)) \rightarrow X$$

Cependant, pour vérifier que le terme du premier ordre 3 soit de la forme $X[\uparrow]$ le filtrage du premier ordre ne suffit plus : nous avons besoin d'$\mathcal{E}$-filtrage, c'est à dire, filtrage modulo une théorie équationnelle $\mathcal{E}$. Étant donné un calcul de substitutions $\mathcal{E}$ nous aurons besoin de résoudre l'équation $3 \overset{?}{=}_{\mathcal{E}} X[\uparrow]$.

Un autre exemple, peut être moins évident, est obtenu par la règle de commutation C :

$$imply(\exists\alpha.\forall\beta.X, \forall\beta.\exists\alpha.X) \rightarrow true$$

qui exprime le fait que la formule qui apparaît dans le premier argument de la fonction *imply* implique le deuxième argument. La traduction naïve vers le premier ordre, notamment $imply(\exists(\forall(X)), \forall(\exists(X))) \rightarrow true$, n'est évidemment pas correcte. Autant nous prendrons son codage dans le formalisme des indices de De Bruijn $SERS_{db}$ et après nous le traduirons vers le premier ordre en utilisant la conversion qui est présentée dans le chapitre 7 de la thèse en obtenant C_{fo} :

$$imply(\exists(\forall(X)), \forall(\exists(X[2 \cdot 1 \cdot \uparrow^2]))) \rightarrow true$$

Maintenant, la règle C_{fo} a exactement la signification que l'on attends.

Le but du chapitre 7 est de donner un algorithme de conversion, appelé la *Procédure de Conversion*, qui permet de coder la réécriture d'ordre supérieur dans la réécriture du premier ordre modulo une théorie équationelle $\mathcal{E}$. Ceci est intéressant du point de vue théorique parce que le pouvoir d'expression des systèmes de réécriture d'ordre supérieur et du premier ordre n'est pas le même. Pourtant, un sujet plus pratique se

[5] Dans le formalisme *SERS* les *LHS* sont toujours des motifs d'ordre supérieur, mais c'est ne pas le cas pour d'autres formalismes comme par exemple les *HRS*.

manifeste, la possibilité de *transférer* des résultats développés dans les systèmes du premier ordre vers ceux d'ordre supérieur. Dans le chapitre 8 nous transférons le Théorème de Standardisation de la réécriture du premier ordre vers celle d'ordre supérieur. Des techniques concernant la confluence, terminaison, complétion, des stratégies d'évaluation, etc., doivent être étudiées. Ce n'est pas encore clair comment on peut transférer des techniques telles que dependency pairs [AG00], semantic labelling [Zan95] ou complétion [BD88] vers la réécriture d'ordre supérieur. Même les techniques qu'ont été déjà formulées pour l'ordre supérieur comme le RPO pour les systèmes d'ordre supérieur [JR99], sont beaucoup plus compliquées que dans ses versions du premier ordre correspondantes [Der82, KL80]. Nous obtenons aussi une caractérisation de la classe des $SERS_{db}$ (qui inclut le λ-calculus) telle que chaque $SERS_{db}$ peut être traduit à un système de réécriture du premier ordre complet ($\mathcal{E} = \emptyset$). Nous argumentons que ces systèmes, appelés *$SERS_{db}$essentiellement du premier ordre* , sont plus convenables pour transférer des propriétés.

Un commentaire final sur la Procédure de Conversion concerne le fait que nous n'ajouterons pas un calcul de substitutions explicites concret à cette procédure. Nous avons plutôt choisit de travailler avec une formulation abstraite des calcul de substitutions explicites comme il a été fait dans [Kes96, Kes00] pour traiter la confluence de plusieurs lambda calculs avec substitutions explicites. En conséquence, la méthode que nos proposons peut être utilisée avec plusieurs calculs de substitutions explicites tels que σ [ACCL91], $\sigma_{\Uparrow}$ [HL89], v [BBLRD96], f [Kes96], d [Kes96], s [KR95], χ [LRD95].

La procédure de conversion

La section 7.2 de la thèse introduit le formalisme du premier ordre appelé "Explicit Expression Reduction Systems" (*ExERS*) utilisé pour traduire les systèmes de réécriture d'ordre supérieur fondés sur les indices de De Bruijn aux systèmes du premier ordre. Un *ExERS* est un système de réécriture du premier ordre contenant :

- Un ensemble de règles de réécriture propres gouvernant le comportement des symboles de fonction et de liaison dans la signature.

- Un ensemble de règles de réécriture des substitutions, appelé le *calcul de substitution* gouvernant le comportement des symboles de substitution dans la signature, et utilisé pour propager et exécuter/éliminer les substitutions.

Un ensemble de règles de réécriture arbitraires ne remplit pas nécessairement les conditions requises par un calcul de substitutions. Pour cela nous donnons une présentation générale fondée sur des macros. N'importe quelle instance de ce calcul, obtenue en associant des opérateurs de substitution fixés à ces macros, sera donc considérée un calcul de substitutions. Des propriétés supplémentaires imposées à ces calculs fourniront ce que nous appelons *Basic Substitution Calculi* qui sera dénoté par $\mathcal{W}$. Cette idée a été introduite par D. Kesner [Kes96, Kes00] pour donner une preuve unique de confluence comprenant toute une série de calculs de substitutions explicites fondés sur des indices de De Bruijn. Nous bénéficions ainsi de ce fait en pouvant réduire la réécriture d'ordre supérieur à un cadre du premier ordre où le calcul de substitutions peut être n'importe quel calcul de substitutions explicites qui s'adapte à notre présentation fondée sur des macros. Nous ne sommes donc pas obligés de restreindre notre étude à un calcul particulier de substitutions explicites.

La réécriture dans un *ExERS* $\mathcal{R}_{\mathcal{W}}$ est tout simplement la réécriture dans un système du premier ordre $\mathcal{R}$ modulo $\mathcal{W}$-égalité. Cependant, nous fournissons aussi un sous-ensemble des *ExERS* appelé *sytèmes de premier ordre simples* (*FExERS*), dénotés aussi $\mathcal{R}_{\mathcal{W}}$ où $\mathcal{R}$ est un système de réécriture du premier ordre dans lequel la réécriture est donnée tout simplement par les règles de $\mathcal{R} \cup \mathcal{W}$. Pour qu'un *ExERS* remplisse les conditions d'un *FExERS* nous exigeons que les *LHS*s des règles ne contiennent aucune occurrence de l'opérateur de substitution. Par exemple, si $\mathcal{W}$ est un calcul de substitution de base tel que le σ-calcul et $\mathcal{R} = \{app(\lambda X, Y) \rightarrow_{\beta_{db}} X[cons(Y, id)]\}$ alors $\mathcal{R}_{\mathcal{W}}$ est un *FExERS*, et si $\mathcal{R}' = \mathcal{R} \cup \{\lambda(app(X[shift], 1)) \rightarrow_{\eta_{db}} X\}$, $\mathcal{R}'_{\mathcal{W}}$ est un *ExERS*. Alors nous avons $1[app(\lambda 1, c) \cdot id] \rightarrow_{\mathcal{R}_\sigma} 1[1[c \cdot id] \cdot id]$. Aussi, $\lambda(app(3, 1)) \rightarrow_{\mathcal{R}'_\sigma} 2$. La dernière réduction est obtenue en remarquant que $\lambda(app(3, 1)) =_\sigma \lambda(app(2[\uparrow], 1)) \rightarrow_{\eta_{db}} 2$.

Nous présentons brièvement des exemples d'applications de la *Procédure de Conversion*, un algorithme pour traduire un système de réécriture d'ordre supérieur dans le formalisme $SERS_{db}$ vers un *ExERS* du premier ordre. La Procédure de Conversion est assez compliquée puisque plusieurs conditions, essentiellement par rapport aux étiquettes des méta-variables, doivent être vérifiées pour qu'une valuation puisse être admise comme *valide*. On peut considérer, par exemple, la η_{db}-règle $\lambda(app(X_\alpha, 1)) \rightarrow X_\epsilon$. La condition sur les valuations $SERS_{db}$ pour participer à la relation de réécriture induite sur les termes est qu'elle soit valide, comme nous le présentons au chapitre 6. La validité assurera, dans ce cas, que la mèta-variable X_α ne soit pas instanciée avec l'indice

1. La Procédure de Conversion devra codifier cette condition dans le cadre du premier ordre. L'idée est de remplacer toutes les occurrences des méta-variables X_l par une variable du premier ordre X suivie d'une *substitution explicite d'actualisation* des indices appropriées qui calcule les valuations valides. Alors, le résultat serait : $\lambda(app(X[shift], 1)) \to X$. Cependant, celui-ci est un exemple simple, mais dans la situation générale l'ajout des macros *shift* ne sera pas suffisant. Un témoin de ce fait est la règle de commutation de symboles de liaison C-règle que nous avons vu plus haut.

Voici quelques exemples de conversion de règles où nous avons fixé $\mathcal{W}$ comme le σ-calcul. Nous encourageons le lecteur à se référer au chapitre 7 de la thèse pour des détails supplémentaires.

$SERS_{db}$-règle de réécriture	*conversion*
$\lambda(app(X_\alpha, 1)) \to X_\epsilon$	$\lambda(app(X[\uparrow], 1)) \to X$
$\lambda(\lambda(X_{\alpha\beta})) \to \lambda(\lambda(X_{\beta\alpha}))$	$\lambda(\lambda X) \to \lambda(\lambda(X[2 \cdot 1 \cdot (\uparrow \circ \uparrow)]))$
$f(\lambda(\lambda(X_{\alpha\beta})), \lambda(\lambda(X_{\beta\alpha}))) \to \lambda(X_\gamma)$	$f(\lambda(\lambda(X[\uparrow \circ \uparrow])), \lambda(\lambda(X[\uparrow \circ \uparrow]))) \to \lambda(X[\uparrow])$
$app(\lambda X_\alpha, Z_\epsilon) \to_{\beta_{db}} X_\alpha[\![Z_\epsilon]\!]$	$app(\lambda X, Z) \to X[Z \cdot id]$

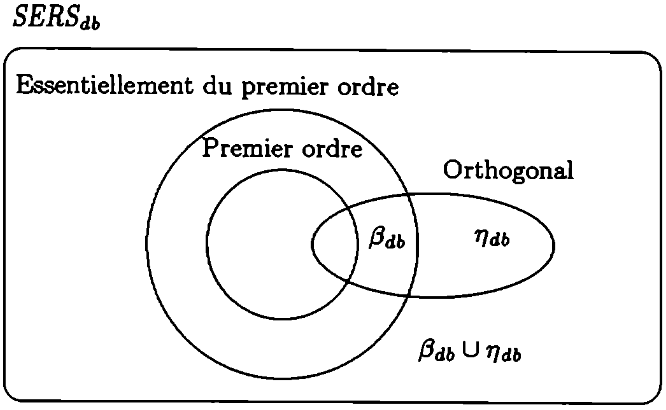
Le système résultant de la Procédure de Conversion est codé comme un *ExERS*. Dans quelques cas ce dernier système peut être codé comme un *FExERS* où la réduction est définie sur les termes du premier ordre et le filtrage est tout simplement celui du premier ordre (filtrage syntaxique), en arrivant à un système *de premier ordre simple.*

Propriétés de la procédure de conversion et des systèmes essentiellement du premier ordre

Nous étudions aussi la connexion entre la réécriture d'ordre supérieur et la réécriture du premier ordre modulo : la *Proposition de Simulation* dit que tout pas de réécriture d'ordre supérieur peut être simulé ou implanté par la réécriture de premier ordre, et la *Proposition de Projection* dit que les pas de réécriture dans la version du premier ordre d'un système $\mathcal{R}$ d'ordre supérieur peuvent être projetés dans $\mathcal{R}$. La Proposition de Simulation établit que si a se réécrit en b dans un $SERS_{db}$ $\mathcal{R}$ alors a aussi se réécrite en b dans la version du premier ordre $fo(\mathcal{R})_\mathcal{W}$. La Proposition de Projection établit que les dérivations dans un *ExERS* ou *FExERS* $fo(\mathcal{R})_\mathcal{W}$ peuvent être projetées dans des dérivations dans $\mathcal{R}$: si a se réécrit en b dans $fo(\mathcal{R})_\mathcal{W}$ alors $\mathcal{W}(a)$ se réécrit en $\mathcal{W}(b)$ dans $\mathcal{R}$ où $\mathcal{W}(a)$ est la $\mathcal{W}$ forme normale de a. Ceci assure que nous n'ajoutons pas des dénuées de sens dans le système traduit au premier ordre. Des propriétés supplémentaires des dérivations projetées sont étudiées au chapitre 8, où des dérivations standard sont considérées.

Finalement nous fournissons un critère très simple appelé *condition-fo* que peut être utilisé pour décider si un système de réécriture d'ordre supérieur peut être traduit dans un système de réécriture de premier ordre simple (i.e. modulo une théorie équationnelle vide). En particulier, nous pouvons vérifier que plusieurs calculs d'ordre supérieur dans la littérature, tel que le lambda calcul, satisfont cette propriété. Comme le lecteur peut remarquer d'après le chapitre 7 où la condition-fo est définie en détail, plusieurs résultats concernant les systèmes d'ordre supérieur (ex. perpétuité [KOO01a], standardisation [Mel96]) exigent *linéarité à gauche* (une méta-variable peut apparatre au plus une fois dans le *LHS* d'une règle), et *complètement étendu ou locale* (si une méta-variable $X(t_1, \ldots, t_n)$ apparat dans le *LHS* d'une règle de réécriture alors $t_1, \ldots, t_n$ est la liste des variables liées par dessus). Le lecteur peut trouver intéressant de remarquer aussi que ces conditions ensemble semblent impliquer la condition-fo. Une preuve de ce fait entranerait le développement de résultats dans les systèmes de réécriture d'ordre superieur ou via une traduction convenable au formalisme $SERS_{db}$; nous le laissons comme travail futur.

Bien sûr, tous les systèmes de réécriture du premier ordre sont des $SERS_{db}$ essentiellement du premier ordre, d'où ces dernières systèmes ne sont pas nécessairement linéaire à gauche. Aussi, un $SERS_{db}$ orthogonal n'est-il pas nécessairement essentiellement du premier ordre, l'exemple principal de ce fait étant le système dont la seule règle est η_{db}. Nous illustrons cette situation.



Il nous semble juste de dire, d'une manière informelle, qu'un système $SERS_{db}$ est essentiellement du premier ordre si le filtrage de motifs d'ordre supérieur peut être réduit au filtrage syntaxique du premier ordre. Nous soutenons que les systèmes $SERS_{db}$ essentiellement du premier ordre sont appropriées pour transférer des résultats des systèmes du premier ordre. Comme évidence de notre thèse nous entreprenons dans le chapitre 8 la tâche de transférer une propriété non triviale des systèmes de réécriture du premier ordre (linéaire à gauche) à cette classe de systèmes: le Théorème de Standardisation.

Transfert de la standardisation

La procédure de conversion est intéressante d'au moins deux points de vue. De la perspective d'expressivité elle établit comment la réécriture d'ordre supérieur peut être codée comme la réécriture du premier ordre modulo une théorie équationnelle, et d'ailleurs elle caractérise un sous-ensemble des $SERS_{db}$ pour lesquelles la théorie équationnelle est vide. De la perspective pratique elle ouvre la possibilité de transférer des résultats du cadre du premier ordre ver celui d'ordre supérieur. Le chapitre 8 de la thèse essaye de poursuivre cette dernière perspective plus en détail. Exactement, nous étudions comment lever le Théorème de Standardisation du premier ordre à l'ordre supérieur.

Concretement, nous montrons le Théorème de Standardisation pour la classe des systèmes de réécriture d'ordre supérieur (linéaire à gauche) qui sont essentiellement du premier ordre. Ceci prouve que certaines techniques développées pour le premier ordre sont applicables à la classe des systèmes de réécriture d'ordre supérieur qui sont essentiellement du premier ordre. Nous retrouvons une notion similaire (même un peu plus forte) à celle de système d'ordre supérieur essentiellement du premier ordre dans l'étude de stratégies perpetuelles [KOO01b] aussi bien que dans l'étude de standardisation axiomatique [Mel96] lorsqu'on regarde les conditions imposées au formalisme d'ordre supérieur pour que les preuves fonctionnent.

Le transfert de la standardisation est accompli en utilisant des idées dues à P-A. Melliès. En effet, dans [Mel00] il montre le résultat suivant : toute dérivation standard v de M à N dans $\lambda\sigma$ où N est en σ- forme normale est projetée sur une dérivation standard $\sigma(v)$ de $\sigma(M)$ à N dans le λ-calculus. Nous montrons que, en effet, ceci est vrai non seulement pour le λ-calculus, mais pour tous les systèmes essentiellement du premier ordre, dans lequel nous retrouvons bien evidemment le λ-calculus. Nous baptisons ce résultat le *transfert de standardisation généralisé* : si $\mathcal{R}$ est un $SERS_{db}$ linéaire à gauche et essentiellement du premier ordre alors toute dérivation standard v de M à N dans $fo(\mathcal{R})_\sigma$ où N est en σ-forme normale est projetée sur une dérivation standard $\sigma(v)$ de $\sigma(M)$ à N dans $\mathcal{R}$.

La procédure résultante pour standardiser une $SERS_{db}$-dérivation Υ consiste en :

1. "Implanter" la $SERS_{db}$-dérivation Υ comme une *FExERS*-dérivation v en utilisant la Proposition de Simulation.

2. Appliquer la standardisation du premier ordre à v [Bou85] pour obtenir une dérivation ϕ du premier ordre standard.

3. Projeter la dérivation ϕ en utilisent la Proposition de Projection pour obtenir $\sigma(\phi)$. Utiliser ensuite le résultat du transfert de standardisation généralisé pour conclure que $\sigma(\phi)$ est une dérivation standard dans le cadre de l'ordre supérieur.

Nous pouvons signaler qu'un bénéfice supplémentaire de notre résultat est la possibilité d'étudier la théorie des dérivations "nécessaires" pour les systèmes linéaire à gauche et non-orthogonaux en appliquant la technique développées dans [Mel00]. Ceci est laissé comme travail futur.

En montrant à travers un exemple concret (le théorème de standardisation) comment transférer des résultat du premier ordre vers l'ordre supérieur, notre contribution nous pousse encore plus à l'étude des propriétés d'ordre supérieur a travers leurs images au premier ordre. Notre traduction des systèmes d'ordre supérieur vers le premier ordre ouvre la porte a une nouvelle approche technique pour comprendre les systèmes d'ordre supérieur.

A la memoria de María Elena Ferrando y Augusto Horacio Castro

Acknowledgments

I cannot but begin expressing the deepest of gratitudes to my thesis supervisor, Delia Kesner. I couldn't possibly list all the reasons so let me just say that her encouragement, acute comments, constant support and candour have been a fundamental source of inspiration for me. It has been an honour and the greatest of pleasures working with her.

Much of the work presented in this thesis was carried out in Argentina. It is thanks to the guidance of Alejandro Ríos at the University of Buenos Aires that this thesis was ever completed. His expertise in the area and his detailed comments have played a key role. It has been gratifying to work in collaboration with him.

Marie-Christine Rousset has kindly accepted to be the president of the jury, thank you. It is without doubt a privilege for me that two key figures in the study of rewriting, among other fields, such as Gilles Dowek and Vincent Van Oostrom have accepted to be rapporteurs of my thesis. Finally I would like to extend my gratitude to Pierre-Louis Curien for finding the time to be present in the day of my defense as a member of the jury.

I would also like to thank all the members of the DEMONS group of the Laboratoire de Recherche en Informatique for their hospitality, particularly Jean-Pierre Jouannaud, Laurence Puel, Delia Kesner, Frédéric Blanqui, Jean-Christoph Filliâtre, Xavier Urbain, Julien Forest and Pierre Courtieu.

Of course, a warm thanks to the LIFIA laboratory of the University of La Plata, where it all began. I extend my gratitude to Gabriel Baum, his guidance from an early stage of my studies has been instrumental. Also, when it comes to LIFIA I cannot avoid mentioning Matías Menni, Pablo Martínez López (alias Fidel), Hernán 'hache' Badenes and Esteban 'steve' de la Canal (los pichones no tan pichones).

Thanks to Irene Loisseau for her constant support and for helping me to find my way through the jungle of paperwork which hides behind the scholarships that have allowed me to invest a year at LRI. During my stay at Paris it was the warmth with which Roberto Di Cosmo and Delia Kesner received me that made me feel much more at home than I expected possible. Also, I would like to mention 'la banda del petit pavillion', my other family: Jacinta, Pablo el químico, Kuei (and his misterious healing pellets), Nacho, Gustavo el Cordobés and Elina Manchinelli (la peti).

The place: Necochea, Atlantic coast of Argentina. The time: Summer vacation of 1998. The weather: Never seen so much rain. The deed: Thank you Pablo for encouraging me to take my PhD studies, and for our friendship. Quike you've always been a part of everything.

My mother and father, Silvina and Agustina have always been there for me, gracias.

Finally, I thank my wife Jimena, for all her love and support, and Coco (el batatoide).

Eduardo Bonelli
7th of November 2001,
Orsay, France.

Contents

Chapter 1

Introduction

Computation may be seen as the task of transforming some given *input object* into some new *output object* by means of *transformation rules.* A Term Rewrite System [Klo92] is a model of computation in the sense that objects are specified as sequences of symbols and transformation rules as sets of pairs of objects captured by rewrite schemas or rules. An early example of a Term Rewrite System is the λ-calculus [Chu41], devised by A.Church in the 1930s. The objects are called λ-terms and they represent functions; the only transformation rule is the β-rewrite rule, which represents the result of applying a function to an argument. For instance, the term '$\lambda x.x$' represents the identity function, the leftmost occurrence of 'x' plays a similar role as that of the formal parameter of a function or procedure in imperative programming languages such as Pascal. The λ-term '$(\lambda x.x)$ 4' represents function application, namely the identity function applied to the representation of the number four. Note that application is represented by juxtaposition. We may compute by actually applying a function to an argument by means of the β-rewrite rule:

$$(\lambda x.M)N \quad \rightarrow_\beta \quad M\{x \leftarrow N\}$$

where M and N denote arbitrary λ-terms, and x an arbitrary variable. The symbol $\bullet\{\bullet \leftarrow \bullet\}$[1] appearing on the right-hand side (*RHS*) of the rule denotes the operation of *metalevel substitution*: $M\{x \leftarrow N\}$ stands for the λ-term resulting from replacing the occurrences of the variable x (the formal parameter) in M by N (the actual parameter). In fact, just the so called free variable occurrences of x are replaced but we shall leave these details for the moment. A λ-term of the form $(\lambda x.M)N$, where M and N denote arbitrary λ-terms and x an arbitrary variable, is called a β-redex. An example of a one-step computation using the β-rewrite rule is:

$$(\lambda x.x)4 \quad \rightarrow_\beta \quad x\{x \leftarrow 4\} = 4$$

The λ-term $(\lambda x.x)4$ is said to β-reduce or β-contract to the λ-term 4. A λ-term without occurrences of a β-redex is called a *β-normal form*, or simply a *normal form.* The λ-term 4 is a normal form.

The λ-calculus with just the β-rewrite rule may be shown to represent or encode all computable functions (formally defined as the partial recursive functions). This fact, together with its simple formulation, justifies its appeal to the Computer Science community. However, it is hard to imagine writing even simple programs by laying out λ-terms. Functional programming languages are user-friendly versions of the λ-calculus. Examples of these languages are: ML [MTH90], Haskell [HW88] and CAML [WL93]. Although they have evolved a great deal over the years, they may be seen to share the λ-calculus as a common theoretical foundation. Thus, for these languages, the λ-calculus provides a convenient test-bed for studying new language features.

Some previously irrelevant issues arise with the interest of Computer Science in λ-calculus as a basis for programming languages:

- Does computation eventually terminate for any λ-term?
- Given a λ-term with two or more β-redexes, can they be computed in any order?
- In terms of consumption of computational resources, say CPU time, is the number of β-rewrite steps a faithful measure? For example, do the following rewrite steps 'cost' the same?

$$\begin{array}{lll} (\lambda x.x)4 & \rightarrow_\beta & x\{x \leftarrow 4\} = 4 \\ (\lambda x.xx)4 & \rightarrow_\beta & (xx)\{x \leftarrow 4\} = 44 \end{array}$$

[1] We shall often use the '$\bullet$' symbol as a place holder at metalevel.

Termination or Strong Normalization, the first issue, has been studied rather thoroughly. The λ-calculus, as we have presented it above, does not enjoy termination. The simplest example[2] is the λ-term $\Delta\Delta$ where $\Delta = \lambda x.xx$.

$$(\lambda x.xx)\Delta \rightarrow_\beta (xx)\{x \leftarrow \Delta\} = \Delta\Delta \rightarrow_\beta \Delta\Delta \rightarrow_\beta \dots$$

However for restricted subsets[3] termination may be shown to hold. Let us address the second issue. The Church-Rosser Theorem is perhaps the first important syntactic result developed for the λ-calculus[4]. It states that if two λ-terms M_1 and M_2 result from some other λ-term M by a number of β-rewrite steps, then there exists some λ-term M_3 such that both M_1 and M_2 reduce via some number of β-rewrite steps to M_3 (M_1 and M_2 are said to be 'joinable'). This may be illustrated as follows:

$$\begin{tikzcd}
& M \arrow[dl, two heads, "\beta"'] \arrow[dr, two heads, "\beta"] & \\
M_1 \arrow[dr, dotted, two heads, "\beta"'] & & M_2 \arrow[dl, dotted, two heads, "\beta"] \\
& M_3 &
\end{tikzcd}$$

where the $\twoheadrightarrow_\beta$ arrow denotes zero or more β-rewrite steps. Thus if we contract different redexes in M we may always find a common reduct. An important consequence of the Church-Rosser property is that if a λ-term has a normal form then it is unique.

As regards the third issue, a finer analysis of the properties of β-rewrite sequences or β-derivations (analysis which we shall boldly call implementation techniques), also much work has been done. Graph sharing mechanisms and Abstract Machines are common to this area. A recent approach is to promote the substitution operation from the *metalevel* (the language of discourse) to the *object-level* (the language of study) by introducing it as a new term. This entails the addition of new rewrite rules, which make up the *substitution calculus*, to describe its behaviour. Also, the β-rewrite rule must be modified by replacing metalevel substitution with the new explicit substitution operator which implements it. As a result of this process we obtain a *calculus of explicit substitutions for the λ-calculus*. Before looking at an example, let us take a closer look at metalevel substitution (the operation used on the *RHS* of the β-rewrite rule) on which calculi of explicit substitution are based. As already mentioned, $M\{x \leftarrow N\}$ stands for the λ-term resulting from replacing the occurrences of the variable x (the formal parameter) in M by N (the actual parameter). This notion is defined by considering all the possible forms M may take, namely a variable, an abstraction (a term of the form $\lambda y.P$ for some λ-term P and variable y) or an application of a λ-term to another λ-term:

$$\begin{array}{lcll}
(PQ)\{x \leftarrow N\} & \overset{\text{def}}{=} & P\{x \leftarrow N\}Q\{x \leftarrow N\} & \\
(\lambda y.P)\{x \leftarrow N\} & \overset{\text{def}}{=} & \lambda y.(P\{x \leftarrow N\}) & x \neq y \\
x\{x \leftarrow N\} & \overset{\text{def}}{=} & N & \\
y\{x \leftarrow N\} & \overset{\text{def}}{=} & y & x \neq y
\end{array}$$

The first clause may be read as follows: the result of substituting the occurrences of x by N in PQ is that of substituting the occurrences of x by N in P on one hand, and applying the resulting term to the one obtained by substituting the occurrences of x by N in Q, on the other. We may assume that there is no term of the form $\lambda x.P$ in M, this stems from the possibility of changing the names of so called bound variables, a detailed account of which may be found in Chapter 2. The second clause may be explained similarly, and the final two clauses speak for themselves. So the $\{x \leftarrow N\}$ may be seen to traverse M until it reaches a variable, in which case this variable is either replaced by a copy of N, or is left unaltered and at the same time a copy of N is discarded.

By orienting the clauses of this definition from left to right and replacing the curly brackets by square ones, thus promoting the substitution operator into new operator in the object-language as discussed above, we obtain

[2] It may be proven that indeed $\Delta\Delta$ is the smallest λ-term, in the sense of the number of variables, applications and λ-symbols, admitting an infinite β-rewrite derivation [RSSX99, Sørensen's Omega Theorem]. See also [Ler76]

[3] Such as the simply typable terms, or the polymorphically typable terms [GLT89].

[4] λ_I-calculus to be precise, see introduction to Chapter 3.

the λx-calculus [Ros92, Blo97]:

$$\begin{array}{llll}
(\lambda x.M)N & \rightarrow_{Beta} & M\langle x := N\rangle & \\
(PQ)\langle x := N\rangle & \rightarrow_{App} & P\langle x := N\rangle Q\langle x := N\rangle & \\
(\lambda y.P)\langle x := N\rangle & \rightarrow_{Lam} & \lambda y.(P\langle x := N\rangle) & x \neq y \\
x\langle x := N\rangle & \rightarrow_{Var} & N & \\
y\langle x := N\rangle & \rightarrow_{Varf} & y & x \neq y
\end{array}$$

So now a term is either a variable, an application of a term to another term (represented by juxtaposition, as before), an abstraction, or a term of the form $P\langle x := Q\rangle$ called a *closure*. Thus the *RHS* of the *Beta*-rewrite rule is a new term in the calculus, and may be seen as a pending substitution yet to be executed. The substitution calculus of λx is obtained from λx by disregarding the *Beta*-rewrite rule, and is abbreviated x. Each β-rewrite step may be simulated in the λx-calculus by means of a number of λx-rewrite steps, namely a *Beta*-rewrite step followed by a number of x-rewrite steps. Consider, for example, the first β-rewrite step of the above mentioned infinite β-derivation. It may be simulated in λx as follows:

$$(\lambda x.xx)\Delta \rightarrow_{Beta} (xx)\langle x := \Delta\rangle \rightarrow_{App} x\langle x := \Delta\rangle x\langle x := \Delta\rangle \rightarrow_{Var} \Delta x\langle x := \Delta\rangle \rightarrow_{Var} \Delta\Delta$$

A benefit of calculi of explicit substitutions, among others, is that substitution may now be computed in a controlled manner. For example, some pending substitutions may not need to be executed, as illustrated below:

$$\begin{array}{lll}
(\lambda y.(\lambda x.z)(yy))N & \rightarrow_{Beta} & ((\lambda x.z)(yy))\langle y := N\rangle \\
& \rightarrow_{App} & (\lambda x.z)\langle y := N\rangle(yy)\langle y := N\rangle \\
& \rightarrow_{Lam} & (\lambda x.z\langle y := N\rangle)(yy)\langle y := N\rangle \\
& \rightarrow_{Varf} & (\lambda x.z)(yy)\langle y := N\rangle \\
& \rightarrow_{Beta} & z\langle x := (yy)\langle y := N\rangle\rangle \\
& \rightarrow_{Varf} & z
\end{array}$$

Note that there has been no need to compute the substitution $(yy)\langle y := N\rangle$ thus reducing computation time and unnecessary duplication of the term N. Calculi of explicit substitutions shall constitute the main theme of this thesis.

When confronted with a calculus of explicit substitutions for the λ-calculus, it is only natural to wonder whether its fundamental dynamical properties (those of the λ-calculus) are retained. We list three examples:

- Simulation: If a λ-term M β-rewrites to N then M should also rewrite to N in the calculus of explicit substitutions. This is coherent with our view of calculi of explicit substitutions as a fine-grained analysis of the β-rewrite step, and hence of the process of substitution.

- Church-Rosser: The λ-calculus enjoys the Church-Rosser property. Its explicit substitution calculus should do so too. If one interprets the 'meaning' of a term as its normal form (hence in our simplified[5] form setting terms without normal form have no 'meaning'), then failure of Church-Rosser could render some terms 'ambiguous' (terms with more than one normal form). This is certainly undesirable since as already mentioned, there are no 'ambiguous' λ-terms in the λ-calculus.

- Preservation of Strong Normalization (PSN): If a λ-term admits no infinite β-rewrite derivation then when computing the same term in the calculus of explicit substitutions no infinite rewrite derivation should arise either. The process of augmenting the λ-calculus with explicit substitutions may be seen as a process of enrichment of derivations. Any pair of λ-terms M and N such that $M \twoheadrightarrow_\beta N$ is benefited with a rich supply of alternative derivations. PSN guarantees that we enrich with caution.

Further properties arise such as strong normalization of the substitution calculus (which in our example would be x): All the rewrite derivations of the substitution calculus should terminate.

1.1 Hot Spots

Although the origins of explicit substitution dates back to the work of N.G.de Bruijn [Bru78] and also P-L.Curien [Cur86, Cur93], only in the last decade has it received full attention ([KR00] provides a survey). A wide range of research subjects have arised of which we shall mention just a few:

[5] Indeed, it is simplified since in λ-calculus it is terms without a so called head-normal form that may be interpreted as having no 'meaning' [Bar84, Th.16.1.3].

- Discovering a λ-calculus with explicit substitutions simultaneously satisfying confluence on open terms (CR), simulation of one-step β-rewrites (Simulation), and preservation of strong normalization has been an important source of research. Although Simulation and CR are simpler to obtain, P-A.Melliès has shown that preservation of strong normalization may not hold [Mel95]. A first step was taken by C.Muñoz [Muñ96]: the $\lambda\zeta$-calculus satisfies CR and PSN but not Simulation (however, it simulates innnermost β-rewrite steps). A further attempt was the λse-calculus by F.Kamareddine and A.Ríos [KR97]: λse enjoys CR and Simulation, and for some time it was not known if PSN held, however only recently B.Guillaume came up with a proof of failure of PSN [Gui99]. Based on the latter proof, B.Guillaume and R.David proposed the $\lambda\mathtt{ws}$-calculus [DG99], a calculus which satisfies PSN and CR. It, moreover, satisfies Simulation provided λ-terms are decorated with certain labels, which may be seen as representing application of the weakening rule when considering a typing discipline. Thus, although compliance with Simulation is somewhat questionable, $\lambda\mathtt{ws}$ appears to be the furthest one may get today in the direction of a calculus satisfying all three of the above mentioned properties. Further work in this direction is the calculus of H. Goguen and J.Goubault-Larrecq [GGL00] based on extended K, S and I combinators.

- Various techniques for proving preservation of strong normalization in calculi of explicit substitutions have appeared. The first of these proofs of PSN seems to have been given independently by R.Bloo [Blo95] and P.Lescanne [BBLRD96] (see [Blo97] for historical remarks). Later, R.Bloo and H.Geuvers provided a new technique for proving PSN based on recursive path orders [BH98]. In this thesis we shall have a chance of taking a closer look at this technique, together with the one by P.Lescanne. E.Ritter introduced a technique for proving PSN based on Girard's 'candidats de reductibilité' [Rit99]. In order to prove PSN for the $\lambda\mathtt{ws}$-calculus B.Guillaume and R.David [DG99, DG01] have seen themselves in the need of introducing yet a further proof technique based on constricting strategies (see Chapter 4 of this thesis for further details), since previous techniques seem not to be applicable. Perhaps the most recent is that of V.van Oostrom et al [KOO01b] for λx based on standardization. This plethora of methods suggests that no sufficiently general technique for proving PSN has yet been found.

- Reducing higher-order formalisms/problems to a first-order setting has been a further area of active research in explicit substitutions. Perhaps a word on our intended meaning of 'reducing' is in order: three ingredients are required, some higher-order formalism/problem, some first-order formalism/problem, and an encoding of the former into the latter. The prime example is that of the lambda calculus. Calculi of explicit substitutions such as $\lambda\sigma$ [ACCL91], $\lambda\upsilon$ [BBLRD96] and λs [KR95] are first-order formulations of the lambda calculus. Indeed, by introducing de Bruijn indices notation [Bru72, Bru78] and explicit substitutions a first-order term rewrite system (no binding operators, substitution - as defined above -, nor α-conversion present) is obtained in which each β-rewrite step may be encoded. G.Dowek, T.Hardin, and C.Kirchner [DHK95] reduce higher-order unification to first-order unification modulo the calculus of explicit substitutions $\lambda\sigma$ [ACCL91], they also consider the case of higher-order pattern unification in [DHKP98]. M.Ayala-Rincón and F.Kamareddine do the same using the λse-calculus of explicit substitutions [ARK00]. Other examples are that of reducing higher-order logic to first-order modulo [DHK01] and higher-order theorem proving to first-order modulo [DHK98]. In this thesis we shall address the issue of reducing higher-order rewriting to first-order rewriting modulo.

- Extending the notion of explicit substitution calculi beyond the λ-calculus has also deserved much attention. R.Bloo and K.Rose define Explicit Combinatory Reduction Systems [BR96] by augmenting J.W.Klop's Combinatory Reduction Systems (*CRS*) [Klo80] with explicit substitutions. Explicit Combinatory Reduction Systems are not first-order rewrite systems since they require dealing with α-conversion. B.Pagano defines Explicit Reduction Systems (*XRS*) [Pag98] a first-order formalism of higher-order rewriting, based on an extension of the $\lambda\sigma_{\Uparrow}$-calculus [HL89], which caters for arbitrary binders and function symbols (and not just lambda abstraction and application as in the lambda calculus). Since no relation with existing higher-order rewrite formalisms such as *HRS* [Nip91], *CRS* [Klo80], *ERS* [Kha90], etc. is established, in the light of our above mentioned interpretation of 'reducing', we are inclined not to consider this work as reducing higher-order rewriting to first-order rewriting. More recently, M-O.Stehr introduced the Calculus of Indexed Names and Named Indices (*CINNI*) [Ste00], based on so called Berkling's notation - a convenient amalgamation of de Bruijn indices and names notation (see [Ste00] for references). It is a first-order calculus which allows the encoding of binders and substitution. However, as in the case of *XRS* no relation is established with existing higher-order rewrite systems. We shall study an encoding of Z.Khasidashvili's *ERS* [Kha90] in a first-order setting with the aid of explicit substitutions.

1.2 Overview of the Thesis

This thesis is divided into three parts which conform its main body. The latter is preceded by a brief introduction to the theories of rewriting and lambda calculus followed by an overview of some basic calculi of explicit substitutions and their outstanding properties.

Part I In Part I we study perpetual rewrite strategies in two calculi of explicit substitutions, namely, $\lambda\mathrm{x}$ and $\lambda\mathbf{ws}$. A perpetual strategy is one that preserves the possibility of infinite derivations. Current literature on perpetual strategies in term rewriting require that the property of orthogonality be fulfilled, however calculi of explicit substitution are not orthogonal[6]. We exploit the techniques developed for proving preservation of strong normalization in order to obtain perpetual strategies for calculi of explicit substitutions. Moreover, we also obtain a characterization of the strongly normalizing terms (those admitting no infinite derivations). The latter is particularly welcome in the setting of $\lambda\mathbf{ws}$ due to the presence of substitution composition which complicates matters. Perhaps a word or two on substitution composition may portray the difficulties encountered when present in a calculus of explicit substitutions.

Consider the $\lambda\mathrm{x}$-calculus. In a term of the form $P\langle x := Q\rangle$ let us call Q the body of the substitution. The point is that in the $\lambda\mathrm{x}$-calculus the bodies of substitutions are sealed units: although $\lambda\mathrm{x}$-rewrite steps may take place inside them, they do not interact in any way with other subterms of the term in which they occur - in the sense that there is no rewrite rule that 'combines' a body of a substitution with some other term. Indeed, by inspecting the rewrite rules of the $\lambda\mathrm{x}$-calculus one may observe that the bodies of all substitutions on the *RHS*s occur identically on their respective *LHS*. When a rule allowing the composition of substitution is introduced, such as:

$$M\langle x := N\rangle\langle y := O\rangle \quad \rightarrow_c \quad M\langle x := N\langle y := O\rangle\rangle \quad \text{if } y \text{ does not occur free in } M$$

this no longer holds. In the c-rewrite rule, the substitution body $N\langle y := O\rangle$ does not occur on the *LHS*. As a consequence, bodies of substitutions which behaved well as sealed units in the sense that they were not sources of infinite derivations may no longer do so. For example, the term $z\langle x := yy\rangle\langle y := \Delta\rangle$ has two substitution bodies, namely, yy and Δ. Both are terminating terms. However, if we apply the c-rewrite rule we obtain $z\langle x := (yy)\langle y := \Delta\rangle\rangle$ where a new substitution body $(yy)\langle y := \Delta\rangle$, source of an infinite $\lambda\mathrm{x}$-derivation, has appeared. To sum up, devising perpetual rewrite strategies and characterizing the strongly normalizing terms in $\lambda\mathbf{ws}$ shall require more work than for $\lambda\mathrm{x}$, this is developed in Chapter 4.

Chapter 3 deals with the $\lambda\mathrm{x}$-calculus for which we also show how we may take advantage of our studies on perpetuality in proving strong normalization of a polymorphic lambda calculus with explicit substitutions.

Polymorphism is a typing discipline that allows functions to be applied to arguments of possibly different types. This promotes code reuse and, in the presence of higher-order functions, is one of the essential ingredients in any modern functional programming language. The classical example of a polymorphic function is the identity function $I = \lambda x.x$. Note that given some term, this function returns a copy of it, regardless of the nature of the term in question. Thus, for *any* type τ we say that I has type $\tau \rightarrow \tau$. This may be internalized in the language of types by assigning the type $\forall\tau.\tau \rightarrow \tau$ to I, where τ plays the role of a type variable, ranging over all elements in the universe of types. System F [Gir72, GLT89] is a set of typing rules for typing polymorphic terms, together with two rewrite rules: the β-rewrite rule, and a rewrite rule for instantiating type variables by arbitrary types. We shall augment the F-calculus with explicit substitutions obtaining F_{es} and, in addition to strong normalization, also study subject reduction for the extended typing rules. Since in F we may abstract not only term variables but also type variables, two distinct notions of substitution shall be introduced and studied: term and type substitutions. Subject Reduction and Strong Normalization are then considered. The proof of the latter property is obtained by applying J-Y.Girard's 'candidats de reductibilité' proof technique [Gir72]. The work reported in Chapter 3 of this part of the thesis has been published as [Bon99a, Bon01]. Chapter 4 is joint work with A.Arbiser and A.Ríos.

Part II Part II introduces the ς-calculus of M.Abadi and L.Cardelli [AC96]. This calculus is at the level of the lambda calculus but is based on objects instead of functions. Objects are composed of methods. The basic operations on objects are method invocation and method override. Fields may be represented as methods which do not use their self parameter (Section 5.3). An encoding of the lambda calculus in ς,

[6] This should be considered a virtue, not a defect!

the *function-object* translation, is provided in [AC96]: each β-rewrite step may be encoded as a number of ς-rewrite steps. This encoding makes use of fields and metalevel substitution. We shall augment ς with explicit substitutions à la λv [BBLRD96], previously shifting to a de Bruijn indices notation, obtaining the ς_{dbes}-calculus. An attempt to encode λv in ς_{dbes} by extending in a natural way the function-object translation shall reveal two obstacles:

1. encoding application: explicit substitutions interfere with the encoding of fields as methods which do not use their self parameter.
2. encoding abstraction: the use of metalevel substitution in the function-object translation requires a new notion of explicit substitution in order to be encoded soundly in this setting.

The first issue is taken care of by introducing fields as primitive constructs, the second by introducing the notion of invoke substitution. Simulation of λv is then seen to hold. Finally, we consider confluence and preservation of strong normalization. A weak form of composition between explicit substitution and invoke substitution (which is also explicit) shall require the latter issue to be considered with care. The work reported in this part has been published as [Bon99b].

Part III Part III is concerned with reducing higher-order rewriting to first-order rewriting modulo. This part is subdivided into three chapters. An important requirement was fixed at the offset: a well-established higher-order rewrite formalism was to be used as the departing formalism. Chapter 6 introduces (a simplification of) Z.Khasidashvili's Expression Reduction Systems (*ERS*) [Kha90], and introduces a de Bruijn notation for it in order to get rid of α-conversion. The result is the $SERS_{db}$ rewrite formalism. Chapter 7 introduces the first-order rewrite formalism with explicit substitutions *ExERS*. We then present an encoding, called the Conversion Procedure, translating any higher-order rewrite system into first-order rewriting modulo an equational theory $\mathcal{E}$ (in the *ExERS*-formalism). The latter equational theory is that of the substitution calculus. In other words, a rewrite step $M \rightarrow_{\mathcal{R}} N$ in a higher-order $SERS_{db}$ $\mathcal{R}$ may be encoded as a rewrite step

$$M =_{\mathcal{E}} M' \rightarrow_{fo(\mathcal{R})} N' =_{\mathcal{E}} N$$

where $M' \rightarrow_{fo(\mathcal{R})} N'$ is a rewrite step in the first-order version of $\mathcal{R}$ (the first-order rewrite system resulting from applying the Conversion Procedure to $\mathcal{R}$) and $M =_{\mathcal{E}} M'$ implements higher-order pattern matching. Assuming the substitution calculus $\mathcal{S}_{\mathcal{E}}$ from which $\mathcal{E}$ originates is a confluent first-order term rewrite system, and that M, N are terms without closures, it may be proved that in fact we have:

$$M \twoheadleftarrow_{\mathcal{S}_{\mathcal{E}}} M' \rightarrow_{fo(\mathcal{R})} N' \twoheadrightarrow_{\mathcal{S}_{\mathcal{E}}} N$$

That is to say, a higher-order rewrite step is encoded as a series of $\mathcal{S}_{\mathcal{E}}$ expansions, followed by a first-order rewrite step in $fo(\mathcal{R})$, and finally a series of rewrite steps in the substitution calculus[7]. Finally, we provide a simple syntactical criterion to determine a subclass of the $SERS_{db}$ systems which may be encoded as full first-order systems in the sense that $M = M'$, where '=' denotes syntactical equality. It is fair to say that, for these systems, higher-order pattern matching may be directly encoded as syntactic first-order matching. In other words, in order to determine if a rewrite rule is applicable to some term syntactic matching suffices. This class includes many systems such as, for example, the λ-calculus. Chapter 8 argues that for this subclass of systems techniques developed for first-order rewriting may be lifted or transferred to higher-order rewriting. It does so by transferring the Standardization Theorem [CF58, HL91, Klo80, Bou85, Mel96]. This is achieved by generalizing a result due to P-A.Melliès [Mel00]. The material reported in chapters 6 and 7 of this part is joint work with D.Kesner and A.Ríos and has been published as [BKR00, BKR01].

Finally, we conclude and discuss further research directions. Also, the conclusions pertaining to each chapter have been grouped together in this final chapter.

There is no interdependence between parts I, II and III, they may be read in any order. Due to the rather technical nature of this thesis the reader is advised not to cover all proofs on a first reading, notably in chapters 6 and 7. In the hope of contributing to readability I have moved some of the more routine proofs to an appendix.

This work has been typeset using LaTeX and Xy-pic.

[7] A similar decomposition of higher-order rewrite steps is studied extensively by V. van Oostrom and F. van Raamsdonk [OR94, Oos94] in order to define a general formalism for higher-order rewriting encompassing many known higher-order rewrite systems in the literature. See Chapter 7 for further details.

Chapter 2

Rewriting, Lambda Calculus and Explicit Substitutions

This chapter presents the theory and basic results of rewriting, lambda calculus and calculi of explicit substitutions relevant to this thesis. We shall first give a brief overview of *abstract* rewriting and shall also consider *term* rewriting. We then present the lambda calculus with variable names followed by the lambda calculus where the variable names are replaced by certain numbers, called de Bruijn indices [Bru72, Bru78]. Finally, we provide a brief overview of some calculi of explicit substitutions.

The primary aim of this chapter is to fix notation and by no means pretends to be a tutorial on the subject. As we go along we shall provide the reader with pointers to appropriate literature.

2.1 Rewriting

Rewriting is a model of computation in that a class of objects together with a class of transformation rules specifying how these objects may be transformed into other objects, is provided. Depending on the choice of objects we may have different flavours of rewriting. If the objects are terms (i.e. elements of the algebra of terms as defined in universal algebra) then we speak of term rewriting, if the objects are graphs then we speak of graph rewriting, and so on.

For a survey on rewriting the reader may wish to consult [Hue80, DJ90, Klo92]. Recently a text book on rewriting has appeared [BN98].

Definition 2.1 An *Abstract Rewrite System* (*ARS*) $\mathcal{R}$ is a pair (A, R) where A is a set of objects and R is a binary relation on A (i.e. $R \subseteq A \times A$). We call R the *rewrite relation* or the *reduction relation* of $\mathcal{R}$. If $a, b \in A$ and $(a, b) \in R$ then we write aRb or $a \rightarrow_{\mathcal{R}} b$ and say that *a $\mathcal{R}$-rewrites* or *$\mathcal{R}$-reduces* to b. If $\mathcal{R}$ is clear from the context we just say that a *rewrites* or *reduces* to b.

Note that Abstract Rewrite Systems are indeed abstract since no further requirements than those of Definition 2.1 are demanded.

We use the '=' symbol to denote equality of objects in A. Also, we write $\twoheadrightarrow_{\mathcal{R}}$ for the smallest reflexive and transitive relation containing $\rightarrow_{\mathcal{R}}$. Furthermore, $=_{\mathcal{R}}$ stands for the smallest reflexive, symmetric and transitive relation containing $\rightarrow_{\mathcal{R}}$. A sequence of the form:

$$a_0 \rightarrow_{\mathcal{R}} a_1 \rightarrow_{\mathcal{R}} a_2 \rightarrow_{\mathcal{R}} \dots \rightarrow_{\mathcal{R}} a_{n-1} \rightarrow_{\mathcal{R}} a_n$$

is called a (finite) *$\mathcal{R}$-derivation from a_0 to a_n*. Let $a_0, a_1, \dots, a_n, \dots$ be elements of A. A sequence of the form

$$a_0 \rightarrow_{\mathcal{R}} a_1 \rightarrow_{\mathcal{R}} a_2 \rightarrow_{\mathcal{R}} \dots \rightarrow_{\mathcal{R}} a_n \rightarrow_{\mathcal{R}} \dots$$

is an infinite $\mathcal{R}$-derivation from a_0.

Definition 2.2 Let $\mathcal{R} = (A, R)$ be an *ARS*.

- We say $\mathcal{R}$ satisfies the *diamond property* if for every $a, b, c \in A$ such that $a \rightarrow_{\mathcal{R}} b$ and $a \rightarrow_{\mathcal{R}} c$ there exists $d \in A$ such that $b \rightarrow_{\mathcal{R}} d$ and $c \rightarrow_{\mathcal{R}} d$.

- We say $\mathcal{R}$ is *locally confluent* if for every $a, b, c \in A$ such that $a \to_{\mathcal{R}} b$ and $a \to_{\mathcal{R}} c$ there exists $d \in A$ such that $b \twoheadrightarrow_{\mathcal{R}} d$ and $c \twoheadrightarrow_{\mathcal{R}} d$.
- We say $\mathcal{R}$ is *confluent* if for every $a, b, c \in A$ such that $a \twoheadrightarrow_{\mathcal{R}} b$ and $a \twoheadrightarrow_{\mathcal{R}} c$ there exists $d \in A$ such that $b \twoheadrightarrow_{\mathcal{R}} d$ and $c \twoheadrightarrow_{\mathcal{R}} d$.
- We say $\mathcal{R}$ is *Church-Rosser* if for every $a, b \in A$ such that $a =_{\mathcal{R}} b$ there exists $d \in A$ such that $a \twoheadrightarrow_{\mathcal{R}} d$ and $b \twoheadrightarrow_{\mathcal{R}} d$.

The items comprising Definition 2.2 are depicted in Figure 2.1.

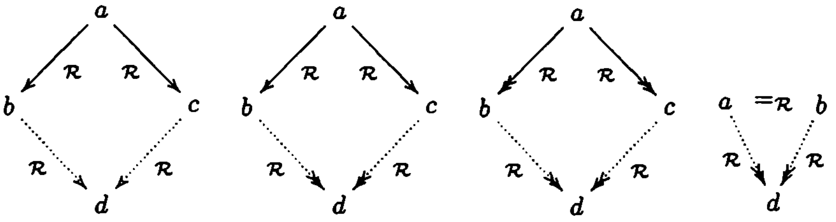


Figure 2.1: Properties of *ARS*

It may be shown that $\mathcal{R}$ is confluent if and only if $\mathcal{R}$ is Church-Rosser. Note that if $\mathcal{R}$ is confluent then it is locally confluent; the converse is not true, as the following example illustrates:

$$\begin{tikzcd} a & b \arrow[l] \arrow[r, bend left] & c \arrow[l, bend left] \arrow[r] & d \end{tikzcd}$$

However if $\mathcal{R}$ is *terminating* then the converse does hold.

Definition 2.3 Let $\mathcal{R} = (A, R)$ be an *ARS*.

- An element $a \in A$ is said to be in *$\mathcal{R}$-normal form* if there is no $b \in A$ such that $a \to_{\mathcal{R}} b$.
- If $a \twoheadrightarrow_{\mathcal{R}} b$ and b is a $\mathcal{R}$-normal form then b is said to be a *$\mathcal{R}$-normal form of a*.
- An element $a \in A$ is *$\mathcal{R}$-normalizing* if there exists a $\mathcal{R}$-normal form of a. If all elements of A are $\mathcal{R}$-normalizing then we say $\mathcal{R}$ is *weakly normalizing*.
- An element $a \in A$ is *strongly $\mathcal{R}$-normalizing* if every $\mathcal{R}$-derivation starting from a is finite (thus ending in a $\mathcal{R}$-normal form). If all elements of A are strongly $\mathcal{R}$-normalizing then we say $\mathcal{R}$ is *strongly normalizing*. We use $SN_{\mathcal{R}}$ for the set of elements in A that are strongly $\mathcal{R}$-normalizing.
- An element $a \in A$ is *$\mathcal{R}$-finitely branching* if the set $\{b \mid a \to_{\mathcal{R}} b\}$ is finite. $\mathcal{R}$ is *finitely branching* if every $a \in A$ is $\mathcal{R}$-finitely branching.

When $\mathcal{R}$ is clear from the context we often omit the prefix '$\mathcal{R}$-' in the above defined notions.

Note that if a term is not strongly $\mathcal{R}$-normalizing then it admits at least one infinite $\mathcal{R}$-derivation. If an element $a \in A$ admits an infinite $\mathcal{R}$-derivation then we write $\infty_{\mathcal{R}}(a)$.

Another word used to designate strongly normalizing *ARS* is 'terminating'. Confluence and termination are two of the most important properties studied in *ARS*.

Lemma 2.4 (Newman's Lemma) Let $\mathcal{R}$ be an *ARS*. If $\mathcal{R}$ is terminating and weakly confluent then it is confluent.

See [Hue80] for a proof.

Below IN stands for the natural numbers including the number zero.

Definition 2.5 Let $\mathcal{R} = (A, R)$ be an *ARS*. We define the function $\mathit{maxred}_{\mathcal{R}}(\bullet) : A \to \mathbb{N} \cup \{\infty\}$ as

$$\mathit{maxred}_{\mathcal{R}}(a) \overset{\text{def}}{=} \begin{cases} n & \text{if there is a } \mathcal{R}\text{-derivation } a \to_{\mathcal{R}} a_1 \to_{\mathcal{R}} a_2 \ldots \to_{\mathcal{R}} a_n \\ & \text{such that for any } \mathcal{R}\text{-derivation } a \to_{\mathcal{R}} a'_1 \to_{\mathcal{R}} a'_2 \ldots \to_{\mathcal{R}} a'_m \text{ we have } m \leq n \\ \infty & \text{otherwise} \end{cases}$$

Thus if $a \in A$ is strongly $\mathcal{R}$-normalizing and $\mathcal{R}$-finitely branching , $\mathit{maxred}_{\mathcal{R}}(a)$ returns the length of the longest $\mathcal{R}$-derivation from a otherwise it returns the special symbol ∞.

The proof of the following result is left to the reader.

Lemma 2.6 Let $\mathcal{R} = (A, R_1 \cup R_2)$ and $\mathcal{S} = (B, >)$ be *ARS*'s such that $\mathcal{S}$ is strongly normalizing. If

1. (A, R_2) is strongly normalizing, and
2. there exists a function $f : A \to B$ such that $a \to_{R_1} b$ implies $f(a) > f(b)$ and $a \to_{R_2} b$ implies $f(a) \geq f(b)$[1],

then $\mathcal{R}$ is strongly normalizing.

We shall now consider term rewriting. More precisely, we shall present first-order term rewriting: the set of objects are first-order terms.

Definition 2.7 (First-Order Terms) A signature Σ is a pair $(\Sigma_f, \mathcal{V})$ where Σ_f is a set of function symbols, each of which is equipped with a natural number called its arity, and $\mathcal{V}$ is a denumerably infinite set of objects called *variables*. The set of *first-order terms generated by* Σ is the smallest set $\mathcal{T}_\Sigma$ such that:

1. for all $x \in \mathcal{V}$, $x \in \mathcal{T}_\Sigma$.
2. for every function symbol $g \in \Sigma_f$ of arity n, for every $a_1, \ldots, a_n \in \mathcal{T}_\Sigma$ we have $g(a_1, \ldots, a_n) \in \mathcal{T}_\Sigma$. If $n = 0$ then we just write g.

The set $\mathcal{T}_\Sigma$ is known as the term algebra over signature Σ. A term without variables is called a *ground* term.

Definition 2.8 (Subterm) Let $a, b \in \mathcal{T}_\Sigma$. We say that a is a *subterm* of b iff $a \subseteq b$ holds. The latter is defined as the smallest reflexive relation such that: if $c \subseteq d$ then $c \subseteq g(a_1, \ldots, a_{i-1}, d, a_{i+1}, \ldots, a_n)$ for all $g \in \Sigma_f$ of arity n and for all $a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n \in \mathcal{T}_\Sigma$. If a is a subterm of b and $a \neq b$ then we say that a is a *strict subterm* of b, written $a \subset b$.

Given some term $a \in \mathcal{T}_\Sigma$ we may replace the variables in a by other terms. This operation is known as (first-order) *substitution*, and is specified in terms of assignments.

Definition 2.9 (Assignment) Let $\Sigma = (\Sigma_f, \mathcal{V})$ be a signature. An *assignment* over Σ is a function $\rho : \mathcal{V} \to \mathcal{T}_\Sigma$ such that $\rho(x) \neq x$ for only finitely many variables. An assignment can be extended homomorphically to a mapping $\overline{\rho} : \mathcal{T}_\Sigma \to \mathcal{T}_\Sigma$ as follows:

$$\begin{array}{lcl} \overline{\rho}(x) & \overset{\text{def}}{=} & \rho(x) \\ \overline{\rho}(g(a_1, \ldots, a_n)) & \overset{\text{def}}{=} & g(\overline{\rho}(a_1), \ldots, \overline{\rho}(a_n)) \end{array}$$

where $g \in \Sigma_f$ of arity n. This extension is referred to as a substitution and is abbreviated ρ (i.e. without the overlining).

Thus a substitution replaces *simultaneously* all occurrences of variables by their respective ρ-images.

Definition 2.10 (Unifiable Terms) Let Σ be a signature. Two terms $a, b \in \mathcal{T}_\Sigma$ are said to be *unifiable* if there exists a substitution ρ over Σ such that $\rho(a) = \rho(b)$. This substitution is known as a *unifier* of a and b.

The notion of unifiable terms shall be required when defining orthogonal first-order term rewrite systems (Definition 2.12).

[1] The relation $\geq$ is defined in $\mathcal{S}$ as expected: $a \geq b$ iff $a > b$ or $a = b$.

Definition 2.11 (First-Order Term Rewrite System) A *first-order term rewrite system (TRS) over* Σ is a pair $\mathcal{R} = (\Sigma, R)$ where $\Sigma = (\Sigma_f, \mathcal{V})$ is a signature and R is a set of pairs of terms (l, r) in $\mathcal{T}_\Sigma$ called *rewrite rules* (l is the *LHS*[2] and r the *RHS* of the rule) such that: l is not a variable, and the set of variables in r are included in that of l.

The rewrite relation induced by R is written $a \rightarrow_{\mathcal{R}} b$ and defined as:

$$\frac{\rho \text{ substitution, } (l,r) \in R}{\rho(l) \rightarrow_{\mathcal{R}} \rho(r)} \qquad \frac{a \rightarrow_{\mathcal{R}} b, \quad g \in \Sigma_f \text{ of arity } n \geq 1}{g(a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_n) \rightarrow_{\mathcal{R}} g(a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_n)}$$

Note that if the *LHS* of a rewrite rule were allowed to be a variable then the resulting *TRS* would be trivially non-terminating. The same happens if we permit occurrences of variables on the *RHS* which do not occur on the left-hand side. In the latter case we also risk rendering our *TRS* trivially non-confluent.

Definition 2.12 (Orthogonal Term Rewrite Systems) Let $\mathcal{R} = (\Sigma, R)$ be a *TRS*.

- A term a is *linear* if all variables occur at most once in a. $\mathcal{R}$ is said to be *left-linear* if the *LHS* of each rule in R is linear.
- Let (l, r) and (g, d) be rewrite rules in R. If there exists a non-variable subterm l' of l such that l' and g are unifiable, then (l, r) and (g, d) are said to *overlap*. Since by this definition every rewrite overlaps with itself we shall rule out this case, in other words, if $l' = l$, and l' and g are unifiable then we shall demand that (l, r) and (g, d) are different rewrite rules. $\mathcal{R}$ is said to be *non-overlapping* or *non-ambiguous* if R does not contain a pair of overlapping rewrite rules.
- If $\mathcal{R}$ is left-linear and non-overlapping then we say it is *orthogonal*.

Orthogonal *TRS* enjoy good properties. In particular, all orthogonal *TRS* are confluent [Hue80].

2.2 The Lambda Calculus

The theory of lambda calculus was introduced by A.Church in the 1930s [Chu32] as part of a more general theory related to his studies on the foundations of mathematics. By orienting the equations of the theory we obtain a confluent rewrite system, a result which was first proved for the equational theory in order to establish its consistency. The lambda calculus deals with functions and function application, and at the same time achieves a high level of abstraction by using an intuitive set of constructors to represent them and just one rule, namely the application of a function to an argument. Terms represent functions and the functions are put to work by applying them to arguments. The strength of the calculus lies in that an argument can be another function. In fact, a function may by applied to (a copy of) itself.

We shall present a brief introduction to the lambda calculus. First we introduce the usual presentation with variables, then we shall consider a presentation in which variables are replaced with numbers called indices. For further details the reader is referred to the standard reference [Bar84]. See also [Kri90].

2.2.1 The Lambda Calculus with Names

Definition 2.13 Let $\mathcal{V}$ be a denumerably infinite set of variables.

- The set of terms of the lambda calculus are called lambda terms or λ-terms and denoted $\mathcal{T}_\lambda$. They are defined as the smallest set such that the following three conditions hold:
  - if $x \in \mathcal{V}$ then $x \in \mathcal{T}_\lambda$,
  - if $M_1, M_2 \in \mathcal{T}_\lambda$ then $(M_1 M_2) \in \mathcal{T}_\lambda$, and
  - if $M \in \mathcal{T}_\lambda$ and $x \in \mathcal{V}$ then $(\lambda x.M) \in \mathcal{T}_\lambda$.

  We shall often abbreviate definitions of terms by using BNF-style notation. In this case we would write:

$$M ::= x \mid (MM) \mid (\lambda x.M)$$

[2] *LHS* stands for left-hand side and *RHS* for right-hand side.

where x ranges over the objects in $\mathcal{V}$.

In order not to clutter the notation of terms with many parenthesis some conventions are taken into account. Application associates to the left and binds stronger than abstraction. For example, $\lambda x.wxy$ stands for $(\lambda x.((wx)y))$. A term of the form M_1M_2 is called an *application*, a term of the form $\lambda x.M$ is called an *abstraction*. The symbol 'λ' is called a *binder* and x in 'λx' is called a *binding variable*.

- The set of *free variables* of a lambda term M is denoted $FV(M)$ and defined as:

$$\begin{array}{lll} FV(x) & \stackrel{\text{def}}{=} & \{x\} \\ FV(M_1M_2) & \stackrel{\text{def}}{=} & FV(M_1) \cup FV(M_2) \\ FV(\lambda x.M) & \stackrel{\text{def}}{=} & FV(M) \setminus \{x\} \end{array}$$

where '$\setminus$' is the subtraction operation on sets. A lambda term M is *closed* if $FV(M) = \emptyset$. The occurrences of (non-binding) variables in a term which are not free are said to be *bound*. Thus, if $M = \lambda x.wxy$ then $FV(M) = \{w, y\}$, and x is a bound variable. The term $M = \lambda x.y$ has no bound variables. Note that two different occurrences of the same variable need not be both bound nor free, as illustrated by the following example: $(\lambda x.x)x$.

We shall use letters $M, N, O, P, \ldots$ to denote arbitrary lambda terms. The subterm relation is defined as in the first-order case by considering the signature $\Sigma_\lambda = (\Sigma_f, \mathcal{V})$ where the set of function symbols Σ_f consists of the application symbol and an infinite number of 'binder' symbols λx, one for each $x \in \mathcal{V}$.

It is common to identify terms differing only in the names of the variables they bind. For example, $\lambda x.x$ and $\lambda y.y$ are considered as representing the same function, namely the identity function. Terms identified in this way (by renaming their bound variables) are called *α-convertible* or *α-equivalent*. We write $=_\alpha$ for the relation of α-conversion. Def. 2.14 below shall provide a formal definition of α-conversion.

Note that the subterm relation is not compatible with renaming in the following sense: if $P \subseteq Q$ and $P' =_\alpha P$, then not necessarily do we have $P' \subseteq Q$. Likewise, if $P \subseteq Q$ and $Q' =_\alpha Q$, then not necessarily do we have $P \subseteq Q'$. For an example of the first case consider $P = \lambda x.x$, $Q = \lambda y.\lambda x.x$, and $P' = \lambda z.z$. The second case may be illustrated in a similar way.

Renaming is important for defining *substitution*. Substitution is the notion corresponding to replacement as seen in *TRS*. However, in *TRS* any variable may be replaced by any term, whereas in the lambda calculus only the free variables may undergo such a transformation. This is a key difference. Indeed, let us consider the lambda terms as first-order terms by using the signature Σ_λ as defined above. Then the assignment $\rho(x) = y$ and $\rho(z) = z$ for all $z \neq x$ applied to the term $M = \lambda y.x$ yields $\rho(\lambda y.x) = \lambda y.y$. The problem with this result is that a variable which enjoyed the status of being free in M, namely x, has now been replaced by a variable which is bound (y). Had we first renamed the bound variable y in M to z yielding $\lambda z.x$ then applying ρ we would have obtained $\lambda z.y$, which is intuitively what we expect if abstractions are considered as representing functions.

Definition 2.14 (Substitution)
- The result of substituting a term N for all free occurrences of a variable x in a term M is written $M\{x \leftarrow N\}$ and defined inductively[3] as follows:

$$\begin{array}{lll} x\{x \leftarrow N\} & \stackrel{\text{def}}{=} & N \\ y\{x \leftarrow N\} & \stackrel{\text{def}}{=} & y \quad \text{if } x \neq y \\ (M_1M_2)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & M_1\{x \leftarrow N\}M_2\{x \leftarrow N\} \\ (\lambda y.M_1)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & \lambda z.M_1\{y \leftarrow z\}\{x \leftarrow N\} \text{ where } z \text{ does not occur at all in } \lambda y.M_1,\ x \text{ or } N \end{array}$$

- α-conversion of lambda terms, denoted $=_\alpha$, is the smallest equivalence relation such that:

$$\begin{array}{lll} \text{if } M\{x \leftarrow y\} =_\alpha N & \text{then} & \lambda x.M =_\alpha \lambda y.N \\ \text{if } M_1 =_\alpha N_1 \text{ and } M_2 =_\alpha N_2 & \text{then} & M_1M_2 =_\alpha N_1N_2 \end{array}$$

[3] In full precision, it is defined by induction on the length of M, that is to say the number of variables, abstractions and applications in M.

The important clause of the definition of substitution is the last one: when a substitution traversing a term reaches a lambda binder we rename its bound variable to a fresh one in order to avoid unwanted capture of variables. Note that any variable z satisfying the conditions of this clause may be selected. Thus one may think of $M\{x \leftarrow N\}$ as a class of terms rather than just one term. The terms in this class are α-equivalent. Therefore, substitution is defined on α-equivalence classes of terms. It may be shown that it is well-defined:

Lemma 2.15 If $M_1 =_\alpha M_2$ and $N_1 =_\alpha N_2$ then $M_1\{x \leftarrow N_1\} =_\alpha M_2\{x \leftarrow N_2\}$.

The following *variable convention* is adopted in order to ease the presentation of the calculus.

Remark 2.16 (Variable Convention) Names of bound variables are different from the names of free variables, and moreover, different occurrences of the lambda binder have different binding variables.

Thus we consider $(\lambda y.y)x$ instead of $(\lambda x.x)x$, and $(\lambda x.xx)(\lambda y.yy)$ instead of $(\lambda x.xx)(\lambda x.xx)$. We shall identify α-equivalent terms, thus '$=$' shall stand for '$=_\alpha$', unless otherwise stated.

Substitution satisfies the following property.

Lemma 2.17 (Substitution Lemma) $M\{x \leftarrow N\}\{y \leftarrow O\} = M\{y \leftarrow O\}\{x \leftarrow N\{y \leftarrow O\}\}$ if $x \notin FV(O)$.

The β-rewrite rule is stated as follows.

Definition 2.18 (β-rewrite rule) We say M *β-rewrites* to N iff $M \rightarrow_\beta N$, where the latter relation is defined by the following inference schemes:

$$\frac{}{(\lambda x.M)N \rightarrow_\beta M\{x \leftarrow N\}} \qquad \frac{M_1 \rightarrow_\beta N_1}{M_1M_2 \rightarrow_\beta N_1M_2} \qquad \frac{M_2 \rightarrow_\beta N_2}{M_1M_2 \rightarrow_\beta M_1N_2} \qquad \frac{M \rightarrow_\beta N}{\lambda x.M \rightarrow_\beta \lambda x.N}$$

The leftmost inference scheme of Def. 2.18 is called the *β-rewrite axiom*. It is the only inference scheme which makes use of substitution. The substitution $M\{x \leftarrow N\}$ takes place at the *metalevel*, thus it is external to the calculus. The β-rewrite rule is sometimes defined by exhibiting just the β-rewrite axiom and then demanding that the 'contextual closure' of this axiom be taken[4]. Contextual closure means we should add the remaining inference schemes of Def. 2.18 to the β-rewrite axiom. We too shall follow this practice in order to shorten the presentation, whenever possible. Nevertheless, in this chapter and for expository purposes we present the full inference schemes for some of the calculi. Let us see an example of a β-derivation in the lambda calculus.

Example 2.19 Let $\Delta = \lambda x.xx$. We may apply Δ to itself and obtain the following β-derivation:

$$(\lambda x.xx)\Delta \rightarrow_\beta (xx)\{x \leftarrow \Delta\} = \Delta\Delta \rightarrow_\beta \Delta\Delta \rightarrow_\beta \ldots$$

Since we have identified α-equivalent terms it must be verified that the β-rewrite relation is well-defined. Indeed, the following result holds:

Lemma 2.20 If $M =_\alpha M'$ and $M \rightarrow_\beta N$ then there exists a lambda term $N' =_\alpha N$ such that $M' \rightarrow_\beta N'$.

Two further properties satisfied by the β-rewrite relation [Bar84] are:

Lemma 2.21 If $M \rightarrow_\beta M'$, then for every lambda term N we have:

1. $M\{x \leftarrow N\} \rightarrow_\beta M'\{x \leftarrow N\}$.
2. $N\{x \leftarrow M\} \twoheadrightarrow_\beta N\{x \leftarrow M'\}$.

This may be proved by induction on M in the first item, and by induction on N in the second one.

The abstract rewrite system induced by the lambda calculus is obtained by setting $A = \mathcal{T}_\lambda$ and $R = \rightarrow_\beta$. We thus say that the lambda calculus is confluent if the induced abstract rewrite system is, and likewise for the other notions we saw in the previous section. Example 2.19 shows that the lambda calculus is not strongly normalizing.

Proposition 2.22 (Confluence of the lambda calculus) The *ARS* $(\mathcal{T}_\lambda, \rightarrow_\beta)$ is confluent.

See [Bar84] for a proof.

[4] Sometimes, just the axiom is provided and the fact that the contextual closure must be taken is left implicit. In this case the axiom is called "rule" and so we speak of the β-rule.

2.2.2 The Lambda Calculus with De Bruijn Indices

The fact that the definition of substitution (Def. 2.14) is not a function on terms but rather on equivalence classes of terms may be seen as a drawback when implementing it on a computer. In particular, the selection of some appropriate variable z requires checking that it is not used already (we say z is a *fresh* variable). A simple solution[5] to this problem is that of replacing variable names (an 'absolute addressing' mechanism) with de Bruijn indices [Bru72, Bru78] (a 'relative addressing' mechanism). A given occurrence of a bound variable in a term, say x, is replaced by a number which indicates to which binder it corresponds, counting upwards in the tree representation of the term starting from the occurrence of this variable. For example, $\lambda x.(\lambda y.(xy))$ is replaced by $\lambda(\lambda(21))$. This situation is depicted in Figure 2.2, where the '@' symbol stands for application. Note that although the tree representation of a term has not yet been defined formally, we leave it on intuitive level for the time being. A formal definition is given in Chapter 8 (Section 8.2).

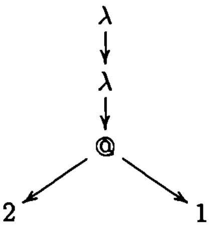


Figure 2.2: An example de Bruijn term

The identity function $\lambda x.x$ is replaced by $\lambda 1$, also $\lambda y.y$ is replaced by $\lambda 1$. Representation of free variables is retrieved from a given ordering of the variables in such a way that an occurrence of a variable n represents the $(n-m)$-th free variable in the aforementioned ordering when $n > m$ and there are m lambda binders above this occurrence n. For example, the term $\lambda x.(yz)$ is written as $\lambda(23)$ assuming $y = x_1$ and $z = x_2$.

Since variable names are no longer used there is no need for α-conversion. So can we say that substitution in the presence of de Bruijn indices notation is reduced to replacement? Certainly not. Consider $(\lambda 1)\{\!\{1 \leftarrow a\}\!\}$ where a is any indexed term, and $b\{\!\{n \leftarrow c\}\!\}$ stands for the result of replacing all occurrences of the index n in b for c. Then if we use replacement we obtain $(\lambda 1)\{\!\{1 \leftarrow a\}\!\} = \lambda a$, an unexpected result since the index 1 in $\lambda 1$ represents a bound variable and hence cannot be substituted for any term at all. When traversing a lambda symbol the index to be replaced should be incremented by one.

A further inconvenience may arise with unwanted capture of free indices (i.e. indices representing free variables), analogous to the case of variable names already discussed. An example is $(\lambda 2)\{\!\{1 \leftarrow 1\}\!\}$. Assuming the above mentioned problem on bound indices has been solved, we have,

$$(\lambda 2)\{\!\{1 \leftarrow 1\}\!\} \;=\; \lambda(2\{\!\{2 \leftarrow 1\}\!\}) \;=\; \lambda 1$$

The index 1 which is substituted for 2 (i.e. the rightmost occurrence of the symbol 1 in the expression $(\lambda 2)\{\!\{1 \leftarrow 1\}\!\}$) represents the first variable in the reference context, however after the substitution has taken place the same index 1 is bound in $\lambda 1$. Thus when traversing a lambda binder substitution shall have to do some index adjusting.

All in all the de Bruijn indices notation takes care of some problems (non-determinism in the definition of substitution) but introduces others (index adjustment).

We now introduce the formal definitions. The lambda calculus as introduced in Section 2.2.1 shall be referred to as the *named lambda calculus*.

Definition 2.23 • The de Bruijn terms of the indexed lambda calculus, denoted $\mathcal{T}_{\lambda_{db}}$, are defined as:

$$a ::= n \mid (aa) \mid (\lambda a)$$

where n is a natural number greater than zero. Notational conventions similar to those of the named lambda calculus are adopted in order not to clutter the notation.

[5] There are other possible solutions which are not dealt with in this thesis such as the use of de Bruijn levels [LRD95] and Berklings notation [Ste00].

- The set of free indices of a de Bruijn term a, denoted $FI(a)$, is defined as:

$$\begin{array}{lcl} FI(n) & \overset{\text{def}}{=} & \{n\} \\ FI(a_1a_2) & \overset{\text{def}}{=} & FI(a_1) \cup FI(a_2) \\ FI(\lambda a_1) & \overset{\text{def}}{=} & FI(a_1)\backslash\!\!\backslash 1 \end{array}$$

  where for every set of indices S, the operation $S\backslash\!\!\backslash j$ is defined as $\{n - j \mid n \in S \text{ and } n > j\}$.

There are some presentations of de Bruijn indices notation in which the first index is 0 rather than 1. However, this is largely a matter of taste.

We shall use letters $a, b, c, \ldots$ for indexed lambda terms. The subterm relation is defined as in the first-order case by considering the signature $(\Sigma_f, \mathcal{V})$ where Σ_f consists of the application operation, the lambda binder and all the indices interpreted as constants. The full set of first-order terms thus obtained is called the set of *open* de Bruijn terms. We restrict our attention to the induced subterm relation over the restricted set of *closed* terms, that is, those terms that do not have occurrences of first-order variables (objects in $\mathcal{V}$).

We shall now consider the definition of substitution in the de Bruijn indices setting. As mentioned above, this requires introducing a family of functions that perform index updating, namely the *updating functions*.

Definition 2.24 (Indexed substitution) The result of substituting a term b for the index n in a term a is denoted $a\{\!\{n \leftarrow b\}\!\}$ and defined:

$$\begin{array}{lcl} (a_1a_2)\{\!\{n \leftarrow b\}\!\} & \overset{\text{def}}{=} & a_1\{\!\{n \leftarrow b\}\!\}a_2\{\!\{n \leftarrow b\}\!\} \\ (\lambda a_1\{\!\{n \leftarrow b\}\!\} & \overset{\text{def}}{=} & \lambda(a_1\{\!\{n+1 \leftarrow b\}\!\}) \\ m\{\!\{n \leftarrow b\}\!\} & \overset{\text{def}}{=} & \begin{cases} m-1 & \text{if } m > n \\ \mathcal{U}_0^n(b) & \text{if } m = n \\ m & \text{if } m < n \end{cases} \end{array}$$

where for $i \geq 0$ and $n \geq 1$ we define the *updating functions* $\mathcal{U}_i^n(\bullet)$ as follows:

$$\begin{array}{lcl} \mathcal{U}_i^n(a_1a_2) & \overset{\text{def}}{=} & \mathcal{U}_i^n(a_1)\mathcal{U}_i^n(a_n) \\ \mathcal{U}_i^n(\lambda a_1) & \overset{\text{def}}{=} & \lambda(\mathcal{U}_{i+1}^n(a_1)) \\ \mathcal{U}_i^n(m) & \overset{\text{def}}{=} & \begin{cases} m+n-1 & \text{if } m > i \\ m & \text{if } m \leq i \end{cases} \end{array}$$

Note how the index n is incremented by one unit when substitution traverses a lambda binder in the second clause of the definition of substitution. Also, the reason for defining $m\{\!\{n \leftarrow b\}\!\} \overset{\text{def}}{=} m - 1$ if $m > n$ is that de Bruijn substitution shall be generated by the (de Bruijn version of the) β-rewrite rule, hence the decrement follows from the fact that a lambda binder has been eliminated. So one might say that this definition is tailored for the lambda calculus.

A minor simplification is to define $m\{\!\{n \leftarrow b\}\!\} \overset{\text{def}}{=} \mathcal{U}_0^{n-1}(b)$ when $m = n$ and modify the definition of the updating functions accordingly by allowing superindices to be zero and defining $\mathcal{U}_i^n(m) \overset{\text{def}}{=} m + n$ when $m > i$.

The Substitution Lemma (Lemma 2.17) may also be formulated in the indexed lambda calculus. In this formulation condition $x \notin FV(O)$ of Lemma 2.17 is reflected as a condition on the indices in the de Bruijn indices setting, namely that $i \leq n$.

Lemma 2.25 (Substitution Lemma for the Indexed Lambda Calculus) Let $a, b, c \in \mathcal{T}_{\lambda_{db}}$. Then for all $n, i \geq 1$ such that $i \leq n$ we have $a\{\!\{i \leftarrow b\}\!\}\{\!\{n \leftarrow c\}\!\} = a\{\!\{n+1 \leftarrow c\}\!\}\{\!\{i \leftarrow b\{\!\{n-i+1 \leftarrow c\}\!\}\}\!\}$.

The β_{db}-rewrite rule is stated as follows.

Definition 2.26 (β_{db}-rewrite rule) We say a *β_{db}-rewrites* to b iff $a \rightarrow_{\beta_{db}} b$, where the latter relation is defined by the following inference schemes:

$$\frac{}{(\lambda a)b \rightarrow_{\beta_{db}} a\{\!\{1 \leftarrow b\}\!\}} \qquad \frac{a_1 \rightarrow_{\beta_{db}} b_1}{a_1a_2 \rightarrow_{\beta_{db}} b_1a_2} \qquad \frac{a_2 \rightarrow_{\beta_{db}} b_2}{a_1a_2 \rightarrow_{\beta_{db}} a_1b_2} \qquad \frac{a \rightarrow_{\beta_{db}} b}{\lambda a \rightarrow_{\beta_{db}} \lambda b}$$

The leftmost inference scheme of Def. 2.26 is called the β_{db}*-rewrite axiom*. Once again, it is not uncommon to present the β_{db}-rewrite rule by taking the contextual closure of the β_{db}-rewrite axiom, and exhibiting just this axiom. In the indexed lambda calculus the infinite derivation of Example 2.19 takes the following form.

Example 2.27 Let $\Delta_{db} = \lambda(11)$. We may apply Δ_{db} to itself and obtain the following β_{db}-derivation:

$$\begin{array}{lll}
(\lambda(11))\Delta_{db} & \rightarrow_{\beta_{db}} & (11)\{\!\{1 \leftarrow \Delta_{db}\}\!\} \\
 & = & 1\{\!\{1 \leftarrow \Delta_{db}\}\!\}1\{\!\{1 \leftarrow \Delta_{db}\}\!\} \\
 & = & \mathcal{U}_0^1(\Delta_{db})\mathcal{U}_0^1(\Delta_{db}) \\
 & = & \Delta_{db}\Delta_{db} \\
 & \rightarrow_{\beta_{db}} & \Delta_{db}\Delta_{db} \\
 & \rightarrow_{\beta_{db}} & \cdots
\end{array}$$

Note that $\mathcal{U}_0^1(\Delta_{db}) = \lambda(\mathcal{U}_1^1(1)\mathcal{U}_1^1(1)) = \lambda(11)$ since $1 \leq 1$.

As in the named lambda calculus, the indexed lambda calculus induces an abstract rewrite system by taking the pair (A, R) where $A = \mathcal{T}_{\lambda_{db}}$ and $R = \rightarrow_{\beta_{db}}$.

Proposition 2.28 (Confluence of the indexed lambda calculus) The *ARS* $(\mathcal{T}_{\lambda_{db}}, \rightarrow_{\beta_{db}})$ is confluent.

See [KR98] for a proof. The latter proof is based on translations between terms with variables and indexed terms.

2.3 Calculi of Explicit Substitutions

This section introduces the concept of explicit substitutions and provides a brief overview of some calculi of explicit substitutions. The central idea is to depart from the lambda calculus and introduce new rules for computing the substitution process from *within* the language, rather than interpret it as a metalevel operation. However since we have provided the reader with two different presentations of the lambda calculus, the named lambda calculus and the indexed lambda calculus, the calculi of explicit substitutions which we shall overview in this section shall, accordingly, be based either on the named lambda calculus or on the indexed lambda calculus. Since variable names provide a more user-friendly environment we shall begin the section by considering the λx-calculus. After that, three calculi of explicit substitutions based on the indexed lambda calculus shall be dealt with, λv, $\lambda\sigma$ and λws. Note that this amounts to a small fraction of the calculi of explicit substitutions published in the literature. A recent survey including pointers to relevant literature is [KR00].

2.3.1 The λx-calculus

The λx-calculus [Ros92, Blo97] is a calculus of explicit substitutions for the named lambda calculus. We shall follow R.Bloo's exposition of the calculus from [Blo97].

As the reader may recall, the β-rewrite axiom (Def. 2.18) takes the following form:

$$\frac{}{(\lambda x.M)N \rightarrow_\beta M\{x \leftarrow N\}}$$

The expression $M\{x \leftarrow N\}$ denotes a term, namely the one resulting from M by substituting all free occurrences of the variable x in M with N. Therefore, the substitution takes place in one go, as an atomic operation. In calculi of explicit substitutions this operation is computed from within the calculus by means of new rewrite rules. The β-rewrite axiom is replaced by the following *Beta*-rewrite axiom:

$$\frac{}{(\lambda x.M)N \rightarrow_\beta M\langle x := N\rangle}$$

This time, the expression $M\langle x := N\rangle$ is just a new term in the language, and no substitution operation is fired. One may regard $M\langle x := N\rangle$ as a term with a pending substitution. This pending substitution is called a *substitution* or a *closure*. The intended meaning of $M\langle x := N\rangle$ is of course the term resulting from substituting all free variable occurrences of x in M by N. However since substitution is left pending in $M\langle x := N\rangle$ new rules must be introduced in order to compute it. These additional rules determine the *substitution calculus*.

As a result one β-rewrite step shall be refined into a series of smaller rewrite steps in the calculus of explicit substitutions. First a *Beta*-rewrite step shall create a closure, then the rules of the substitution calculus are put to work in order to compute the pending substitution.

Definition 2.29 Let $\mathcal{V}$ be a denumerably infinite set of variables. The terms of λx, denoted $\mathcal{T}_{\lambda x}$, are defined inductively as follows:

$$M ::= x \mid (MM) \mid (\lambda x.M) \mid (M\langle x := M\rangle)$$

where x ranges over $\mathcal{V}$. A term is called *pure* if it contains no subterms of the form $M_1\langle x := M_2\rangle$. Similar notational conventions to that of the lambda calculus are adopted. Furthermore, we shall adopt the convention that substitution associates to the left and binds stronger than both application and abstraction.

It is instructive to compare the definition of the λx-terms to that of the terms of the lambda calculus (Def. 2.13). An extra binder has been added. We shall use letters $M, N, O, P, \ldots$ for λx-terms, and u, v, w, x, y, z for variables. A term of the form '$M\langle x := N\rangle$' is pronounced 'M where N is substituted for x'. We call M the *target* of the substitution, N its *body*, and x the *substitution variable*.

In Chapter 3 we shall make use of the size of a λx-term. The size of a λx-term is the number of variables, applications, abstractions and substitutions in it. This may be formalized as follows:

Definition 2.30 The size of a λx-term M is written $|M|$ and defined inductively as follows:

$$\begin{array}{lll}
|x| & \overset{\text{def}}{=} & 1 \\
|M_1M_2| & \overset{\text{def}}{=} & 1 + |M_1| + |M_2| \\
|\lambda x.M| & \overset{\text{def}}{=} & 1 + |M| \\
|M_1\langle x := M_2\rangle| & \overset{\text{def}}{=} & 1 + |M_1| + |M_2|
\end{array}$$

For example, $|\lambda x.(yy)\langle z := u\rangle|$ is 6.

Definition 2.31 The set of free variables of a λx-term M, denoted $FV(M)$, is defined inductively as follows:

$$\begin{array}{lll}
FV(x) & \overset{\text{def}}{=} & \{x\} \\
FV(M_1M_2) & \overset{\text{def}}{=} & FV(M_1) \cup FV(M_2) \\
FV(\lambda x.M) & \overset{\text{def}}{=} & FV(M) \setminus \{x\} \\
FV(M_1\langle x := M_2\rangle) & \overset{\text{def}}{=} & (FV(M_1) \setminus \{x\}) \cup FV(M_2)
\end{array}$$

From this definition one may see that a closure acts as a new binder in the language, in a term $M_1\langle x := M_2\rangle$ all free variable occurrences of x in M_1 are said to be bound by the closure.

Since we are still in the presence of variable names in the extended language of λx we must extend the notion of α-conversion accordingly. This requires first defining metalevel substitution. Note however that metalevel substitution shall not be used in the rewriting process, except for possible renamings of bound variables.

Definition 2.32 (Metalevel Substitution and α-conversion)

- Substitution in λx is defined inductively as follows:

$$\begin{array}{lll}
x\{x \leftarrow N\} & \overset{\text{def}}{=} & N \\
y\{x \leftarrow N\} & \overset{\text{def}}{=} & y \quad \text{if } x \neq y \\
(M_1M_2)\{x \leftarrow N\} & \overset{\text{def}}{=} & M_1\{x \leftarrow N\}M_2\{x \leftarrow N\} \\
(\lambda y.M_1)\{x \leftarrow N\} & \overset{\text{def}}{=} & \lambda z.M_1\{y \leftarrow z\}\{x \leftarrow N\} \\
 & & \text{where } z \text{ does not occur at all in } \lambda y.M_1,\ x \text{ or } N \\
M_1\langle y := M_2\rangle\{x \leftarrow N\} & \overset{\text{def}}{=} & M_1\{y \leftarrow z\}\{x \leftarrow N\}\langle z := M_2\{x \leftarrow N\}\rangle \\
 & & \text{where } z \text{ does not occur at all in } M_1, M_2, y \text{ or } N
\end{array}$$

- α-conversion (denoted $=_\alpha$) on λx-terms is defined as the least equivalence relation such that:

$$\begin{array}{ll}
\text{if } M\{x \leftarrow y\} =_\alpha N & \text{then } \lambda x.M =_\alpha \lambda y.N \\
\text{if } M_1 =_\alpha N_1 \text{ and } M_2 =_\alpha N_2 & \text{then } M_1M_2 =_\alpha N_1N_2 \\
\text{if } M_1\{x \leftarrow y\} =_\alpha N_1 \text{ and } M_2 =_\alpha N_2 & \text{then } M_1\langle x := M_2\rangle =_\alpha N_1\langle y := N_2\rangle
\end{array}$$

We write $[M]_\alpha$ for the α-equivalence class of M.

A variable convention similar to the one adopted for the lambda calculus is adopted here.

Remark 2.33 (Variable Convention for λx) Names of bound variables are different from the names of free variables, and moreover, different occurrences of the lambda binder and the substitution binder have different binding variables.

We now consider the rewrite rules of the λx-calculus.

Definition 2.34 We say M λx*-rewrites* to N iff $M \rightarrow_{\lambda x} N$, where the latter relation is defined by the following inference schemes:

$$\frac{}{(\lambda x.M)N \rightarrow_{\lambda x} M\langle x := N\rangle}\ \mathit{Beta} \qquad \frac{}{(MN)\langle x := P\rangle \rightarrow_{\lambda x} M\langle x := P\rangle N\langle x := P\rangle}\ \mathit{App}$$

$$\frac{}{(\lambda y.M)\langle x := P\rangle \rightarrow_{\lambda x} \lambda y.(M\langle x := P\rangle)}\ \mathit{Lam} \qquad \frac{}{x\langle x := P\rangle \rightarrow_{\lambda x} P}\ \mathit{Var}$$

$$\frac{x \notin FV(M)}{M\langle x := P\rangle \rightarrow_{\lambda x} M}\ \mathit{Gc}$$

$$\frac{M_1 \rightarrow_{\lambda x} N_1}{M_1M_2 \rightarrow_{\lambda x} N_1M_2} \qquad \frac{M_2 \rightarrow_{\lambda x} N_2}{M_1M_2 \rightarrow_{\lambda x} M_1N_2}$$

$$\frac{M \rightarrow_{\lambda x} N}{\lambda x.M \rightarrow_{\lambda x} \lambda x.N} \qquad \frac{M_1 \rightarrow_{\lambda x} N_1}{M_1\langle x := M_2\rangle \rightarrow_{\lambda x} N_1\langle x := M_2\rangle}$$

$$\frac{M_2 \rightarrow_{\lambda x} N_2}{M_1\langle x := M_2\rangle \rightarrow_{\lambda x} M_1\langle x := N_2\rangle}$$

An equivalent presentation, as discussed in Def. 2.18 for the λ-calculus, is given in Figure 2.3. If these rules are adopted as axiom schemes and the contextual closure is taken (meaning that the remaining inference schemes of Def. 2.34 are added to these axiom schemes) then we obtain the inference schemes of Def. 2.34. So the presentation of Figure 2.3 may be seen as a shorthand for that of Def. 2.34.

$$\begin{array}{llll}
(\lambda x.M)N & \rightarrow_{Beta} & M\langle x := N\rangle & \\
(MN)\langle x := P\rangle & \rightarrow_{App} & M\langle x := P\rangle N\langle x := P\rangle & \\
(\lambda y.M)\langle x := P\rangle & \rightarrow_{Lam} & \lambda y.(M\langle x := P\rangle) & \\
x\langle x := P\rangle & \rightarrow_{Var} & P & \\
M\langle x := P\rangle & \rightarrow_{Gc} & M & x \notin FV(M)
\end{array}$$

Figure 2.3: Alternative presentation of λx

The *Gc*-rule is referred to as the *garbage collection rule*; the substitution body P in this rule is called *garbage*. The λx-calculus without the *Beta*-rule is called the *substitution calculus of* λx and is denoted by x. We write x $\setminus$ *Gc* for the rewrite system x without the *Gc*-rule. The x-calculus is strongly normalizing and confluent, and its normal forms are pure terms [Blo97, Ch.13].

Lemma 2.35 The x-calculus is strongly normalizing, confluent, and the x-normal forms are pure terms.

Thus if $M \in \mathcal{T}_{\lambda x}$ then we use x(M) to denote its unique x-normal form. The reason for calling x the substitution calculus is that the following result holds:

Lemma 2.36 Let $M, N \in \mathcal{T}_{\lambda x}$ and $y \in \mathcal{V}$. Then $\mathrm{x}(M\langle y := N\rangle) = \mathrm{x}(M)\{y \leftarrow \mathrm{x}(N)\}$.

Thus indeed the usual metalevel notion of substitution (Def. 2.14) may be computed step-by-step via the x-calculus.

A variant of the λx-calculus is the λx$^-$-calculus whose rules are those of the λx-calculus except for the *Gc*-rule which is replaced by the more *restricted garbage collection rule* $y\langle x := P\rangle \rightarrow_{Varf} y$ where $x \neq y$. Note that λx is more general than λx$^-$ in the sense that $\rightarrow_{\lambda x^-} \subset \rightarrow_{\lambda x}$ but $\rightarrow_{\lambda x} \not\subset \rightarrow_{\lambda x^-}$.

Remark 2.37 Every λx^--redex is a λx-redex.

The abstract rewrite system (A, R) induced by λx is obtained by setting $A = \mathcal{T}_{\lambda x}$ and $R = \rightarrow_{\lambda x}$. Thus notions such as confluence and normalization make sense in λx too. For example,

- $SN_{\lambda x}$ stands for the set of λx-terms that are strongly λx-normalizing.
- If $M \in SN_{\lambda x}$ then $maxred_{\lambda x}(M)$ denotes the length of the longest λx-derivation starting from M to its (unique, see Lemma 2.39 below) λx-normal form. Note that since there are only a finite number of λx-redexes in any λx-term, we may conclude λx is finitely branching.

Definition 2.38 A substitution $\langle x := N\rangle$ is called *void in* $M\langle x := N\rangle$ if $x \notin FV(\mathrm{x}(M))$. *Void reduction*, denoted $\overset{v}{\rightarrow}_{\lambda x}$, is λx-rewriting inside the body of a void substitution. More precisely, it is obtained by the inference scheme:

$$\frac{N \rightarrow_{\lambda x} N' \quad x \notin FV(\mathrm{x}(M))}{M\langle x := N\rangle \rightarrow_{\lambda x} M\langle x := N'\rangle}$$

together with the last five inference schemes of Def. 2.34[6].

The following two results are taken from [Blo97].

Lemma 2.39 (Simulation, Projection, Confluence)

- (Simulation). Let M, N be terms in $\mathcal{T}_\lambda$. If $M \rightarrow_\beta N$ then $M \xrightarrow{+}_{\lambda x} N$.
- (Projection). Let M, N be terms in $\mathcal{T}_{\lambda x}$.
  1. If $M \rightarrow_{\lambda x} N$ then $\mathrm{x}(M) \twoheadrightarrow_\beta \mathrm{x}(N)$.
  2. If $M \rightarrow_{Beta} N$ is not a void reduction then $\mathrm{x}(M) \xrightarrow{+}_\beta \mathrm{x}(N)$.
- (Confluence). The *ARS* $(\mathcal{T}_{\lambda x}, \rightarrow_{\lambda x})$ is confluent.

Perhaps the most interesting property enjoyed by λx is preservation of strong normalization (PSN): if a is strongly β-normalizing then a is strongly λx-normalizing. The proof of this fact is not straightforward. Several techniques have been introduced for proving PSN of which we shall see a few in Part I of this thesis. As regards the λx-calculus in particular, the most appropriate source for further information is [Blo97]. We provide a new proof of this result in Section 4.1.2, of Chapter 3.

Proposition 2.40 Let $\lambda x^\infty \overset{\text{def}}{=} \{M \in \mathcal{T}_{\lambda x} \mid \forall N \subseteq M,\ \mathrm{x}(N) \in SN_{\lambda x}\}$. Then $\lambda x^\infty = SN_{\lambda x}$.

The above characterization of $SN_{\lambda x}$ first appeared in [BG96]. The same work includes in the list of future research lines the study of an *inductive* characterization of the set $SN_{\lambda x}$. By inductively we mean describing the set as the smallest set closed under some set of rules, as when defining the set of λ-terms or the set of theorems of some logic system [Acz77]. This shall be dealt with in Chapter 3. Note that since all strongly β-normalizing pure terms are in λx^∞ one obtains PSN of λx.

Proposition 2.41 (PSN of λx) The λx-calculus enjoys preservation of strong normalization.

Finally, we define λx-rewrite strategies. They shall be used in Chapters 3 and 4.

Definition 2.42 (λx-Rewrite Strategy)

1. A one-step λx-*rewrite strategy* is a function $\mathcal{F}(\bullet) : \mathcal{T}_{\lambda x} \longrightarrow \mathcal{T}_{\lambda x}$ such that for all $M \in \mathcal{T}_{\lambda x}$ we have $M \rightarrow_{\lambda x} \mathcal{F}(M)$, unless M is in normal form, in which case $\mathcal{F}(M) = M$.
2. A many-step λx-*rewrite strategy* is a function $\mathcal{F}(\bullet) : \mathcal{T}_{\lambda x} \longrightarrow \mathcal{T}_{\lambda x}$ such that for all $M \in \mathcal{T}_{\lambda x}$ we have $M \xrightarrow{+}_{\lambda x} \mathcal{F}(M)$, unless M is in normal form, in which case $\mathcal{F}(M) = M$.
3. A strategy is called λx-*perpetual* if $\infty_{\lambda x}(M)$ implies $\infty_{\lambda x}(\mathcal{F}(M))$.

[6] Where the occurrences of $\rightarrow_{\lambda x}$ have been replaced by $\overset{v}{\rightarrow}_{\lambda x}$.

If $\mathcal{F}(\bullet)$ is a one-step λx-rewrite strategy then an $\mathcal{F}(\bullet)$-derivation is a derivation of the form $M \rightarrow_{\lambda x} \mathcal{F}(M) \rightarrow_{\lambda x} \mathcal{F}(\mathcal{F}(M)) \rightarrow_{\lambda x} \ldots$. Similarly, if $\mathcal{F}(\bullet)$ is a many-step λx-rewrite strategy then an $\mathcal{F}(\bullet)$-derivation is a derivation of the form $M \xrightarrow{+}_{\lambda x} \mathcal{F}(M) \xrightarrow{+}_{\lambda x} \mathcal{F}(\mathcal{F}(M)) \xrightarrow{+}_{\lambda x} \ldots$. Likewise, we may define rewrite strategies for other calculi such as the lambda calculus. If our interest in λx-strategies is that they construct valid λx-derivations in a *deterministic* manner then Def. 2.42 shall suffice. However, note the following:

- A strategy as defined above does not necessarily indicate which redex occurrence to contract in order to go from M to $\mathcal{F}(M)$, it is possible that there be more than one. Consider for example the term $M = x\langle y := N\rangle\langle y := N\rangle$ where $y \notin FV(N)$ and $x \neq y$. Then $M \rightarrow_{\lambda x} x\langle y := N\rangle$ by rewriting two different Gc-redex occurrences which we indicate by underlining them: $\underline{x\langle y := N\rangle\langle y := N\rangle}$ or $\underline{x\langle y := N\rangle}\langle y := N\rangle$.
- A further source of ambiguity is when a redex occurrence does not suffice to determine which rewrite rule should selected to rewrite, this occurs in rewrite systems where two different rules overlap at the root position and the result of applying either of them is the same (see the example of the parallel-or rewrite system in [KOO01a]).

In Chapter 3 we shall study perpetual rewrite strategies for λx. When formulating the strategies we shall indicate the redex occurrence to be contracted and the rewrite rule too.

2.3.2 The λv-calculus

The first calculus of explicit substitutions based on de Bruijn indices notation that we shall look at is the λv-calculus of P.Lescanne [Les94, BBLRD96].

Definition 2.43 The terms of the λv-calculus, denoted $\mathcal{T}_{\lambda v}$, are defined by the following two sorted grammar:

$$\begin{array}{lll} \texttt{Terms}\ M & ::= & n \mid (MM) \mid (\lambda M) \mid (M[s]) \\ \texttt{Subst}\ s & ::= & \uparrow \mid M/ \mid \Uparrow(s) \end{array}$$

where n is a natural number greater than zero. Once again, the '$\subseteq$' symbol is used for the subterm relation, and it is defined as expected. Terms of sort `Terms` without subterms of the form $M[s]$ are called *pure terms*.

In contrast to the λx-calculus terms may be of two sorts, namely `Terms` or `Subst`. Terms of the former sort speak for themselves, terms of the latter sort represent substitutions. A substitution may have one of the following forms: either it is a *shift* substitution ($\uparrow$), or it is a *simple* substitution ($M/$), or it is a *lift* of a substitution s ($\Uparrow(s)$). The usual conventions that application and $\bullet[\bullet]$ associate to the left, and that $\bullet[\bullet]$ binds stronger than application which binds stronger than λ, are adopted. Also repeated application of lift is defined as follows: $\Uparrow^0(s) \stackrel{\text{def}}{=} s$, $\Uparrow^{n+1}(s) = \Uparrow(\Uparrow^n(s))$. All substitutions in λv are of the form $\Uparrow^n(M/)$ or $\Uparrow^n(\uparrow)$ for n some natural number.

Definition 2.44 We say M *λv-rewrites* to N iff $M \rightarrow_{\lambda v} N$, where the latter relation is defined by the contextual closure of the following inference axioms:

$$\begin{array}{lll} (\lambda M)N & \rightarrow_{Beta} & M[N/] \\ \\ (MN)[s] & \rightarrow_{App} & M[s]N[s] \\ (\lambda M)[s] & \rightarrow_{Lam} & \lambda(M[\Uparrow(s)]) \\ (n+1)[\Uparrow(s)] & \rightarrow_{RVarLift} & n[s][\uparrow] \\ 1[\Uparrow(s)] & \rightarrow_{FVarLift} & 1 \\ (n+1)[M/] & \rightarrow_{RVar} & n \\ 1[M/] & \rightarrow_{FVar} & M \\ n[\uparrow] & \rightarrow_{VarShift} & n+1 \end{array}$$

The λv-calculus without the *Beta*-rewrite rule is the substitution calculus, or the v-calculus. The shift and simple substitutions may be seen as basic substitutions, whereas the lift operator adjusts substitutions. Shift represents the substitution which increments all the free indices of its target by one unit. For example, $1[\uparrow] \rightarrow_{\lambda v} 2$. However $(\lambda 1)[\uparrow] \twoheadrightarrow_{\lambda v} \lambda 1$, since the index 1 is not free in the term $\lambda 1$.

Note how the lift substitution operator is introduced when a substitution traverses the lambda binder in rule *Lam*. $\Uparrow(s)$ indicates that if s assigns some term M to index n then it should now do so to index $n+1$.

Also it indicates that if M is ever assigned to some index $n+1$ then the free indices in M must de adjusted since they shall now appear under a new lambda binder (the one s has just traversed). These two observations are expressed by the rules *RVarLift* and *FVarLift*.

Lemma 2.45 The υ-calculus is strongly normalizing, confluent, and the υ-normal forms are pure terms.

The relation between explicit substitution and the updating operators and implicit substitutions is expressed in the following lemma.

Lemma 2.46 (Relating explicit and metalevel substitution) Let $M, N \in \mathcal{T}_{\lambda\upsilon}$ of sort $\texttt{Terms}$. Then:

- $\upsilon(M[\Uparrow^{m_1}(\uparrow)]\dots[\Uparrow^{m_n}(\uparrow)]) = \mathcal{U}^{n+1}_{m_1+\dots+m_n}(\upsilon(M))$.
- $\upsilon(M[\Uparrow^{n}(N)]) = \upsilon(M)\{\!\{n \leftarrow \upsilon(N)\}\!\}$.

Other properties of the $\lambda\upsilon$-calculus are:

Lemma 2.47 (Simulation, Projection, Confluence) • (Simulation). Let M, N be terms in $\mathcal{T}_{\lambda_{db}}$. If $M \to_{\beta_{db}} N$ then $M \xrightarrow{+}_{\lambda\upsilon} N$.

- (Projection). Let $M, N \in \mathcal{T}_{\lambda\upsilon}$ of sort $\texttt{Terms}$. If $M \to_{\lambda\upsilon} N$ then $\upsilon(M) \twoheadrightarrow_{\beta_{db}} \upsilon(N)$.
- (Confluence). The *ARS* $(\mathcal{T}_{\lambda\upsilon}, \to_{\lambda\upsilon})$ is confluent.

As λx, $\lambda\upsilon$ also enjoys preservation of strong normalization [BBLRD96]. The technique which is used to prove this result shall be studied in detail in Chapter 3.

Proposition 2.48 (PSN of $\lambda\upsilon$) The $\lambda\upsilon$-calculus enjoys preservation of strong normalization.

2.3.3 The $\lambda\sigma$-calculus

The $\lambda\sigma$-calculus was introduced by M.Abadi et al [ACCL91] and has its roots in the work of P-L.Curien [Cur86]. It is usually regarded as the first calculus of explicit substitutions whose properties such as confluence and strong normalization of the substitution calculus were studied. For some time it was not known if $\lambda\sigma$ enjoyed preservation of strong normalization until P-A.Melliès introduced a counterexample in [Mel95]. An analysis of this counterexample is provided in R.Bloo's PhD thesis [Blo97].

Definition 2.49 The terms of the $\lambda\sigma$-calculus, denoted $\mathcal{T}_{\lambda\sigma}$, are defined by the following two sorted grammar:

$$\begin{array}{lll}
\texttt{Terms}\ M & ::= & 1 \mid (MM) \mid (\lambda M) \mid (M[s]) \\
\texttt{Subst}\ s & ::= & \uparrow \mid id \mid M \cdot s \mid s \circ s
\end{array}$$

Indices greater than 1 are expressed with the aid of explicit substitutions: $2 \stackrel{\text{def}}{=} 1[\uparrow]$, $3 \stackrel{\text{def}}{=} 1[\uparrow \circ \uparrow]$, and so on. Once again, the '$\subseteq$' symbol is used for the subterm relation, and it is defined as expected. Terms of sort $\texttt{Terms}$ without subterms of the form $M[s]$ are called *pure terms*.

Definition 2.50 We say M *$\lambda\sigma$-rewrites* to N iff $a \to_{\lambda\sigma} b$, where the latter relation is defined by the contextual closure of the following inference schemes:

$$\begin{array}{lll}
(\lambda M)N & \to_{Beta} & M[N \cdot id] \\
 & & \\
(MN)[s] & \to_{App} & M[s]N[s] \\
(\lambda M)[s] & \to_{Lam} & \lambda(M[1 \cdot (s \circ \uparrow)]) \\
M[s][t] & \to_{Clos} & M[s \circ t] \\
1[M \cdot s] & \to_{VarCons} & M \\
1[id] & \to_{VarId} & 1 \\
 & & \\
(M \cdot s) \circ t & \to_{Map} & M[t] \cdot (s \circ t) \\
id \circ s & \to_{IdL} & s \\
(s_1 \circ s_2) \circ s_3 & \to_{Ass} & s_1 \circ (s_2 \circ s_3) \\
\uparrow \circ (M \cdot s) & \to_{ShiftCons} & s \\
\uparrow \circ id & \to_{ShiftId} & \uparrow
\end{array}$$

Although both $\lambda\upsilon$ and $\lambda\sigma$ have terms of sort `Terms` and `Subst`, only $\lambda\sigma$ provides inference schemes for rewriting substitutions. Indeed, the last five rules of Def. 2.50 rewrite terms of sort `Subst`. Also, in contrast to $\lambda\upsilon$, $\lambda\sigma$ provides composition of substitutions, if s and t are substitutions (i.e. terms of sort `Subst`) then $s \circ t$ denotes their composition: first apply substitution s then apply substitution t.

$\lambda\sigma$ without the *Beta*-rewrite rule is called the substitution calculus and is denoted σ.

Lemma 2.51 The σ-calculus is strongly normalizing, confluent, and the σ-normal forms are pure terms.

See [ACCL91, Río93, Zan95].

Note that although there is no rule of the form $s \circ id \rightarrow s$ or of the form $M[id] \rightarrow M$ they are admissible in the sense that all its ground instances may be simulated in the σ-calculus [ACCL91].

The following lemma relates $\lambda\sigma$-reduction and β-reduction, and also mentions confluence.

Lemma 2.52 (Simulation, Projection, Confluence)

- (Simulation). Let M, N be terms in $\mathcal{T}_{\lambda_{db}}$. If $M \rightarrow_{\beta_{db}} N$ then $M \xrightarrow{+}_{\lambda\sigma} N$.
- (Projection). Let $M, N \in \mathcal{T}_{\lambda\sigma}$ of any sort. If $M \rightarrow_{\lambda\sigma} N$ then $\sigma(M) \twoheadrightarrow_{\beta_{db}} \sigma(N)$.
- (Confluence). The *ARS* $(\mathcal{T}_{\lambda\sigma}, \rightarrow_{\lambda\sigma})$ is confluent.

The $\lambda\sigma$-calculus exhibits a more complex dynamical nature than λx and $\lambda\upsilon$. This is due to the presence of substitution composition. A consequence of this fact is failure of preservation of strong normalization. This result was proved by P-A.Melliès [Mel95] and came somewhat as a surprise. Also, it has contributed significantly to boost researchers' interest in calculi of explicit substitutions.

2.3.4 The λws-calculus

The λws-calculus is a calculus of explicit substitutions based on de Bruijn indices notation which also enjoys the presence of substitution composition however in a different way than that of $\lambda\sigma$. Moreover, so called *update tags* are also present. A term with an update tag is of the form $\langle i\rangle M$, indicating that all free indices in M should be incremented by i units. Substitutions have the form $M[i/N, j]$ indicating that indices i must be replaced by $\langle i\rangle N$ and that there was a tag $\langle j\rangle$ embracing the *Beta*-redex that fired the substitution. The following rewrite step illustrates how substitutions may be composed in λws:

$$4\underbrace{[0/00,0]}_{s}\underbrace{[0/\lambda 00,0]}_{t} \rightarrow_{\lambda\text{ws}} 4\underbrace{[0/(00)[0/\lambda 00,0],1]}_{s o t}$$

There are no updating tags in the above two terms. Note that, as originally introduced in [DG99, DG01], de Bruijn indices start from 0 instead of 1.

As the reader may have noted there is no explicit operator in the language to denote the composition of substitutions as in the case of $\lambda\sigma$. In fact there is only one sort, namely the sort of terms. However, a substitution may be composed with another one in the sense of the example above or may even jump over another one as described in the $c2$ rewrite rule of Def. 2.58.

Just as $\lambda\upsilon$ 'implements' the indexed lambda calculus, λws also implements a calculus. This calculus, called λw, is very similar to the indexed lambda calculus but, as already mentioned, differs in that it includes so called *updating tags*. So before going into the details of the λws-calculus we shall briefly go over λw.

Definition 2.53 (The λw-calculus) The set of terms $M \in \mathcal{T}_{\lambda\text{w}}$ is defined as:

$$\begin{array}{lcll} M & ::= & N \mid \langle k\rangle N & \text{where } k \in \mathbb{N}_0 \\ N & ::= & n \mid \lambda M \mid MM & \text{where } n \in \mathbb{N}_0 \end{array}$$

The $\langle\bullet\rangle\bullet$ operator is the *updating* or *update operator*. Note that no two consecutive updating operators may appear in a term in $\mathcal{T}_{\lambda\text{w}}$. The updating operator has higher precedence than the application operator, so $\langle k\rangle PQ$ means $(\langle k\rangle P)Q$. The λw-calculus is defined on the set $\mathcal{T}_{\lambda\text{w}}$ by the rules:

$$\begin{array}{lcl} (\lambda M)N & \rightarrow_{\beta 1} & m(M\{0/N,0\}) \\ (\langle k\rangle\lambda M)N & \rightarrow_{\beta 2} & m(M\{0/N,k\}) \end{array}$$

where m is given by the m(ixing)-rule: $\langle i\rangle\langle j\rangle M \to_m \langle i+j\rangle M$. Metalevel substitution $\bullet\{\bullet/\bullet,\bullet\}$ is defined as:

$$\begin{array}{lcl}
(\lambda M)\{i/N,j\} & \overset{\text{def}}{=} & \lambda M\{i+1/N,j\} \\
(M_1 M_2)\{i/N,j\} & \overset{\text{def}}{=} & M_1\{i/N,j\}M_2\{i/N,j\} \\
(\langle k\rangle M)\{i/N,j\} & \overset{\text{def}}{=} & \begin{cases} \langle j+k-1\rangle M & i<k \\ \langle k\rangle M\{i-k/N,j\} & i \geq k \end{cases} \\
n\{i/N,j\} & \overset{\text{def}}{=} & \begin{cases} n & n<i \\ \langle i\rangle N & n=i \\ n+j-1 & n>i \end{cases}
\end{array}$$

where $i,j \in \mathbb{N}$.

Intuitively a term of the form $\langle k\rangle M$ represents the term M where all free indices have been incremented by k units. The mixing rule allows all adjacent updating tags to be fused, and hence the reducts of β_1 and β_2 to be terms in $\mathcal{T}_{\lambda\mathtt{w}}$. Note that it is the presence of the update tags that has forced the usual β_{db}-rewrite rule to be split in two, namely $\beta 1$ and $\beta 2$.

Example 2.54 Here is an example of a $\lambda\mathtt{w}$-derivation.

$$\begin{array}{lll}
(\lambda(00))\lambda(00) & \to_{\beta 1} & m((00)\{0/\lambda(00),0\}) = (\langle 0\rangle\lambda(00))\langle 0\rangle\lambda(00) \\
 & \to_{\beta 2} & m((00)\{0/\langle 0\rangle\lambda(00),0\}) = m((\langle 0\rangle\langle 0\rangle\lambda(00))\langle 0\rangle\langle 0\rangle\lambda(00)) = (\langle 0\rangle\lambda(00))\langle 0\rangle\lambda(00)
\end{array}$$

It is instructive to establish the relation between $\lambda\mathtt{w}$ and the λ_{db}-calculus. For this we need the following translation from $\lambda\mathtt{w}$-terms to de Bruijn terms. $E : \mathcal{T}_{\lambda\mathtt{w}} \to \mathcal{T}_{\lambda_{db}}$ is defined as follows:

$$\begin{array}{lcl}
E(n) & \overset{\text{def}}{=} & n \\
E(\lambda M) & \overset{\text{def}}{=} & \lambda E(M) \\
E(MN) & \overset{\text{def}}{=} & E(M)E(N) \\
E(\langle k\rangle M) & \overset{\text{def}}{=} & \mathcal{U}_0^k(E(M))
\end{array}$$

The relation between this new notion of metalevel substitution and the usual notion of metalevel substitution as defined in Def. 2.24 is as follows: $E(M\{i/N,j\}) = \mathcal{U}^j_{i+1}(E(M))\{\!\{i \leftarrow E(N)\}\!\}$. This may be used to show:

Lemma 2.55 1. Let $M, N \in \mathcal{T}_{\lambda\mathtt{w}}$. If $M \to_{\lambda\mathtt{w}} N$ then $E(M) \to_{\beta_{db}} E(N)$.

2. Let $M \in \mathcal{T}_{\lambda\mathtt{w}}$ and $P \in \mathcal{T}_{\lambda_{db}}$ such that $E(M) \to_{\beta_{db}} P$. Then there exists $N \in \mathcal{T}_{\lambda\mathtt{w}}$ such that $M \to_{\lambda\mathtt{w}} N$ and $E(N) = P$.

Definition 2.56 (Terms in $\lambda\mathtt{ws}$) The set of terms of the $\lambda\mathtt{ws}$-calculus, denoted $\mathcal{T}_{\lambda\mathtt{ws}}$, is defined as follows:

$$M \quad ::= \quad n \mid \lambda M \mid MM \mid \langle k\rangle M \mid M[i/M,j] \quad \text{where } n,i,j,k \in \mathbb{N}_0$$

$\bullet[\bullet/\bullet,\bullet]$ is called the *substitution operator*. Terms without occurrences of the substitution operator are called *pure terms*. *Positions* in terms are defined as usual; we use $M|_p$ to denote the subterm of M occurring at position p.

Remark 2.57 For the readers familiar with the λs-calculus [KR95] the translation into λs of $\langle k\rangle P$ is $\varphi_0^k(P)$, and of $P[i/Q,j]$ is $\varphi^j_{i+1}(P)\sigma^i Q$.

We shall use $\langle\vec{k}\rangle$ to denote the sequence of explicit updating operators $\langle k_1\rangle \ldots \langle k_n\rangle$. In particular, if we want to stress the fact that $n > 0$ we write $\langle\vec{k}^+\rangle$. Also, we shall use $\sum\vec{k}$ to abbreviate $\sum_{i=1}^n k_i$ where $\langle\vec{k}\rangle$ is $\langle k_1\rangle \ldots \langle k_n\rangle$. The following characterization of the $\lambda\mathtt{ws}$-terms may be proved by induction on M.

Definition 2.58 (The λws-calculus) The λws-calculus is defined by the following rewrite rules:

$$\mathtt{B}\left\{\begin{array}{lll} (\lambda M)N & \rightarrow_{b1} & M[0/N,0] \\ (\langle k\rangle\lambda M)N & \rightarrow_{b2} & M[0/N,k] \end{array}\right.$$

$$\mathtt{ws}\left\{\begin{array}{llll} (\lambda M)[i/N,j] & \rightarrow_{l} & \lambda(M[i+1/N,j]) & \\ (M_1M_2)[i/N,j] & \rightarrow_{a} & M_1[i/N,j]M_2[i/N,j] & \\ (\langle k\rangle M)[i/N,j] & \rightarrow_{e1} & \langle k+j-1\rangle M & i<k \\ (\langle k\rangle M)[i/N,j] & \rightarrow_{e2} & \langle k\rangle M[i-k/N,j] & i\geq k \\ n[i/N,j] & \rightarrow_{n1} & n & n<i \\ n[i/N,j] & \rightarrow_{n2} & \langle i\rangle N & n=i \\ n[i/N,j] & \rightarrow_{n3} & n+j-1 & n>i \\ M[k/N,l][i/P,j] & \rightarrow_{c1} & M[k/N[i-k/P,j],j+l-1] & k\leq i<k+l \\ M[k/N,l][i/P,j] & \rightarrow_{c2} & M[i-l+1/P,j][k/N[i-k/P,j],l] & k+l\leq i \\ \langle i\rangle\langle j\rangle M & \rightarrow_{m} & \langle i+j\rangle M & \end{array}\right.$$

The B-calculus is just rules $b1$ and $b2$. The substitution ws-calculus is the λws-calculus without the rules $b1$ and $b2$. The p-calculus is the ws-calculus without the m-rule. In [DG01] it is proved that p and ws are strongly normalizing and confluent. We use $NF_{\mathtt{p}}$ to denote the set of p-normal forms. Note that pure terms are not necessarily in substitution normal form (i.e. in ws-normal form) since they may contain m-redexes.

The λws-calculus is the first[7] lambda calculus with explicit substitutions to satisfy simulation of one-step β-rewrite reduction[8], confluence on open terms and preservation of strong normalization.

Lemma 2.59 (Simulation, Projection, Confluence)

- (Simulation). Let M, N be terms in $\mathcal{T}_{\lambda_{\mathtt{v}}}$. If $M \rightarrow_{\lambda_{\mathtt{v}}} N$ then $M \overset{+}{\rightarrow}_{\lambda_{\mathtt{ws}}} N$.
- (Projection). Let $M, N \in \mathcal{T}_{\lambda_{\mathtt{ws}}}$. If $M \rightarrow_{\lambda_{\mathtt{ws}}} N$ then $\mathtt{ws}(M) \twoheadrightarrow_{\lambda_{\mathtt{v}}} \mathtt{ws}(N)$.
- (Confluence). The ARS $(\mathcal{T}_{\lambda_{\mathtt{ws}}}, \rightarrow_{\lambda_{\mathtt{ws}}})$ is confluent.

In fact, if the grammar of Def. 2.58 is enlarged with metavariables $X, Y, \ldots$ obtaining the set of *open* terms, say $\mathcal{T}^{o}_{\lambda_{\mathtt{ws}}}$:

$$M \quad ::= \quad n \mid X \mid \lambda M \mid MM \mid \langle k\rangle M \mid M[i/M,j] \quad \text{where } n,i,j,k \in \mathbb{N}_0$$

where X ranges over the set of metavariables, then the ARS $(\mathcal{T}^{o}_{\lambda_{\mathtt{ws}}}, \rightarrow_{\lambda_{\mathtt{ws}}^{o}})$ is confluent too, where $\rightarrow_{\lambda_{\mathtt{ws}}^{o}}$ is just λws-rewriting but over the terms in $\mathcal{T}^{o}_{\lambda_{\mathtt{ws}}}$. We say that λws is confluent on open terms.

Proposition 2.60 (PSN) λws enjoys preservation of strong normalization.

See [DG01] for a proof.

[7] Together, it seems [DG01], with the work by H.Goguen and J.Goubault-Larrecq [GGL00].

[8] However, see Lemma 2.59(1) and note that the terms M and N belong to $\mathcal{T}_{\lambda_{\mathtt{v}}}$ and not $\mathcal{T}_{\lambda_{db}}$.

Part I

Perpetuality in Calculi of Explicit Substitutions

Chapter 3

Perpetuality in λx

Part I of this thesis studies perpetuality in calculi of explicit substitution. Perpetuality studies the contraction of rewrite steps that preserve the possibility of infinite derivations. We shall mainly be concerned with *perpetual redexes*, in other words redexes whose contraction preserves the possibility of infinite derivations. A rewrite strategy which always contracts a perpetual redex is called a *perpetual rewrite strategy*. For example, the leftmost strategy is not perpetual for the λ-calculus. Indeed, consider the λ-term $M = (\lambda x.y)(\Delta\Delta)$ where $\Delta = \lambda x.xx$. Then M admits an infinite β-derivation, simply contract repeatedly the rightmost β-redex:

$$(\lambda x.y)(\Delta\Delta) \rightarrow_\beta (\lambda x.y)(\Delta\Delta) \rightarrow_\beta \ldots$$

Yet by contracting the leftmost β-redex in M we obtain the term $N = y$ which is a normal form. As the reader may observe, this is due to the erasing nature of the redex contracted since the subterm $\Delta\Delta$ no longer appears in N, it has been erased. The λ_I-calculus is obtained from the λ-calculus by restricting term formation: for every abstraction $\lambda x.M$ there is at least one free variable occurrence of x in M. In the λ_I-calculus all redexes are perpetual [CR36]. The interest in perpetual strategies is that if they normalize a term M then this term is strongly normalizing, that is, *all* derivations starting from M are finite. For example, we shall apply our studies on perpetuality for characterizing inductively the set of all the terms in λx that are strongly λx-normalizing. By inductively we mean describing the set as the smallest set closed under some set of rules, as when defining the set of λ-terms or the set of theorems of some logic system [Acz77]. We shall also apply it for proving strong normalization of a typed polymorphic lambda calculus with explicit substitutions.

Perpetual rewrite strategies for the λ-calculus were introduced in [BBKV76], a recent survey is [RSSX99]. A Term Rewrite System is called uniformly normalizable if all redexes are perpetual. J.W.Klop [Klo80][1]. showed that all non-erasing orthogonal *CRS*s (hence all non-erasing orthogonal Term Rewrite Systems) enjoy this property, thus generalizing Church's Theorem [CR36] stating that the λ_I-calculus is uniformly normalizable. In the direction of characterizing perpetual redexes, Z.Khasidashvili [Kha94, Kha01] proved that all non-erasing redexes are perpetual in orthogonal *ERS*s (hence all non-erasing redexes are perpetual in orthogonal Term Rewriting Systems), thus generalizing the Conservation Theorem [BBKV76] stating that β_I-redexes (i.e. β-redexes in λ_I) are perpetual in the λ-calculus. See [KOO01a] for a thorough treatment of perpetual rewrite strategies and a characterization of perpetual redexes in higher-order term rewrite systems.

All these results are formulated for orthogonal systems, but the calculus of explicit substitutions λx is not orthogonal. Indeed, the *Beta*-rewrite rule and the *App*-rewrite rules overlap forming a critical pair.

$$\begin{tikzcd}
 & ((\lambda x.M)N)\langle y := O\rangle \arrow[dl, "\mathit{Beta}"'] \arrow[dr, "\mathit{App}"] & \\
M\langle x := N\rangle\langle y := O\rangle & & (\lambda x.M)\langle y := O\rangle N\langle y := O\rangle
\end{tikzcd}$$

Thus the results on perpetuality already developed do not apply. So we study perpetuality in the λx-calculus by adapting a technique originally introduced for proving preservation of strong normalization of calculi of explicit substitutions [BBLRD96]. Applications of this study are a formulation of an inductive definition of the set of strongly λx-normalizable terms, two perpetual rewrite strategies for λx one of which is computable, and a

[1] However, see [Oos96].

proof of strong normalization of typed terms in a polymorphic lambda calculus with explicit substitutions, F_{es}. System F [Gir72, GLT89] is a set of typing rules for typing polymorphic terms, together with two rewrite rules: the β-rewrite rule, and a rewrite rule for instantiating type variables by arbitrary types. We shall augment the F-calculus with explicit substitutions obtaining F_{es} and, in addition to strong normalization, we also study subject reduction for the extended typing rules. In fact the original motivation from which the work reported in this chapter resulted was an attempt to apply J.Y.Girard's 'candidats de reductibilité' technique in order to prove strong normalization for F_{es}. Reducibility for proving properties of calculi of explicit substitution first appear in [Muñ97b] and [Rit93], however only for proving weak normalization. As regards strong normalization, apart from this work, there have been independent contributions [Rit99, DL01, Her00]. The results developed in this chapter have been published in [Bon99a, Bon01]. Further work on perpetuality for non-orthogonal systems, with applications to calculi of explicit substitutions, has been developed in [KOO01b].

Structure of the chapter

We shall begin by going over the fundamental results behind the proof technique devised by P.Lescanne et al in [BBLRD96] to prove preservation of strong normalization for λv. This technique is based on assigning a measure on derivations and applying a minimality argument, à la Nash-Williams' [NW63] proof of Kruskal's tree theorem, in order to arrive at a contradiction. We apply this technique in order to conclude that if M and N are $\lambda\mathtt{x}$-terms then:

1. if $M = P_1\langle x := P_2\rangle P_3 \dots P_n \in SN_{\lambda\mathtt{x}}$ then $N = (\lambda x.P_1)P_2 \dots P_n \in SN_{\lambda\mathtt{x}}$.
2. if $M \rightarrow_{\mathtt{x}\backslash Gc} N$ and $N \in SN_{\lambda\mathtt{x}}$ then $M \in SN_{\lambda\mathtt{x}}$.
3. if $M \rightarrow_{Gc} N$ and $P, N \in SN_{\lambda\mathtt{x}}$, where P is the garbage erased by the Gc-redex, then $M \in SN_{\lambda\mathtt{x}}$.

This shall suffice in order to provide an inductive characterization of the terms in $SN_{\lambda\mathtt{x}}$ and perpetual rewrite strategies for $\lambda\mathtt{x}$. These strategies are not refinements of the usual strategies for the λ-calculus. In other words, if $\mathcal{F}(\bullet)$ is some one-step rewrite strategy for the λ-calculus, hence $M \rightarrow_\beta \mathcal{F}(M)$ for all λ-terms M not in normal form, then in the explicit substitution formulation $\mathcal{F}_\mathtt{x}$ of $\mathcal{F}(\bullet)$ we shall not necessarily have $M \twoheadrightarrow_{\lambda\mathtt{x}} \mathcal{F}(M)$ where the steps in the this derivation are of the form $M \rightarrow_{\lambda\mathtt{x}} \mathcal{F}_\mathtt{x}(M) \rightarrow_{\lambda\mathtt{x}} \mathcal{F}_\mathtt{x}(\mathcal{F}_\mathtt{x}(M)) \rightarrow_{\lambda\mathtt{x}} \dots \mathcal{F}^n_\mathtt{x}(M)$.

A further application of the above mentioned result is the possibility of applying Girard's 'candidats de reductibilité' [Gir72] proof technique in order to prove strong normalization of a polymorphic lambda calculus with explicit substitutions. For this we formulate F_{es}, a polymorphic λ-calculus of explicit substitutions which incorporates two flavours of substitutions: term and type substitutions. Subject reduction is proved for F_{es}. The latter result was shown to fail for other formulations of higher-order lambda calculi with explicit substitutions based on $\lambda\mathtt{x}$ [Blo97]. This situation was later reverted in [Blo99, Blo01], so it may be seen as an independent solution to this problem. Additional related work is that by C.Muñoz [Muñ97b] who defines a typed lambda calculus with explicit substitutions (based on a left-linear variant of $\lambda\sigma$) and dependent types. He proves subject reduction for this calculus by introducing type annotations in the $\bullet \cdot \bullet$ constructor of $\lambda\sigma$.

Finally, we prove strong normalization for all polymorphically typable terms.

3.1 The Perpetuality Proposition

We shall begin our study on infinite derivations in $\lambda\mathtt{x}$. For this we recall the closure tracing technique introduced in [BBLRD96] for proving preservation of strong normalization for the calculus λv. In [BBLRD96] infinite derivations starting from pure terms are studied since it is PSN which is of interest, here we study infinite derivations starting from *any* term. On the way we shall encounter some simplifications on this technique as a tool for proving PSN.

An overview of the $\lambda\mathtt{x}$-calculus is given in Chapter 2. The reader already familiar with this calculus of explicit substitutions may read on, otherwise we recommend taking a glimpse at Section 2.3.1 before continuing. Nonetheless, we recall some of the properties of $\lambda\mathtt{x}$ which we shall use in this section.

Definition 2.38. A substitution $\langle x := N\rangle$ is called *void in* $M\langle x := N\rangle$ if $x \notin FV(\mathtt{x}(M))$; M is said to be the *target* of the substitution and N its *body*. *Void reduction*, denoted $\xrightarrow{v}_{\lambda\mathtt{x}}$, is $\lambda\mathtt{x}$-reduction inside the body of a void substitution.

A λx-*context*, or simply a context, is a λx-term with a 'hole'. More precisely, if we use the distinguished variable $\Box$ to denote this hole then a context is just a term with exactly one occurrence of $\Box$. An example is $\lambda x.\Box\langle y := z\rangle$.

Let us recall how β-rewrite steps may be executed via the λx-calculus:

Lemma 2.39 (Projection). Let M, N be terms in $\mathcal{T}_{\lambda\mathtt{x}}$. Then

1. If $M \rightarrow_{\lambda\mathtt{x}} N$ then $\mathtt{x}(M) \twoheadrightarrow_\beta \mathtt{x}(N)$.
2. If $M \rightarrow_{Beta} N$ is not a void reduction then $\mathtt{x}(M) \xrightarrow{+}_\beta \mathtt{x}(N)$.

Consider some infinite λx-derivation $\phi : M_0 \rightarrow_{\lambda\mathtt{x}} M_1 \rightarrow_{\lambda\mathtt{x}} \ldots$. Since x is strongly normalizing an infinite number of *Beta*-rewrite steps must take place in ϕ. If all these *Beta*-rewrite steps were not void then we would obtain an infinite β-derivation from the pure term $\mathtt{x}(M_0)$, using the Projection Lemma. Therefore, if we already know that $\mathtt{x}(M_0)$ is strongly β-normalizing, then we may conclude that at some point on, in ϕ, all *Beta*-rewrite steps are void. In fact, it may be shown that at some point on in ϕ *all* λx-rewrite steps are void. This result is the starting point of P.Lescanne et al's method for proving PSN. We formalize it below (see [Blo97, Proposition 4.15]).

Lemma 3.1 If $M_0, M_1, \ldots \in \mathcal{T}_{\lambda\mathtt{x}}$ such that $\mathtt{x}(M_0) \in SN_\beta$ and $M_0 \rightarrow_{\lambda\mathtt{x}} M_1 \rightarrow_{\lambda\mathtt{x}} \ldots$ is an infinite λx-derivation then there is a $k \in \mathbb{N}$ such that for all $i \geq k$, $M_i \xrightarrow{v}_{\lambda\mathtt{x}} M_{i+1}$.

Proof. Since x is strongly normalizing we may assume the infinite λx-derivation has the form $M_0 \twoheadrightarrow_{\mathtt{x}} M_1 \rightarrow_{Beta} M_2 \twoheadrightarrow_{\mathtt{x}} M_3 \ldots$. Now by Lemma 2.39(1) we have $\mathtt{x}(M_0) \twoheadrightarrow_\beta \mathtt{x}(M_2) \twoheadrightarrow_\beta \mathtt{x}(M_4) \twoheadrightarrow_\beta \mathtt{x}(M_6) \ldots$, where by Lemma 2.39(2) we have $\mathtt{x}(M_{2n}) \xrightarrow{+}_\beta \mathtt{x}(M_{2n+2})$ if $M_{2n+1} \rightarrow_{Beta} M_{2n+2}$ is not void.

Now since $\mathtt{x}(M_0) \in SN_\beta$ there is a $j \in \mathbb{N}$ such that for all $i \geq j$ we have $M_{2i+1} \xrightarrow{v}_{Beta} M_{2i+2}$. We must now prove that from some point onwards not only the *Beta*-reductions are void but also the x-reductions. This is done by defining an interpretation h on $\mathcal{T}_{\lambda\mathtt{x}}$:

$$\begin{array}{lcll} h(x) & \overset{\text{def}}{=} & 1 & \\ h(MN) & \overset{\text{def}}{=} & h(M) + h(N) + 1 & \\ h(\lambda z.N) & \overset{\text{def}}{=} & h(N) + 1 & \\ h(M\langle z := N\rangle) & \overset{\text{def}}{=} & \begin{cases} (1 + h(N)) \times h(M) & \text{if } z \in FV(\mathtt{x}(M)) \\ 2 \times h(M) & \text{if } z \notin FV(\mathtt{x}(M)) \end{cases} \end{array}$$

In the last clause of the definition of h note that if $z \notin FV(\mathtt{x}(M))$ then $h(N)$ is neglected, hence any reduction steps inside N shall not alter $h(M\langle z := N\rangle)$. In fact one may verify that:

- if $M \xrightarrow{v}_{\lambda\mathtt{x}} N$ then $h(M) = h(N)$, and
- if $M \rightarrow_{\mathtt{x}} N$ is not void then $h(M) > h(N)$.

Thus there must be a $k > j$ such that for all $i \geq k$ we have that not only $M_{2i+1} \xrightarrow{v}_{Beta} M_{2i+2}$ but also $M_{2i} \xrightarrow{v}_{\mathtt{x}} M_{2i+1}$. ▪

Note that in contrast to [BBLRD96] there has been no need to define internal/external positions (since they have been captured by the definition of h) and to prove commutation of internal/external reductions in order to prove Lemma 3.1 [BBLRD96, Lemma 13]. This is clearly an advantage since commutation results are long, tedious and technical. It thus suggests a simplified variant of the proof technique introduced in [BBLRD96] for proving PSN.

So now we know that if $\phi : M_0 \rightarrow_{\lambda\mathtt{x}} M_1 \rightarrow_{\lambda\mathtt{x}} \ldots$ is an infinite λx-derivation and $\mathtt{x}(M_0)$ is strongly β-normalizing, then at some point on in ϕ all λx-rewrite steps are void. We shall see that from ϕ we may learn of the existence of an infinite λx-derivation in which from some point on all λx-rewrite steps take place within the 'same' closure. This requires the notion of a skeleton of a term which first appeared in [KR95]. The corresponding concept in terms of positions occurs under the name *frontier* in [BBLRD96].

Definition 3.2 (Skeleton) The *skeleton* of a term M in $\mathcal{T}_{\lambda x}$, denoted $\mathcal{SK}(M)$, is defined inductively as follows

$$\begin{array}{lcl} \mathcal{SK}(x) & \overset{\text{def}}{=} & x \\ \mathcal{SK}(N_1N_2) & \overset{\text{def}}{=} & \mathcal{SK}(N_1)\mathcal{SK}(N_2) \\ \mathcal{SK}(\lambda y.N) & \overset{\text{def}}{=} & \lambda y.\mathcal{SK}(N) \\ \mathcal{SK}(N_1\langle z := N_2\rangle) & \overset{\text{def}}{=} & \mathcal{SK}(N_1)\langle z := \bullet\rangle \end{array}$$

where the '•' symbol may be seen as a place-holder (much the same as the '□' symbol in a context). Intuitively, the skeleton of a term is the part where only reductions which do not take place in bodies of substitutions are possible.

Remark 3.3 Note that if $M \overset{v}{\rightarrow}_{\lambda x} N$ then $\mathcal{SK}(M) = \mathcal{SK}(N)$.

Lemma 3.4 If $M_0, M_1, \ldots \in \mathcal{T}_{\lambda x}$ such that $\mathrm{x}(M_0) \in SN_\beta$ and $M_0 \rightarrow_{\lambda x} M_1 \rightarrow_{\lambda x} \ldots$ is an infinite λx-derivation then there exists $k \in \mathbb{N}$, a variable y, a context C and λx terms $P, Q_k, Q_{k+1}, Q_{k+2}, \ldots$ such that

$$\begin{array}{rcl} M_0 \rightarrow_{\lambda x} M_1 \rightarrow_{\lambda x} \cdots \rightarrow_{\lambda x} M_k & = & C[P\langle y := Q_k\rangle] \\ & \overset{v}{\rightarrow}_{\lambda x} & C[P\langle y := Q_{k+1}\rangle] \\ & \overset{v}{\rightarrow}_{\lambda x} & C[P\langle y := Q_{k+2}\rangle] \end{array}$$

where for $i \geq k$ the $i+1$-rewrite step takes place in Q_i.

Proof. Consider an infinite λx-derivation $\phi : M = M_1 \rightarrow_{\lambda x} M_2 \rightarrow_{\lambda x} M_3 \ldots$ starting from M. Then since $\mathrm{x}(M) \in SN_\beta$ we may apply Lemma 3.1 and obtain a $k \in \mathbb{N}$ such that for all $i \geq k$ we have $M_i \overset{v}{\rightarrow}_{\lambda x} M_{i+1}$. And hence by Remark 3.3 we have $\mathcal{SK}(M_k) = \mathcal{SK}(M_i)$ for all $i \geq k$.

Now as there are only a finite number of closures in $\mathcal{SK}(M_k)$ and, since the reductions within these closures are independent in the sense that they occur in parallel positions, by König's Lemma an infinite derivation must take place within the same closure in $\mathcal{SK}(M_k)$. Thus $M_k = C[P\langle y := Q_k\rangle]$ for some context C and

$$\begin{array}{rcl} M_1 \rightarrow_{\lambda x} M_2 \rightarrow_{\lambda x} \cdots \rightarrow_{\lambda x} M_k & = & C[P\langle y := Q_k\rangle] \\ & \overset{v}{\rightarrow}_{\lambda x} & C[P\langle y := Q_{k+1}\rangle] \\ & \overset{v}{\rightarrow}_{\lambda x} & C[P\langle y := Q_{k+2}\rangle] \\ & & \cdot \end{array}$$

is an infinite λx-derivation starting from M. ▪

Finally, Lemma 3.4 together with the fact, proved below, that closures may be traced back shall provide all the tools for proving our main perpetuality result.

Lemma 3.5 (One-step closure tracing) Let $M, N \in \mathcal{T}_{\lambda x}$ with $M \rightarrow_{\lambda x} N = C[P\langle y := Q\rangle]$. Then,

1. either $M = C'[P'\langle y := Q\rangle]$ for some context C',
2. or, $M = C'[P'\langle y := Q'\rangle]$ for some context C' and $Q' \rightarrow_{\lambda x} Q$,
3. or, $M = C[(\lambda y.P)Q]$.

Proof. We use induction on M and consider the following two cases:

- The reduction takes place at the root. First note that if $P\langle y := Q\rangle$ occurs in a subterm of N which is also a subterm of M then for some context C' and $P' = P$ the first item holds trivially. Also note that this includes the cases where the rule applied at the root is *Var* or *Gc*. Otherwise we must have one of the following:
  - $M = (\lambda y.P)Q \rightarrow_{Beta} P\langle y := Q\rangle = N$ with $C = \Box$. Then the third item holds.

- $M = (M_1M_2)\langle y := Q\rangle \to_{App} M_1\langle y := Q\rangle M_2\langle y := Q\rangle = N$ where P is M_1 or M_2. Then we take $P' = M_1M_2$ and $C' = \Box$ and the first item holds.
- $M = (\lambda z.P)\langle y := Q\rangle \to_{Lam} \lambda z.P\langle y := Q\rangle = N$. Then we take $C' = \Box$, $P' = \lambda z.P$ and item one holds.

- The reduction is internal.
  - $M = x$. The result holds vacuously.
  - $M = M_1M_2$ with either $M_1 \to_{\lambda x} M_1'$, or $M_2 \to_{\lambda x} M_2'$. We consider the first case, the other being similar. Thus $N = M_1'M_2$. We have two further cases to consider.
    - * The subterm $P\langle y := Q\rangle$ occurs in M_1'. We then use the induction hypothesis.
    - * The subterm $P\langle y := Q\rangle$ occurs in M_2. Then the first item trivially holds.
  - $M = \lambda z.M_1$ with $M_1 \to_{\lambda x} M_1'$. We use the induction hypothesis.
  - $M = M_1\langle z := M_2\rangle$ with
    - * either, $M_1 \to_{\lambda x} M_1'$ and $N = M_1'\langle z := M_2\rangle$. Then if $P\langle y := Q\rangle$ occurs in M_1' we use the induction hypothesis. If it occurs in M_2 then the first item trivially holds. Finally, if $N = P\langle y := Q\rangle$ then we take $C' = \Box$, $P' = M_1$ and the first item holds.
    - * or, $M_2 \to_{\lambda x} M_2'$ and $N = M_1\langle z := M_2'\rangle$. Then if $P\langle y := Q\rangle$ occurs in M_1 then the first item trivially holds. If it occurs in M_2' then we use the induction hypothesis. Finally, if $N = P\langle y := Q\rangle$ then we take $C' = \Box$, $P' = M_1$ and $Q' = M_2$ and the second item holds.

■

This result extends naturally to many-step derivations. It may be proved by induction on the number of rewrite steps.

Lemma 3.6 (Closure tracing) Let $M_1, \ldots, M_n \in \mathcal{T}_{\lambda x}$ such that $M_i \to_{\lambda x} M_{i+1}$ for $i \in 1,..,n-1$, and $M_n = C[P\langle y := Q\rangle]$. Then,

1. either there is an $i \in 1,..,n-1$ such that $M_i = C'[(\lambda y.P')Q']$ for some context C' and $Q' \twoheadrightarrow_{\lambda x} Q$,
2. or, $M_1 = C'[P'\langle y := Q'\rangle]$ for some context C' and $Q' \twoheadrightarrow_{\lambda x} Q$.

Definition 3.7 (Derivation ordering) Let ϕ and ψ be two infinite derivations starting from a term M_1. Then the derivation $\phi : M_1 \to_{\lambda x, p_1} M_2 \to_{\lambda x, p_2} M_3 \ldots M_n \to_{\lambda x, p_n} M_{n+1} \ldots$ is said to be *smaller* than the derivation $\psi : M_1 \to_{\lambda x, q_1} M_2 \to_{\lambda x, q_2} M_3 \ldots M_n \to_{\lambda x, q_n} M_{n+1}' \cdots$ if $p_i = q_i$ for $i < n$ and q_n is a proper prefix of p_n.

Remark 3.8 Suppose $M = C[P\langle y := Q\rangle] \to_x N$. Then there are two possibilities:

1. either, the reduction step takes place in Q, i.e. $N = C[P\langle y := Q'\rangle]$ and $Q \to_x Q'$,
2. or, it does not take place inside Q.

In the second case we then have that the substitution body Q also occurs in N or else it was erased as a result of applying the *Gc*-rule (Q is subterm of garbage erased by *Gc*). This observation shall be made more precise in the following lemma.

Lemma 3.9 Let $M = C[P\langle y := Q\rangle] \to_x N$ such that the x-reduction step does not take place inside Q, and let S be the term (substitution body) eliminated by the *Gc*-rule if it was applied, then

- either, Q occurs in N
- or, $Q \subseteq S$.

Proof. By induction on the context C.

- $C = \Box$. Then $M = P\langle y := Q\rangle \to_x N$. We have two further cases to consider:

  - the reduction takes place at the root. If the rule applied was *App*, *Lam* or *Var* then the first item holds. If the rule applied was Gc then $S = Q$ and the second item holds.
  - the reduction is internal. Then we must have $P \to_x P'$ and the first item trivially holds.

- $C = C'O$ (the case $C = OC'$ is analogous). Then the reduction step $M = C'[P\langle y := Q\rangle]O \to_x N$ must be internal and $N = N_1N_2$. We have two further cases to consider:
  - $C'[P\langle y := Q\rangle] \to_x N_1$ and $N_2 = O$. Then the result follows from the induction hypothesis.
  - $N_1 = C'[P\langle y := Q\rangle]$ and $O \to_x N_2$. Then the first item trivially holds.
- $C = \lambda v.C'$. Then the reduction $M = \lambda y.C'[P\langle y := Q\rangle] \to_x N$ must be internal and the result follows by the induction hypothesis.
- $C = C'\langle x := O\rangle$. Thus $C'[P\langle y := Q\rangle]\langle x := O\rangle \to_x N$. We have two further cases to consider:
  - the reduction takes place at the root. If the rule applied is Gc then $S = O$ and $N = C'[P\langle y := Q\rangle]$ and the first item trivially holds. Suppose then that the rule applied is *App*, *Lam* or *Var*. Then $C' \neq \Box$ and one of the following must hold: $C' = C''R$, $C' = RC''$ or $C' = \lambda y.C''$ for some context C''. In all cases the first item holds.
  - the reduction is internal. Then we have two cases:
    * either, $C'[P\langle y := Q\rangle] \to_x N_1$ and $N = N_1\langle x := O\rangle$ in which case the result follows from the induction hypothesis.
    * or, $O \to_x N_1$ and $N = C'[P\langle y := Q\rangle]\langle x := N_1\rangle$ and the first item holds.
- $C = O\langle v := C'\rangle$. Then we have $M = O\langle v := C'[P\langle y := Q\rangle]\rangle \to_x N$. We consider two cases:
  - the reduction takes place at the root. If the rule applied was *App*, *Lam* or *Var* then the first item holds. If the rule applied was Gc then $S = C'[P\langle y := Q\rangle]$ and the second item holds.
  - the reduction is internal. Then
    * either, $O \to_x N_1$ and $N = N_1\langle v := C'[P\langle y := Q\rangle]\rangle$ and the first item holds trivially.
    * or, $C'[P\langle y := Q\rangle] \to_x N_1$ and $N = O\langle v := N_1\rangle$ and the result follows from the induction hypothesis.

∎

Before proceeding to the main proposition of this section we would like to discuss the relation with the work in [BBLRD96] for proving PSN.

- *What P. Lescanne et al do for PSN.* They consider a minimal infinite λx-derivation starting from a *pure* term. Thus they can *always* trace back the closure (guaranteed to exist by Lemma 3.4) to its (unique) point of creation (here the *Beta*-rule) and obtain a *smaller* derivation than the original one, hence contradicting minimality.
- *What we do for perpetuality.* Suppose $M \to_x N$ and $N \in SN_{\lambda x}$. Given a minimal infinite λx-derivation starting from M we trace back the closure (guaranteed to exist by Lemma 3.4). But now there are *two possible* situations as dictated by Lemma 3.6:
  - either, the closure was created by a *Beta*-rule. We then argue as above and contradict minimality.
  - or, (an ancestor of) the body of the closure, say Q, *belongs to the original term* M. But then since $M \to_x N$ we may reason as follows:
    * either, the substitution body Q *also occurs* in N, in which case we arrive at a contradiction since $N \in SN_{\lambda x}$ and $Q \notin SN_{\lambda x}$,
    * or, Q is a subterm of garbage, i.e. $M \to_{Gc} N$ and $Q \subseteq P$ where P is the substitution body eliminated by Gc, then by requiring that all eliminated garbage be strongly λx-normalizing terms we arrive at a contradiction,

- or, the reduction step $M \to_{\mathtt{x}} N$ takes place inside the substitution body Q, i.e. $Q \to_{\mathtt{x}} Q'$ for some $Q' \subset N$. Then we may apply inductively the same reasoning in order to obtain a contradiction. Note that since there are a finite number of closures in M this case may not repeat itself an infinite number of times.

These ideas may be formalized as follows.

Lemma 3.10 Let $M, N \in \mathcal{T}_{\lambda\mathtt{x}}$.

1. Suppose $M \to_{\mathtt{x}\setminus Gc} N$ and $N \in SN_{\lambda\mathtt{x}}$ then $M \in SN_{\lambda\mathtt{x}}$.
2. Suppose $M = C[P_1\langle z := P_2\rangle] \to_{Gc} C[P_1] = N$ where the Gc-rewrite step takes place at the position of the hole, and $P_2, N \in SN_{\lambda\mathtt{x}}$ then $M \in SN_{\lambda\mathtt{x}}$.

Proof. Both items are proved simultaneously by (course of values) induction on the number of occurrences n of the closure operator in M. Note that since $M \to_{\mathtt{x}} N$ by hypothesis there must be at least one occurrence of the closure operator in M.

- $n = 1$. Suppose that $M \notin SN_{\lambda\mathtt{x}}$ and let $\phi : M = M_1 \to_{\lambda\mathtt{x}} M_2 \to_{\lambda\mathtt{x}} M_3 \ldots$ be a *minimal infinite derivation* starting from M. Now since $N \in SN_{\lambda\mathtt{x}}$ then $\mathtt{x}(N) \in SN_\beta$ and hence $\mathtt{x}(M) \in SN_\beta$ (since $\mathtt{x}(M) = \mathtt{x}(N)$). Then by Lemma 3.4 we may construct the following infinite $\lambda\mathtt{x}$-derivation,

$$\begin{array}{rcl} \phi' : M_1 \to_{\lambda\mathtt{x}} M_2 \to_{\lambda\mathtt{x}} \ldots \to_{\lambda\mathtt{x}} M_k & = & C[P\langle y := Q_k\rangle]_p \\ & \overset{v}{\to}_{\lambda\mathtt{x}} & C[P\langle y := Q_{k+1}\rangle]_p \\ & \overset{v}{\to}_{\lambda\mathtt{x}} & C[P\langle y := Q_{k+2}\rangle]_p \\ & & \cdot \\ & & \cdot \end{array}$$

for some $k \in \mathbb{N}$, position p, variable y, context C and $\lambda\mathtt{x}$-terms $P, Q_k, Q_{k+1}, Q_{k+2}, \ldots$. Note that the sequence $(Q_k, Q_{k+1}, Q_{k+2}, \ldots)$ is an infinite $\lambda\mathtt{x}$-derivation.

Now by the closure tracing Lemma 3.6 there are two possibilities for the origin of the closure $\langle y := Q_k\rangle$:

- either, it was created sometime before by an application of the *Beta*-rule, i.e. there exists $j < k$, a context C' and a position p_j such that:

$$M_j = C'[(\lambda y.P')Q]_{p_j} \to_{Beta,p_j} C'[P'\langle y := Q\rangle]_{p_j} = M_{j+1}$$

with $Q \twoheadrightarrow_{\lambda\mathtt{x}} Q_k$. But then we may construct the infinite $\lambda\mathtt{x}$-derivation

$$\begin{array}{rcl} \phi'' : M_1 \to_{\lambda\mathtt{x}} M_2 \ldots \to_{\lambda\mathtt{x}} M_j & = & C'[(\lambda y.P')Q]_{p_j} \\ & \twoheadrightarrow_{\lambda\mathtt{x}} & C'[(\lambda y.P')Q_k]_{p_j} \\ & \overset{v}{\to}_{\lambda\mathtt{x}} & C'[(\lambda y.P')Q_{k+1}]_{p_j} \\ & \overset{v}{\to}_{\lambda\mathtt{x}} & C'[(\lambda y.P')Q_{k+2}]_{p_j} \\ & & \cdot \end{array}$$

Note that ϕ'' is smaller than ϕ since at step j we reduce a proper subterm of $(\lambda y.P')Q$ and obtain nonetheless an infinite $\lambda\mathtt{x}$-derivation. Thus we contradict the minimality of ϕ.

- or, (an ancestor of) the body of the closure, *belongs to the original term* M, i.e. $M = C'[P'\langle y := Q\rangle]$ for some context C' and terms P' and Q with $Q \twoheadrightarrow_{\lambda\mathtt{x}} Q_k$. Here we have two further cases to consider:

1. either the reduction step $M = C'[P'\langle y := Q\rangle] \to_{\mathtt{x}} N$ takes place in Q, i.e. $N = C'[P'\langle y := Q'\rangle]$ and $Q \to_{\mathtt{x}} Q'$,
2. or it does not take place inside Q.

But the first case is not possible since $n = 1$ and therefore there are no occurrences of the closure operator in Q. So the reduction step $M = C'[P'\langle y := Q\rangle] \to_{\mathtt{x}} N$ does not take place inside Q, and therefore by Lemma 3.9, we have that either Q occurs in N (which is in $SN_{\lambda\mathtt{x}}$) or is a subterm of garbage (i.e. $Q \subseteq P_2$) and hence also is in $SN_{\lambda\mathtt{x}}$ as the additional requirement on application of Gc-rule states. Thus we arrive at a contradiction (closure tracing has determined that there is an infinite $\lambda\mathtt{x}$-derivation starting from Q and, on the other hand, we have $Q \in SN_{\lambda\mathtt{x}}$).

- $n > 1$. We start off as in the previous case; the only difference is when the closure tracing lemma determines that (an ancestor of) the body of the closure, *belongs to the original term M*, i.e. $M = C'[P'\langle y := Q\rangle]$ for some context C' and terms P' and Q with $Q \twoheadrightarrow_{\lambda x} Q_k$. Here it is possible that the reduction step $M = C'[P'\langle y := Q\rangle] \rightarrow_x N$ takes place in Q, i.e. $N = C'[P'\langle y := Q'\rangle]$ and $Q \rightarrow_x Q'$. In this case, since $N \in SN_{\lambda x}$ we have $Q' \in SN_{\lambda x}$ and we may apply the induction hypothesis w.r.t. Q and obtain that $Q \in SN_{\lambda x}$. Hence we obtain a contradiction.

Proposition 3.11 (Perpetuality Proposition) Let $M, N \in \mathcal{T}_{\lambda x}$. Then

1. if $M = P_1\langle x := P_2\rangle P_3 \dots P_n \in SN_{\lambda x}$ then $N = (\lambda x.P_1)P_2 \dots P_n \in SN_{\lambda x}$.
2. if $M \rightarrow_{x\setminus Gc} N$ and $N \in SN_{\lambda x}$ then $M \in SN_{\lambda x}$.
3. if $M = D[P_1\langle z := P_2\rangle]_q \rightarrow_{Gc,q} D[P_1]_q = N$ for some context D and $P_2, N \in SN_{\lambda x}$ then $M \in SN_{\lambda x}$.

Proof. The last two items have already been proved (Lemma 3.10). For the first item we consider the term $P_1\langle x := P_2\rangle P_3 \dots P_n \in SN_{\lambda x}$. Then $P_1, \dots, P_n \in SN_{\lambda x}$. Therefore, any infinite derivation starting from the term $(\lambda x.P_1)P_2 \dots P_n$ must have the form,

$$\begin{array}{lll} (\lambda x.P_1)P_2 \dots P_n & \twoheadrightarrow_{\lambda x} & (\lambda x.P_1')P_2' \dots P_n' \\ & \rightarrow_{Beta} & P_1'\langle x := P_2'\rangle P_3' \dots P_n' \\ & \rightarrow_{\lambda x} & \dots \end{array}$$

where $P_i \twoheadrightarrow_{\lambda x} P_i'$ for $i \in 1..n$. Then there is an infinite derivation,

$$\begin{array}{lll} P_1\langle x := P_2\rangle P_3 \dots P_n & \twoheadrightarrow_{\lambda x} & P_1'\langle x := P_2'\rangle P_3' \dots P_n' \\ & \rightarrow_{\lambda x} & \dots \end{array}$$

contradicting the hypothesis. ∎

Remark 3.12 Note that the above technique may be applied to the explicit substitution formulation of the substitution lemma, more precisely, we have that if $y \notin FV(P)$ and $C[M\langle x := P\rangle\langle y := N\langle x := P\rangle\rangle]_p$ is strongly λx-normalizing then $C[M\langle y := N\rangle\langle x := P\rangle]_p$ is also strongly λx-normalizing. Obviously this does not mean that we may add the inferred rule (i.e. $M\langle y := N\rangle\langle x := P\rangle \rightarrow_{Co} M\langle x := P\rangle\langle y := N\langle x := P\rangle\rangle$) to λx without (trivially) losing strong normalization.

This remark shall be used in the proof of strong normalization for a polymorphic lambda calculus with explicit substitutions (Lemma 3.63).

3.2 Some Applications of the Perpetuality Analysis

This section considers some applications of the perpetuality analysis elaborated in Section 3.1: an inductive characterization of $SN_{\lambda x}$, two perpetual rewrite strategies for λx, and strong normalization by reducibility for a polymorphic lambda calculus with explicit substitutions F_{es}. This last issue shall be considered in detail in Section 3.3. The technique presented in this section is based on similar results obtained for the λ-calculus with the β-rewrite rule [RS95].

3.2.1 Characterizing Terminating Terms in λx

We give an inductive characterization of the terms in $SN_{\lambda x}$ by combining the technique presented by F. van Raamsdonk and P. Severi in [RS95] and the Perpetuality Proposition (Proposition 3.11). This characterization shall be used in the proof of perpetuality of some rewrite strategies studied in Section 3.2.2.

Lemma 3.13 Every $M \in \mathcal{T}_{\lambda x}$ is of one of the following forms:

1. $xP_1 \dots P_n$
2. $\lambda x.P$

3. $(\lambda x.P)P_1 \dots P_n$

4. $P_0\langle x_1 := Q_1\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n$ where $m \geq 1$ and P_0 is not a closure (i.e. it is either a variable, an abstraction or an application).

Proof. By induction on the term M. ▪

A word on notation before continuing: if $M, N_1, \dots, N_m \in \mathcal{T}_{\lambda\mathbf{x}}$ and $x_1, \dots, x_m \in \mathcal{V}$ for $m \geq 0$ then we use M^* to denote the term $M\langle x_1 := N_1\rangle \dots \langle x_m := N_m\rangle$.

Definition 3.14 Let $\mathcal{SN} \subseteq \mathcal{T}_{\lambda\mathbf{x}}$ be the smallest set closed under the inference schemes:

$$\frac{P_1, \dots, P_n \in \mathcal{SN} \quad x \in \mathcal{V}}{xP_1 \dots P_n \in \mathcal{SN}}\,R1 \qquad \frac{P \in \mathcal{SN}}{\lambda x.P \in \mathcal{SN}}\,R2$$

$$\frac{M\langle x := N\rangle P_1 \dots P_n \in \mathcal{SN}}{(\lambda x.M)\ NP_1 \dots P_n \in \mathcal{SN}}\,R3$$

$$\frac{N^* P_1 \dots P_n \in \mathcal{SN} \quad x \in \Sigma_v}{x\langle x := N\rangle^* P_1 \dots P_n \in \mathcal{SN}}\,R4 \qquad \frac{y^* P_1 \dots P_n \in \mathcal{SN} \quad N \in \mathcal{SN} \quad x \in \mathcal{V}}{y\langle x := N\rangle^* P_1 \dots P_n \in \mathcal{SN}}\,R5$$

$$\frac{(M_1\langle x := N\rangle M_2\langle x := N\rangle)^* P_1 \dots P_n \in \mathcal{SN}}{(M_1 M_2)\langle x := N\rangle^* P_1 \dots P_n \in \mathcal{SN}}\,R6 \qquad \frac{(\lambda y.M\langle x := N\rangle)^* P_1 \dots P_n \in \mathcal{SN}}{(\lambda y.M)\langle x := N\rangle^* P_1 \dots P_n \in \mathcal{SN}}\,R7$$

where inference scheme $(R5)$ is subject to the restriction $x \neq y$.

Each inference scheme is quantified over n and m (if m also occurs in it). Thus for example inference scheme $(R5)$ should be read as: for all $m \geq 0$ and $n \geq 0$, if $N \in \mathcal{SN}$ and $y\langle x_1 := N_1\rangle \dots \langle x_m := N_m\rangle P_1 \dots P_n \in \mathcal{SN}$ and $y \neq x$ then $y\langle x := N\rangle\langle x_1 := N_1\rangle \dots \langle x_m := N_m\rangle P_1 \dots P_n \in \mathcal{SN}$.

Lemma 3.15 $SN_{\lambda\mathbf{x}} = \mathcal{SN}$.

Proof.

- $SN_{\lambda\mathbf{x}} \subseteq \mathcal{SN}$. Let $M \in SN_{\lambda\mathbf{x}}$. We prove by induction on $(maxred_{\lambda\mathbf{x}}(M), |M|)$, using the usual lexicographic ordering, that $M \in \mathcal{SN}$. By Lemma 3.13 we have the following cases to consider:
  - $M = xP_1 \dots P_n$. Then since M is strongly λx-normalizing, $P_1, \dots, P_n$ are strongly λx-normalizing. By the induction hypothesis we have that $P_1, \dots, P_n \in \mathcal{SN}$. Thus by inference scheme $(R1)$ we obtain $xP_1 \dots P_n \in \mathcal{SN}$ (note that this case includes the base case, i.e. all variables are in $\mathcal{SN}$).
  - $M = \lambda x.P$. As above P is strongly λx-normalizing, and therefore by the induction hypothesis, $P \in \mathcal{SN}$. Using inference scheme $(R2)$ we conclude that $\lambda x.P \in \mathcal{SN}$.
  - $M = (\lambda x.P_0)P_1 \dots P_n$. Then since $M \rightarrow_{Beta} P_0\langle x := P_1\rangle P_2 \dots P_n$ we have that $P_0\langle x := P_1\rangle P_2 .. P_n \in SN_{\lambda\mathbf{x}}$. By the induction hypothesis $P_0\langle x := P_1\rangle P_2 \dots P_n \in \mathcal{SN}$, and thus by inference scheme $(R3)$ we conclude that $M \in \mathcal{SN}$.
  - $M = P_0\langle x_1 := Q_1\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n$ where $m \geq 1$ and P_0 is not a closure (i.e. not of the form $P'\langle z := Q'\rangle$). Thus we have the following cases to consider:
    - $P_0 = x_1$. Then $Q_1\langle x_2 := Q_2\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n \in SN_{\lambda\mathbf{x}}$, and by the induction hypothesis this term belongs to $\mathcal{SN}$. Then by inference scheme $(R4)$ we obtain that $M \in \mathcal{SN}$.
    - $P_0 = y \neq x_1$. Then $y\langle x_2 := Q_2\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n \in SN_{\lambda\mathbf{x}}$ and by induction hypothesis it is also an element of $\mathcal{SN}$. Also, since $M \in SN_{\lambda\mathbf{x}}$ we have $Q_1 \in SN_{\lambda\mathbf{x}}$, and by the induction hypothesis $Q_1 \in \mathcal{SN}$. Then by inference scheme $(R5)$ we obtain that $M \in \mathcal{SN}$.
    - $P_0 = \lambda y.N$. Then $(\lambda y.N\langle x_1 := Q_1\rangle)\langle x_2 := Q_2\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n \in SN_{\lambda\mathbf{x}}$. By the induction hypothesis $(\lambda y.N\langle x_1 := Q_1\rangle)\langle x_2 := Q_2\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n \in \mathcal{SN}$. Then using inference scheme $(R7)$ we obtain that $M \in \mathcal{SN}$.
    - $P_0 = N_1 N_2$. Then we have $(N_1\langle x_1 := Q_1\rangle N_2\langle x_1 := Q_1\rangle)\langle x_2 := Q_2\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n \in SN_{\lambda\mathbf{x}}$. And $(N_1\langle x_1 := Q_1\rangle N_2\langle x_1 := Q_1\rangle)\langle x_2 := Q_2\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n \in \mathcal{SN}$, by the induction hypothesis. Using inference scheme $(R6)$ we may conclude that $M \in \mathcal{SN}$.

- $\mathcal{SN} \subseteq SN_{\lambda\mathbf{x}}$. By induction on the derivation of $M \in \mathcal{SN}$.
  - $M = xP_1 \dots P_n$ where $P_1, \dots, P_n \in \mathcal{SN}$. By the induction hypothesis we have $P_1, \dots, P_n \in SN_{\lambda\mathbf{x}}$. And therefore $xP_1 \dots P_n$ is also strongly λx-normalizing.
  - $M = \lambda x.P$ where $P \in \mathcal{SN}$. Similar to the previous case.
  - $M = (\lambda x.P_0)P_1 \dots P_n$ where $P_0\langle x := P_1\rangle P_2 \dots P_n \in \mathcal{SN}$. Then by induction hypothesis we have $P_0\langle x := P_1\rangle P_2 \dots P_n \in SN_{\lambda\mathbf{x}}$. Thus by the Perpetuality Proposition 3.11(1), $M \in SN_{\lambda\mathbf{x}}$.
  - $M = x_1\langle x_1 := Q_1\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n$ where $Q_1\langle x_2 := Q_2\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n \in \mathcal{SN}$. Then by induction hypothesis $Q_1\langle x_2 := Q_2\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n \in SN_{\lambda\mathbf{x}}$ and thus by the Perpetuality Proposition 3.11(2) we may conclude that $M \in SN_{\lambda\mathbf{x}}$.
  - $M = y\langle x_1 := Q_1\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n$ where $y \neq x_1$, $Q_1 \in \mathcal{SN}$ and $y\langle x_2 := Q_2\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n \in \mathcal{SN}$. Then by induction hypothesis $Q_1 \in SN_{\lambda\mathbf{x}}$ and
    $y\langle x_2 := Q_2\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n \in SN_{\lambda\mathbf{x}}$. And applying the Perpetuality Proposition 3.11(3) we obtain $M \in SN_{\lambda\mathbf{x}}$.
  - $M = (N_1N_2)\langle x_1 := Q_1\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n$ or $M = (\lambda y.N)\langle x_1 := Q_1\rangle \dots \langle x_m := Q_m\rangle P_1 \dots P_n$. Similar to the above cases using the Perpetuality Proposition 3.11(2).

•

Some variations of Def. 3.14 are possible. Rule ($R5$) may be replaced by the following rule ($R5'$) yielding the characterization $\mathcal{SN}'$:

$$\frac{M^* P_1 \dots P_n \in \mathcal{SN} \quad N \in \mathcal{SN} \quad x \in \Sigma_v}{M\langle x := N\rangle^* P_1 \dots P_n \in \mathcal{SN}} \; R5'$$

with the restriction that $x \notin FV(M)$. Then Lemma 3.15 may be proved for $\mathcal{SN}'$. As a consequence, we also obtain $\mathcal{SN} = \mathcal{SN}'$.

Another possible characterization of $SN_{\lambda\mathbf{x}}$, as reported in [Xi96] for the λ-calculus with β-reduction, is $SN_{\lambda\mathbf{x}} = \{M \in \mathcal{T}_{\lambda\mathbf{x}} \mid \mathcal{H}(M) < \infty\}$ where

$$\mathcal{H}(M_1) = \max\{n \mid M_1 \triangleright M_2 \triangleright \dots \triangleright M_n\}$$

and $\triangleright$ is the union of the strict subterm relation and leftmost reduction for λx (written here $\rightarrow_l$), i.e. $\triangleright \overset{\text{def}}{=} \supset \cup \rightarrow_l$ where $\lambda x.P \supset P$, $PQ \supset P$, $PQ \supset Q$, $P\langle x := Q\rangle \supset P$, $P\langle x := Q\rangle \supset Q$. The proof relies on the Perpetuality Proposition.

3.2.2 Perpetual Rewrite Strategies for λx

Here we extend the perpetual rewrite strategies for the lambda calculus presented in [BK82] and [BBKV76] to the calculus of explicit substitutions λx, and, following [RS95], use the characterization of $SN_{\lambda\mathbf{x}}$ (Def. 3.14) to prove that they indeed constitute perpetual strategies. Section 2.3.1 provides the definition of λx-rewrite strategy. Recall that a λx-rewrite strategy $\mathcal{F}(\bullet)$ is called λx-*perpetual* if $\infty_{\lambda\mathbf{x}}(M)$ implies $\infty_{\lambda\mathbf{x}}(\mathcal{F}(M))$

First we present the effective strategy $\mathcal{F}_\infty(\bullet)$. Then the strategy $\mathcal{F}(\bullet)$ shall be considered.

Definition 3.16 (The strategy $\mathcal{F}_\infty(\bullet)$) Let M be a term in $\mathcal{T}_{\lambda\mathbf{x}}$ which is not in normal form. Let $M = C[\Delta]$ where Δ is the leftmost $\lambda\mathbf{x}^-$-redex[2] of M. We define $\mathcal{F}_\infty(\bullet)$ as follows:

$$\begin{array}{lcll}
\mathcal{F}_\infty(C[(\lambda y.P)Q]) & \overset{\text{def}}{=} & C[P\langle y := Q\rangle] & \\
\mathcal{F}_\infty(C[(\lambda y.P)\langle z := Q\rangle]) & \overset{\text{def}}{=} & C[\lambda y.P\langle z := Q\rangle] & \\
\mathcal{F}_\infty(C[(P_1P_2)\langle y := Q\rangle]) & \overset{\text{def}}{=} & C[P_1\langle y := Q\rangle P_2\langle y := Q\rangle] & \\
\mathcal{F}_\infty(C[y\langle y := Q\rangle]) & \overset{\text{def}}{=} & C[Q] & \\
\mathcal{F}_\infty(C[z\langle y := Q\rangle]) & \overset{\text{def}}{=} & \left\{\begin{array}{l} C[z] \\ C[z\langle y := \mathcal{F}_\infty(Q)\rangle] \end{array}\right. & \begin{array}{l} \text{if } Q \text{ in normal form,} \\ \text{otherwise} \end{array}
\end{array}$$

In the last inference scheme we have $y \neq z$.

[2] Recall that a $\lambda\mathbf{x}^-$-redex is a λx-redex.

The idea in $\mathcal{F}_\infty(\bullet)$ is to always contract the leftmost λx^--redex unless a (potentially infinite) term may be erased by the restricted garbage collection rule (the only rule which may erase subterms).

Proposition 3.17 $\mathcal{F}_\infty(\bullet)$ is a perpetual rewrite strategy for λx.

Proof. We prove that if $\mathcal{F}_\infty(M)$ is strongly normalizing then M is strongly normalizing. For this, the characterization of Def. 3.14 shall be used. Thus we assume $\mathcal{F}_\infty(M) \in \mathcal{SN}$ and we shall show that $M \in \mathcal{SN}$ by induction on the number of derivation steps of $\mathcal{F}_\infty(M) \in \mathcal{SN}$. By Lemma 3.13 we have the following cases to consider for M:

- $M = yP_1, \dots P_n$. Then we have $\mathcal{F}_\infty(M) = yP_1 \dots P_{i-1}\mathcal{F}_\infty(P_i)P_{i+1} \dots P_n$ where $P_1, \dots, P_{i-1}$ are in normal form. Then since $\mathcal{F}_\infty(M) \in \mathcal{SN}$ we have $\mathcal{F}_\infty(P_i) \in \mathcal{SN}$. Then by the induction hypothesis $P_i \in \mathcal{SN}$ and we can construct a derivation in $\mathcal{SN}$ of M.
- $M = \lambda y.P$. Then we have $\mathcal{F}_\infty(M) = \lambda y.\mathcal{F}_\infty(P)$. By the induction hypothesis $P \in \mathcal{SN}$. Hence, $M \in \mathcal{SN}$.
- $M = (\lambda y.P)P_1 \dots P_n$. Then we have $\mathcal{F}_\infty(M) = P\langle y := P_1\rangle P_2 \dots P_n$. Since $\mathcal{F}_\infty(M) \in \mathcal{SN}$ using inference scheme $(R3)$ we obtain that $M \in \mathcal{SN}$.
- $M = P_0\langle y_1 := Q_1\rangle \dots \langle y_m := Q_m\rangle P_1 \dots P_n$ with $m \geq 1$ and P_0 not a closure (i.e. not of the form $P'\langle z := Q'\rangle$). We consider four further cases:
  - $P_0 = y_1$. Then $\mathcal{F}_\infty(M) = Q_1\langle y_2 := Q_2\rangle \dots \langle y_m := Q_m\rangle P_1 \dots P_n$. Then by using inference scheme $(R4)$ we obtain $M \in \mathcal{SN}$.
  - $P_0 = z \neq y_1$. Then we have two cases to consider:
    - Q_1 is in normal form. Then $\mathcal{F}_\infty(M) = z\langle y_2 := Q_2\rangle \dots \langle y_m := Q_m\rangle P_1 \dots P_n$. Since $Q_1 \in SN_{\lambda x}$ we have $Q_1 \in \mathcal{SN}$. And then by using inference scheme $(R5)$ we obtain $M \in \mathcal{SN}$.
    - Q_1 is not in normal form. Then we have $\mathcal{F}_\infty(M) = z\langle y_1 := \mathcal{F}_\infty(Q_1)\rangle \dots \langle y_m := Q_m\rangle P_1 \dots P_n$. Now since $\mathcal{F}_\infty(M) \in \mathcal{SN}$ then $\mathcal{F}_\infty(Q_1) \in \mathcal{SN}$ and hence, by the induction hypothesis, $Q_1 \in \mathcal{SN}$. Then we can construct a derivation in $\mathcal{SN}$ of M using clause $(R5)$.
  - $P_0 = P_0'P_0''$. Then $\mathcal{F}_\infty(M) = (P_0'\langle y_1 := Q_1\rangle P_0''\langle y_1 := Q_1\rangle)\langle y_2 := Q_2\rangle \dots \langle y_m := Q_m\rangle P_1 \dots P_n$. Then by using inference scheme $(R6)$ we obtain $M \in \mathcal{SN}$.
  - $P_0 = \lambda z.P_0'$. Then $\mathcal{F}_\infty(M) = (\lambda z.P_0'\langle y_1 := Q_1\rangle)\langle y_2 := Q_2\rangle \dots \langle y_m := Q_m\rangle P_1 \dots P_n$. Then by using inference scheme $(R7)$ we obtain $M \in \mathcal{SN}$.

Remark 3.18 Let F be a λx-rewrite strategy and define

$$L_F(M) \stackrel{\text{def}}{=} \min\{n \mid F^n(M) \text{ is in } \lambda x\text{-normal form}\}$$

where $F^0(M) \stackrel{\text{def}}{=} M$ and $F^{n+1}(M) \stackrel{\text{def}}{=} F(F^n(M))$. Then F is said to be *maximal* iff $L_F(M) = \mathit{maxred}_{\lambda x}(M)$ for every λx-term M. The question whether $\mathcal{F}_\infty(\bullet)$ is maximal is left open. The perpetual rewrite strategy F_∞ for the λ-calculus [BBKV76] has been shown to be a maximal strategy [Sør96].

Definition 3.19 (The strategy $\mathcal{F}(\bullet)$) Let M be a term in $\mathcal{T}_{\lambda x}$ which is not in normal form. Let $M = C[\Delta]$ where Δ is the leftmost λx^--redex of M. We define $\mathcal{F}(\bullet)$ as follows:

$$\begin{array}{lcl}
\mathcal{F}(C[(\lambda y.P)Q]) & \stackrel{\text{def}}{=} & C[P\langle y := Q\rangle] \\
\mathcal{F}(C[(\lambda z.P)\langle y := Q\rangle]) & \stackrel{\text{def}}{=} & C[\lambda z.P\langle y := Q\rangle] \\
\mathcal{F}(C[(P_1P_2)\langle y := Q\rangle]) & \stackrel{\text{def}}{=} & C[P_1\langle y := Q\rangle P_2\langle y := Q\rangle] \\
\mathcal{F}(C[y\langle y := Q\rangle]) & \stackrel{\text{def}}{=} & C[Q] \\
\mathcal{F}(C[z\langle y := Q\rangle]) & \stackrel{\text{def}}{=} & \left\{ \begin{array}{ll} C[z] & \text{if } Q \in SN_{\lambda x} \\ C[z\langle y := \mathcal{F}(Q)\rangle] & \text{if } Q \notin SN_{\lambda x} \end{array} \right.
\end{array}$$

In the last inference scheme we have $y \neq z$.

Proposition 3.20 $\mathcal{F}(\bullet)$ is a perpetual rewrite strategy for λx.

Proof. The proof is similar to that of Proposition 3.17. $\bullet$

Note that these strategies are not refinings of the original ones (those defined on the λ-calculus with β-reduction) in the following sense: if we start from a pure term M then the strategy for λx yields a rewrite sequence consisting of a *Beta*-rewrite step followed by x-reduction to x-normal form, which corresponds to the β-reduction step induced by the original strategy. Before illustrating with an example we recall the strategy $F_\infty(\bullet)$ [BBKV76].

Definition 3.21 ($F_\infty(\bullet)$) Let M be a term in $\mathcal{T}_\lambda$ which is not in β-normal form (denoted $M \notin NF_\beta$). Let $M = C[(\lambda y.P)Q]$ where $(\lambda y.P)Q$ is the leftmost β-redex of M. Then $F_\infty(M)$ is defined as follows:

$$F_\infty(C[(\lambda y.P)Q]) \overset{\text{def}}{=} \begin{cases} C[P\{y \leftarrow Q\}] & \text{if } y \in FV(P) \text{ or } Q \in NF_\beta \\ C[(\lambda y.P)F_\infty(Q)] & \text{if } y \notin FV(P) \text{ and } Q \notin NF_\beta \end{cases}$$

Let $M \in \mathcal{T}_\lambda$. More generally, one may wonder whether there exists a positive integer n such $M \xrightarrow{+}_{\lambda x} \mathcal{F}^n_\infty(M)$ and $\mathcal{F}^n_\infty(M) = F_\infty(M)$. Consider the term $M = (\lambda x.\Delta\Delta x)z$ where $\Delta = \lambda y.yy$. Then $F_\infty(M) = \Delta\Delta z$. On the other hand, for no positive integer n is $\mathcal{F}^n_\infty(M)$ a pure term (and hence $\mathcal{F}^n_\infty(M) \neq F_\infty(M)$) as the following portion of reduction of the strategy $\mathcal{F}_\infty(\bullet)$ suggests:

$$\begin{array}{ll}
 & M \\
\rightarrow_{Beta} & (\Delta\Delta x)\langle x := z\rangle \\
\rightarrow_{App} & (\Delta\Delta)\langle x := z\rangle\ x\langle x := z\rangle \\
\rightarrow_{App} & \Delta\langle x := z\rangle\ \Delta\langle x := z\rangle\ x\langle x := z\rangle \\
\rightarrow_{Lam} & (\lambda y.(yy)\langle x := z\rangle)\ \Delta\langle x := z\rangle\ x\langle x := z\rangle \\
\rightarrow_{Beta} & (yy)\langle x := z\rangle\langle y := \Delta\langle x := z\rangle\rangle\ x\langle x := z\rangle \\
\rightarrow_{App} & (y\langle x := z\rangle\ y\langle x := z\rangle)\langle y := \Delta\langle x := z\rangle\rangle\ x\langle x := z\rangle \\
\rightarrow_{App} & y\langle x := z\rangle\langle y := \Delta\langle x := z\rangle\rangle\ y\langle x := z\rangle\langle y := \Delta\langle x := z\rangle\rangle\ x\langle x := z\rangle \\
\rightarrow_{Gc} & y\langle y := \Delta\langle x := z\rangle\rangle\ y\langle x := z\rangle\langle y := \Delta\langle x := z\rangle\rangle\ x\langle x := z\rangle \\
\rightarrow_{Var} & \Delta\langle x := z\rangle\ y\langle x := z\rangle\langle y := \Delta\langle x := z\rangle\rangle\ x\langle x := z\rangle \\
 & \ldots
\end{array}$$

We shall continue our discussion on perpetual rewrite strategies for λx in Section 4.1. The interested reader is invited to skip to the aforementioned section resting assured that the material on the polymorphic lambda calculus with explicit substitutions to be presented next may be read independently.

3.3 The Polymorphic Lambda Calculus with Explicit Substitutions

As a final application of our studies in perpetuality for λx we shall formulate a polymorphic lambda calculus with explicit substitutions called F_{es} and prove the properties of subject reduction and strong normalization (of its typed version). The first subsection presents the untyped F-calculus with explicit substitutions and then introduces its typed version. Subject reduction followed by strong normalization is considered next.

3.3.1 F_{es}: The Rewrite Rules

In this subsection we introduce the polymorphic lambda calculus [Gir72, GLT89] with explicit substitutions. This rewrite system which we shall call F_{es} will first be introduced as an untyped calculus in the sense that no typing rules for terms shall be given, Section 3.3.2 shall deal with its typed version.

Let $\mathcal{V}_t$ be an infinite set of type variables $s, t, u, \ldots$, and $\mathcal{V}$ be an infinite set of term variables (referred to simply as variables) $x, y, z, \ldots$.

Definition 3.22 The set of types and terms (referred to, without distinction, as raw terms) of the F_{es}-calculus is defined by the following two grammars, respectively:

$$\begin{array}{ll}
\textbf{types} & \sigma ::= t \mid \sigma_1 \rightarrow \sigma_2 \mid \forall t.\sigma \mid \sigma[t := \sigma] \\
\textbf{terms} & M ::= x \mid \lambda x : \sigma.M \mid \Lambda t.M \mid MN \mid M\sigma \mid M\langle x := N\rangle \mid M[t := \sigma]
\end{array}$$

The operator $\bullet\langle\bullet := \bullet\rangle$ is called the *term substitution operator* and $\bullet[\bullet := \bullet]$ the *type substitution operator*. A term which does not contain occurrences of a (term or type) substitution operator is called a *pure term*. Likewise, a type which does not contain occurrences of the type substitution operator is called a *pure type*.

We denote with $\mathcal{RT}$ the set of all raw terms, i.e. those generated by the **terms** and **types** grammars, and $\mathcal{T}$ the set of proper terms in $\mathcal{RT}$, i.e. those generated by the **terms** grammar. Letters $\rho, \sigma, \tau, \ldots$ are used for types, letters $M, N, O, P, Q, \ldots$ are used for terms and letters $A, B, C, \ldots$ are used for raw terms.

The set of free variables of a term M is denoted $FV(M)$ and the set of free type variables of a raw term A is denoted $FTV(A)$. We also use $BV(M)$ and $BTV(A)$ to denote the set of bound variables of M and bound type variables of A, respectively. Also, we sometimes use $BTV(A, B)$ as a shorthand for $BTV(A) \cup BTV(B)$. These notions are defined as usual; we give below the definition of FTV as an example.

Definition 3.23 Let A be a raw term. The set of *free type variables of* A is defined as follows:

$$\begin{array}{llll}
FTV(t) & \stackrel{\text{def}}{=} \{t\} & FTV(\Lambda t.P) & \stackrel{\text{def}}{=} FTV(P) \setminus \{t\} \\
FTV(\sigma \to \tau) & \stackrel{\text{def}}{=} FTV(\sigma) \cup FTV(\tau) & FTV(PQ) & \stackrel{\text{def}}{=} FTV(P) \cup FTV(Q) \\
FTV(\forall t.\sigma) & \stackrel{\text{def}}{=} FTV(\sigma) \setminus \{t\} & FTV(P\sigma) & \stackrel{\text{def}}{=} FTV(P) \cup FTV(\sigma) \\
FTV(\sigma[t := \tau]) & \stackrel{\text{def}}{=} (FTV(\sigma) \setminus \{t\}) \cup FTV(\tau) & FTV(P\langle x := Q\rangle) & \stackrel{\text{def}}{=} FTV(P) \cup FTV(Q) \\
FTV(x) & \stackrel{\text{def}}{=} \emptyset & FTV(P[t := \sigma]) & \stackrel{\text{def}}{=} (FTV(P) \setminus \{t\}) \cup FTV(\sigma) \\
FTV(\lambda x : \sigma.P) & \stackrel{\text{def}}{=} FTV(\sigma) \cup FTV(P) & &
\end{array}$$

As mentioned in Chapter 2, when dealing with calculi of explicit substitution and in order to obtain first order rewriting systems for the λ-calculus, it is not uncommon to use de Bruijn indices notation. Since our main objective is to concentrate on the properties of subject reduction and strong normalization we have chosen to use the variable name based λx in order to minimize the 'noise' introduced by updating (i.e. $\mathcal{F}(\bullet)$index updating in de Bruijn indices calculi) and thus provide the reader with a more intuitive setting.

We recall the familiar notions of substitution and α-conversion for terms (types are treated similarly). The present definition extends Def. 2.32 to the extended syntax of F_{es}.

$$\begin{array}{lll}
x\{x \leftarrow N\} & \stackrel{\text{def}}{=} & N \\
y\{x \leftarrow N\} & \stackrel{\text{def}}{=} & y, \text{ if } x \neq y \\
(M_1 M_2)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & M_1\{x \leftarrow N\} M_2\{x \leftarrow N\} \\
(\lambda x : \sigma.M)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & \lambda x : \sigma.M \\
(\lambda y : \sigma.M)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & \lambda y' : \sigma.M\{y \leftarrow y'\}\{x \leftarrow N\}, \\
 & & \text{if } y' \notin FV(N) \cup \{x\} \cup (FV(M) \setminus \{y\}) \\
(M_1\langle x := M_2\rangle)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & M_1\langle x := M_2\{x \leftarrow N\}\rangle \\
(M_1\langle y := M_2\rangle)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & M_1\{y \leftarrow y'\}\{x \leftarrow N\}\langle y' := M_2\{x \leftarrow N\}\rangle, \\
 & & \text{if } y' \notin FV(N) \cup \{x\} \cup (FV(M_1) \setminus \{y\}) \\
(\Lambda t.M)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & \Lambda t.M\{x \leftarrow N\} \\
(M\sigma)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & M\{x \leftarrow N\}\sigma \\
\tau[t := \sigma]\{x \leftarrow N\} & \stackrel{\text{def}}{=} & \tau[t := \sigma] \\
M[t := \sigma]\{x \leftarrow N\} & \stackrel{\text{def}}{=} & M\{t \leftarrow s\}\{x \leftarrow N\}[s := \sigma] \\
 & & \text{if } s \notin FTV(N) \cup (FTV(M) \setminus \{t\})
\end{array}$$

Syntactical equality modulo α-conversion is thus the smallest equivalence relation verifying:

$$\begin{array}{lll}
\text{if } M = P \text{ and } A = B & \text{then} & MA = PB \\
\text{if } M = N, \sigma = \tau \text{ and } y \notin FV(N) \setminus \{x\} & \text{then} & \lambda x : \sigma.M = \lambda y : \tau.N\{x \leftarrow y\} \\
\text{if } M = P, N = Q \text{ and } y \notin FV(N) \setminus \{x\} & \text{then} & M\langle x := N\rangle = P\{x \leftarrow y\}\langle y := Q\rangle \\
\text{if } M = P \text{ and } s \notin FTV(P) \setminus \{t\} & \text{then} & \Lambda t.M = \Lambda s.P\{t \leftarrow s\} \\
\text{if } A = B, \sigma = \tau \text{ and } s \notin FTV(B) \setminus \{t\} & \text{then} & A[t := \sigma] = B\{t \leftarrow s\}[s := \tau]
\end{array}$$

As in the untyped lambda calculus we shall adopt the following variable convention: we assume that the names of bound (term or type) variables shall always be chosen so that they differ from the free ones. Moreover, each occurrence of a (term or type) abstraction operator has a different binding variable.

$$
\begin{array}{llll}
(\lambda x:\sigma.M)N & \rightarrow_{Beta2} & M\langle x:=N\rangle & \\
(\Lambda t.M)\tau & \rightarrow_{\tau Beta} & M[t:=\tau] & \\
 & & & \\
(MN)\langle x:=P\rangle & \rightarrow_{zapp} & M\langle x:=P\rangle N\langle x:=P\rangle & \\
(\lambda y:\sigma.M)\langle x:=P\rangle & \rightarrow_{zlam} & \lambda y:\sigma.(M\langle x:=P\rangle) & \\
x\langle x:=P\rangle & \rightarrow_{zvar} & P & \\
M\langle x:=P\rangle & \rightarrow_{zgc} & M & x\notin FV(M) \\
(M\sigma)\langle x:=P\rangle & \rightarrow_{zappt} & M\langle x:=P\rangle\sigma & \\
(\Lambda t.M)\langle x:=P\rangle & \rightarrow_{zlamt} & \Lambda t.M\langle x:=P\rangle & \\
 & & & \\
(MN)[t:=\tau] & \rightarrow_{ztapp} & M[t:=\tau]N[t:=\tau] & \\
(\lambda y:\sigma.M)[t:=\tau] & \rightarrow_{ztlam} & \lambda y:\sigma[t:=\tau].(M[t:=\tau]) & \\
t[t:=\tau] & \rightarrow_{ztvar1} & \tau & \\
\sigma[t:=\tau] & \rightarrow_{ztgc} & \sigma & t\notin FTV(\sigma) \\
s[t:=\tau] & \rightarrow_{ztvar2} & s & \\
(M\sigma)[t:=\tau] & \rightarrow_{ztappt} & M[t:=\tau]\sigma[t:=\tau] & \\
(\Lambda u.M)[t:=\tau] & \rightarrow_{ztlamt} & \Lambda u.M[t:=\tau] & \\
(\sigma_1\rightarrow\sigma_2)[t:=\tau] & \rightarrow_{zt} & \sigma_1[t:=\tau]\rightarrow\sigma_2[t:=\tau] & \\
(\forall u.\sigma)[t:=\tau] & \rightarrow_{zt} & \forall u.\sigma[t:=\tau] &
\end{array}
$$

Figure 3.1: The F_{es}-calculus

Definition 3.24 (**F_{es} and subsystems**) The F_{es}-calculus is given by the rules in Figure 3.1[3]. The F_{es}-calculus without the rules *Beta2* and *τBeta* is referred to as the ES-rewrite system (or ES-calculus). The F_{es}-calculus without the *Beta2* rule is called the ES^{τ}-rewrite system. The third group of rules is called the ZT-rewrite system.

As regards the choice of the rules perhaps it is worth mentioning that the *Gc*-rule may be replaced by the more restricted garbage collection rule *RGc* without affecting the results. The same applies to the *ztgc*-rule and what might be called the restricted garbage collection rule for types $t[s:=\tau]\rightarrow_{rztgc} t$. Also, the *ztvar2*-rule may be replaced by the more general rule $M[t:=\tau]\rightarrow_{ztgc2} M$.

We shall now prove that the ES^{τ}-rewrite system is strongly normalizing. Intuitively, one notes that the substitution calculus ES 'pushes' substitution operators deeper and deeper into a term/type until they are performed or eliminated. As for the rule *τBeta* one notes that it eliminates an occurrence of the binder Λ. Thus we shall first prove that the ES-rewrite system is strongly normalizing by interpreting it into a simpler calculus equipped with a well-founded reduction notion and showing that reduction is preserved. Then for the full system ES^{τ} we use this result plus Lemmas 2.6 (see appendix) and 3.31.

Definition 3.25 The set of terms constructed from the alphabet $\mathcal{A}=\{\star,\lambda(\bullet,\bullet),\Lambda\bullet,\bullet\rightarrow\bullet,\bullet.\bullet,\bullet\llbracket\bullet\rrbracket\}$, denoted $\mathcal{S}$, is defined by the grammar $a::=\star\mid\lambda(a,a)\mid\Lambda a\mid a.a\mid a\llbracket a\rrbracket\mid a\rightarrow a$. Also, we define the following well-founded ordering $\llbracket\rrbracket\gg\,.\,\gg\star,\lambda,\Lambda,\rightarrow$ on the alphabet $\mathcal{A}$.

Assuming all symbols of $\mathcal{A}$ have multiset status, this well-founded ordering induces a well-founded Recursive Path Ordering (RPO) on the full set of terms $\mathcal{S}$ [Der82] denoted $\succ_{\mathcal{T}_l}$ (see appendix).

Definition 3.26 We define the translation $\mathcal{R}_1(\bullet):\mathcal{RT}\longrightarrow\mathcal{S}$ as

[3] The reason for prefixing the second group of rules with a z is somewhat arbitrary, however some resource for distinguishing these rules from those of λx was sought for since reduction between F_{es} and λx is later compared (Lemma 3.48(2)).

$$\begin{array}{lcllcl}
\mathcal{R}_1(x) = \mathcal{R}_1(t) & \overset{\text{def}}{=} & \star & \mathcal{R}_1(M\langle x := N\rangle) & \overset{\text{def}}{=} & \mathcal{R}_1(M)\llbracket\mathcal{R}_1(N)\rrbracket \\
\mathcal{R}_1(MN) & \overset{\text{def}}{=} & \mathcal{R}_1(M).\mathcal{R}_1(N) & \mathcal{R}_1(M[t := \tau]) & \overset{\text{def}}{=} & \mathcal{R}_1(M)\llbracket\mathcal{R}_1(\tau)\rrbracket \\
\mathcal{R}_1(M\sigma) & \overset{\text{def}}{=} & \mathcal{R}_1(M).\mathcal{R}_1(\sigma) & \mathcal{R}_1(\sigma \rightarrow \tau) & \overset{\text{def}}{=} & \mathcal{R}_1(\sigma) \rightarrow \mathcal{R}_1(\tau) \\
\mathcal{R}_1(\lambda x : \sigma.(M)) & \overset{\text{def}}{=} & \lambda(\mathcal{R}_1(\sigma), \mathcal{R}_1(M)) & \mathcal{R}_1(\forall t.\sigma) & \overset{\text{def}}{=} & \Lambda\mathcal{R}_1(\sigma) \\
\mathcal{R}_1(\Lambda t.M) & \overset{\text{def}}{=} & \Lambda\mathcal{R}_1(M) & & &
\end{array}$$

Note that $\mathcal{R}_1(\bullet)$ simply forgets all variables and all binding variables. For example $\mathcal{R}_1(x\langle x := y\rangle) = \star\llbracket\star\rrbracket$.

Lemma 3.27 The ES-rewrite system is strongly normalizing on $\mathcal{RT}$.

Proof. One shows that if A, B are raw terms in $\mathcal{RT}$ such that $A \rightarrow_{ES} B$ then we have $\mathcal{R}_1(A) \succ_{\mathcal{T}_l} \mathcal{R}_1(B)$.

■

Since the ES-rewrite system is also locally confluent, by Newman's Lemma we may conclude that it is confluent. Thus if $A \in \mathcal{RT}$ then we shall use $ES(A)$ to denote the unique ES-normal form of A. Since the ZT subcalculus is strongly normalizing (it is included in ES) and locally confluent then by Newman's Lemma we may also conclude that it is confluent. Thus if σ is a type then we use $ZT(\sigma)$ to denote its unique ZT-normal form. Also, we shall use $=_{ZT}$ to denote the reflexive, symmetrical and transitive closure of one step reduction in ZT.

Remark 3.28 One may verify in the style of [Río93, pp.63-64] that if σ is a type then $ZT(\sigma)$ is a pure type and if A is a raw term then $ES(A)$ is a pure term or type. Note that if $M \in \mathcal{T}$ then $ZT(M)$ not only may not be a pure term (for example if it has occurrences of the term substitution operator) but also may not have pure types since term substitutions may block the execution of type substitutions.

Before proceeding to the property of strong normalization of ES^{τ}, we mention some technical properties of ZT and ES which shall be used later.

Lemma 3.29 Let ρ, σ, τ be pure types, P and Q terms in $\mathcal{T}$, and τ_1, τ_2 any types.

1. $\rho\{t \leftarrow \tau\}\{u \leftarrow \sigma\{t \leftarrow \tau\}\} = \rho\{u \leftarrow \sigma\}\{t \leftarrow \tau\}$ if $u \notin FTV(\tau)$.
2. $ZT(\tau_1[t := \tau_2]) = ZT(\tau_1)\{t \leftarrow ZT(\tau_2)\}$.
3. $ES(P\langle x := Q\rangle) = ES(P)\{x \leftarrow ES(Q)\}$.
4. $ES(P[t := \tau_1]) = ES(P)\{t \leftarrow ZT(\tau_1)\}$.

Proof. The first item is the substitution lemma and is proved by induction on ρ. The second item is first proved for τ_1 and τ_2 pure types and then using Remark 3.28. Item three is dealt with in a similar fashion. Item four is similar, but uses the observation that for a type σ, we have $ES(\sigma) = ZT(\sigma)$ and also uses item (2).

■

Next we extend the SN property of ES to the rewrite system ES^{τ}, i.e. ES with the additional rule $\tau Beta$. For this we use Lemma 2.6 (see appendix). We thus define a translation $\mathcal{R}_2(\bullet)$ from terms in $\mathcal{RT}$ to $\mathcal{S}$ such that the conditions of Lemma 2.6 are met. The latter translation shall forget type applications and type substitutions.

Definition 3.30 We define the translation $\mathcal{R}_2(\bullet) : \mathcal{RT} \longrightarrow \mathcal{S}$ as

$$\begin{array}{lcllcl}
\mathcal{R}_2(x) = \mathcal{R}_2(\sigma) & \overset{\text{def}}{=} & \star & \mathcal{R}_2(\Lambda t.M) & \overset{\text{def}}{=} & \Lambda\mathcal{R}_2(M) \\
\mathcal{R}_2(MN) & \overset{\text{def}}{=} & \mathcal{R}_2(M).\mathcal{R}_2(N) & \mathcal{R}_2(M\langle x := N\rangle) & \overset{\text{def}}{=} & \mathcal{R}_2(M)\llbracket\mathcal{R}_2(N)\rrbracket \\
\mathcal{R}_2(M\sigma) & \overset{\text{def}}{=} & \mathcal{R}_2(M) & \mathcal{R}_2(M[t := \tau]) & \overset{\text{def}}{=} & \mathcal{R}_2(M) \\
\mathcal{R}_2(\lambda x : \sigma.(M)) & \overset{\text{def}}{=} & \lambda(\mathcal{R}_2(\sigma), \mathcal{R}_2(M)) & & &
\end{array}$$

Lemma 3.31 Let M and N be terms in $\mathcal{T}$. Then

1. If $M \rightarrow_{\tau Beta} N$ then $\mathcal{R}_2(M) \succ_{\mathcal{T}_l} \mathcal{R}_2(N)$, and

2. If $M \to_{ES} N$ then $\mathcal{R}_2(M) \succeq_{\mathcal{T}_l} \mathcal{R}_2(N)$.

Proof. In both items we use induction on the reduction $M \to_r N$ with $r = \tau Beta$ or $r \in ES$. Let us consider the first item. Suppose the reduction is at the root. Then $M = (\Lambda t.P)\tau$ and $N = P[t := \tau]$ and $\mathcal{R}_2(M) = \Lambda\mathcal{R}_2(P)$ and $\mathcal{R}_2(N) = \mathcal{R}_2(P)$ and hence we obtain the desired result. If the reduction is internal then we apply the inductive hypothesis and the monotonicity of RPOs.

The second item is similar. Note that if $M \to_{ZT} N$ or $M \to_{zappt} N$ then $\mathcal{R}_2(M) = \mathcal{R}_2(N)$. For the remaining rules we first consider the case where the reduction takes place at the root, for when the reduction is internal we may apply the induction hypothesis and the monotonicity of $\succeq_{\mathcal{T}_l}$.

- $r = zapp$. Then $M = (PQ)\langle x := R\rangle$ and $N = P\langle x := R\rangle Q\langle x := R\rangle$ and $\mathcal{R}_2(M) = (\mathcal{R}_2(P).\mathcal{R}_2(Q))\llbracket\mathcal{R}_2(R)\rrbracket$ and $\mathcal{R}_2(N) = (\mathcal{R}_2(P)\llbracket\mathcal{R}_2(R)\rrbracket).(\mathcal{R}_2(Q)\llbracket\mathcal{R}_2(R)\rrbracket)$. Since $\llbracket\rrbracket \gg .$ we verify that $\mathcal{R}_2(M) \succeq_{\mathcal{T}_l} \mathcal{R}_2(P)\llbracket\mathcal{R}_2(R)\rrbracket$ and $\mathcal{R}_2(M) \succeq_{\mathcal{T}_l} \mathcal{R}_2(Q)\llbracket\mathcal{R}_2(R)\rrbracket$. Let us consider the first of these, the other one being similar. By the 'equal heads' case we have $\langle\mathcal{R}_2(P).\mathcal{R}_2(Q), \mathcal{R}_2(R)\rangle \succeq'_{\mathcal{T}_l} \langle\mathcal{R}_2(P), \mathcal{R}_2(R)\rangle$[4].

- $r = zlam$. Then $M = (\lambda y : \sigma.P)\langle x := Q\rangle$ and $N = \lambda y : \sigma.P\langle x := Q\rangle$ and we have $\mathcal{R}_2(M) = (\lambda(\mathcal{R}_2(\sigma), \mathcal{R}_2(P))\llbracket\mathcal{R}_2(Q)\rrbracket$ and $\mathcal{R}_2(N) = \lambda(\mathcal{R}_2(\sigma), \mathcal{R}_2(P)\llbracket\mathcal{R}_2(Q)\rrbracket)$. Since $\llbracket\rrbracket \gg \lambda$ we must verify that $\mathcal{R}_2(M) \succeq_{\mathcal{T}_l} \mathcal{R}_2(\sigma)$ and $\mathcal{R}_2(M) \succeq_{\mathcal{T}_l} \mathcal{R}_2(P)\llbracket\mathcal{R}_2(Q)\rrbracket$. The first is direct; for the second we may verify that $\langle\lambda(\mathcal{R}_2(\sigma), \mathcal{R}_2(P)), \mathcal{R}_2(Q)\rangle \succeq'_{\mathcal{T}_l} \langle\mathcal{R}_2(P), \mathcal{R}_2(Q)\rangle$.

- $r = zvar, zgc$. The result follows by the subterm property of recursive path orderings.

- $r = zlamt$. Then $M = (\Lambda t.P)\langle x := Q\rangle$ and $N = \Lambda t.P\langle x := Q\rangle$ and $\mathcal{R}_2(M) = (\Lambda\mathcal{R}_2(P))\llbracket\mathcal{R}_2(Q)\rrbracket$ and $\mathcal{R}_2(N) = \Lambda(\mathcal{R}_2(P)\llbracket\mathcal{R}_2(Q)\rrbracket)$. Then $\llbracket\rrbracket \gg \Lambda$ and we must thus verify that $\mathcal{R}_2(M) \succeq_{\mathcal{T}_l} \mathcal{R}_2(P)\llbracket\mathcal{R}_2(Q)\rrbracket$. We conclude using the 'equal heads' case.

Corollary 3.32 The ES^τ-rewrite system is strongly normalizing on $\mathcal{T}$.

Definition 3.33 (The z and λz-rewrite systems) Let us denote the rewrite system obtained by taking ES and eliminating the subsystem ZT and rules *zappt* and *zlamt*, the z-rewrite system. We define λz as z and the rewrite rule *Beta2*. We recall these rules below.

$$\lambda\mathrm{z}\left\{\begin{array}{lll}
\quad (\lambda x : \sigma.(M))N \;\to_{Beta2} & M\langle x := N\rangle & \\
\mathrm{z}\left\{\begin{array}{lll}
(MN)\langle x := P\rangle & \to_{zapp} & M\langle x := P\rangle N\langle x := P\rangle \\
(\lambda y : \sigma.(M))\langle x := P\rangle & \to_{zlam} & \lambda y : \sigma.(M\langle x := P\rangle) \\
x\langle x := P\rangle & \to_{zvar} & P \\
M\langle x := P\rangle & \to_{zgc} & M \qquad x \notin FV(M)
\end{array}\right. & &
\end{array}\right.$$

The λz and z calculi are the explicit substitution calculus for the simply typed lambda calculus (typed λx) and its subcalculus for computing explicit substitutions (typed x), respectively. Note that from Lemma 3.27 we have that z is strongly normalizing.

3.3.2 F_{es}: The Typing Rules

In this section we introduce the typing rules for F_{es}. As already mentioned, in [Blo97] explicit substitutions were added to the Pure Type Systems formalism [Bar92]. Although we deal only with an explicit substitution version of F, already for this calculus subject reduction was shown to fail. The counterexample exhibited in [Blo97] is the pure term $M = (\Lambda s.(\lambda f : s \to s.\lambda x : s.fx))t$. We refer the reader interested in Explicit Pure Type Systems (*EPTS*) to [Blo97]. After proving subject reduction for F_{es}, we conclude by showing how the typing rules deal with M. Independently in [Blo99, Blo01], the notion of Explicit Type System is introduced, a presentation of pure type systems with explicit substitutions which enjoys the subject reduction property.

Definition 3.34 (Type assignment) A *type assignment* is a finite set of pairs $\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ such that the variables are pairwise distinct and each σ_i is a (not necessarily pure) type. The domain of the type assignment is the set $\{x_1, \ldots, x_n\}$. If the types σ_i with $i \in 1..n$ are pure then Γ is said to be a *pure type assignment*. Type assignments are denoted with capital greek letters $\Gamma, \Delta, \ldots$. We use $Dom(\Gamma)$ to denote the domain of Γ.

[4] A word on notation, $\langle\ldots\rangle$ is used for multisets and $\succeq'_{\mathcal{T}_l}$ is the usual extension of $\succeq_{\mathcal{T}_l}$ to multisets [Oos94, Chapter 1].

Definition 3.35 (Type judgement) A *type judgement* is an expression of the form $\Gamma \rhd M : \sigma$ where Γ is a type assignment, M is a term in $\mathcal{T}$ and σ is a *pure* type. If, moreover, Γ is a pure type assignment and M is a pure term then $\Gamma \rhd M : \sigma$ is said to be a *pure type judgement.*

We shall sometimes use $\Gamma(x_i) = \sigma_i$ to denote the type assigned to the variable x_i by the type assignment $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$. Also we use the abbreviation $t \notin FTV(\Gamma)$ for $t \notin FTV(\Gamma(y))$ for every $y \in Dom(\Gamma)$.

Definition 3.36 Let $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ be a type assignment, t a type variable and τ a type. Then $\Gamma_{[t:=\tau]}$ is any member of the set of type assignments $\{\ \{x_1 : \sigma'_1, \ldots, x_n : \sigma'_n\} \mid \sigma'_i =_{ZT} \sigma_i[t := \tau]\ \mathit{for\ all}\ i \in 1..n\}$.

Definition 3.37 (Derivable type judgements) The set of *derivable type judgements* in F_{es} is defined by the typing rules given in Figure 3.2. The rule *tabs* is subject to the following restriction: $t \notin FTV(\Gamma)$.

$$\dfrac{x : \sigma \in \Gamma}{\Gamma \rhd x : ZT(\sigma)}\ \mathit{var}$$

$$\dfrac{\Gamma \rhd M : \sigma \to \tau \quad \Gamma \rhd N : \sigma}{\Gamma \rhd MN : \tau}\ \mathit{app} \qquad \dfrac{\Gamma, x : \sigma \rhd M : \tau}{\Gamma \rhd (\lambda x : \sigma.(M)) : ZT(\sigma) \to \tau}\ \mathit{abs}$$

$$\dfrac{\Gamma \rhd M : \forall t.\sigma}{\Gamma \rhd M\tau : \sigma\{t \leftarrow ZT(\tau)\}}\ \mathit{tapp} \qquad \dfrac{\Gamma \rhd M : \sigma}{\Gamma \rhd (\Lambda t.M) : \forall t.\sigma}\ \mathit{tabs}$$

$$\dfrac{\Gamma, x : \sigma \rhd M : \tau \quad \Gamma \rhd N : ZT(\sigma)}{\Gamma \rhd M\langle x := N\rangle : \tau}\ \mathit{subs} \qquad \dfrac{\Gamma \rhd M : \sigma}{\Gamma_{[t:=\tau]} \rhd M[t := \tau] : \sigma\{t \leftarrow ZT(\tau)\}}\ \mathit{tsubs}$$

Figure 3.2: Typing rules of the F_{es}-calculus

Definition 3.38 A term $M \in \mathcal{T}$ such that there exists a type assignment Γ and a pure type σ such that the judgement $\Gamma \rhd M : \sigma$ is derivable in the type system of Figure 3.2 is called *typable*. The set of typable terms is denoted $\mathcal{T}^{\to}$.

In a typing judgement $\Gamma \rhd M : \sigma$ the type σ assigned to the term M is always a pure type. Also, we refer to the size of a derivation as the number of applications of rules it contains.

Example 3.39 Two examples of type derivations follow.

$$\dfrac{\dfrac{\dfrac{\dfrac{f : s \to s, x : s \rhd f : s \to s \quad f : s \to s, x : s \rhd x : s}{f : s \to s, x : s \rhd fx : s}\ \mathit{app}}{f : s \to s \rhd \lambda x : s.fx : s \to s}\ \mathit{abs}}{f : t \to t \rhd (\lambda x : s.fx)[s := t] : t \to t}\ \mathit{tsubs}}{\rhd(\lambda f : t \to t.(\lambda x : s.fx)[s := t]) : (t \to t) \to (t \to t)}\ \mathit{abs}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{f : s \to s, x : s \rhd f : s \to s \quad f : s \to s, x : s \rhd x : s}{f : s \to s, x : s \rhd fx : s}\ \mathit{app}}{f : s \to s \rhd \lambda x : s.fx : s \to s}\ \mathit{abs}}{f : (s \to s)[s := t] \rhd (\lambda x : s.fx)[s := t] : t \to t}\ \mathit{tsubs}}{\rhd\lambda f : (s \to s)[s := t].(\lambda x : s.fx)[s := t] : (t \to t) \to (t \to t)}\ \mathit{abs}$$

Note that for the application of *tsubs* in the upper type derivation we use $(t \to t) =_{ZT} (s \to s)[s := t]$.

The typing rules of the usual (Church style presentation) of F is given by all rules in Figure 3.2 without rules *subs* and *tsubs* and restricting all type judgements to pure type judgements. Thus when speaking of derivable type judgements in F we shall explicitly mention so, otherwise we shall assume that the type system referred to is F_{es}.

In devising a rule for typing type substitutions our first intuition is that it should resemble an application of the rule for type variable abstraction followed by that of type variable application. But the introduction of a type substitution in a term M presents some difficulties. Consider a derivable premise $\Gamma \rhd M : \sigma$ and suppose we add the type substitution $[t := \tau]$ to M in order to obtain $\Gamma \rhd M[t := \tau] : \sigma\{t \leftarrow ZT(\tau)\}$. Then we immediately note that the context Γ should not remain the same for if $x \in FV(M)$ then the type assigned to x by the type assignment Γ has now fallen under the scope of the type substitution. For example, consider the application of *tsubs* in the upper derivation of Example 3.39. The variable f, which has type $s \rightarrow s$ in the premise, has now fallen under the scope of type substitution $[s := t]$, so its type must be affected accordingly. This motivates the use of $\Gamma_{[t:=\tau]}$ in the rule *tsubs*.

An alternative approach could be to include the application of the type substitutions in the types of the variables in the type assignment Γ from the start. This requires some mechanism of control in order to ensure that the type substitution applied to the term M is the same as the one which has been applied to the type of the variable in the type assignment (and in the same order if more than one has been applied).

We shall first consider the relation between typing in F and typing in F_{es}. As expected, all terms typable in F are typable in F_{es}(the typing rules for F_{es} include those of F). And if a term is typable in F_{es} then its ES normal form is typable in F (Lemma 3.42).

The following result is used when showing that F has the subject reduction property.

Lemma 3.40 Let $\Gamma, x : \tau \rhd M : \sigma$ and $\Gamma \rhd N : \tau$ be derivable pure type judgements in F. Then $\Gamma \rhd M\{x \leftarrow N\} : \sigma$ is derivable in F.

Proof. By induction on the size of the derivation of $\Gamma, x : \tau \rhd M : \sigma$.

Derivability is closed under type substitution in F.

Lemma 3.41 Let $\Gamma \rhd M : \sigma$ be a pure type judgement and τ a pure type. If $\Gamma \rhd M : \sigma$ is derivable in F and $BTV(M, \sigma, \Gamma) \cap FTV(\tau) = \emptyset$ then $\Gamma\{t \leftarrow \tau\} \rhd M\{t \leftarrow \tau\} : \sigma\{t \leftarrow \tau\}$ is derivable in F.

Proof. By induction on the size of the derivation of $\Gamma \rhd M : \sigma$.

Note that the *subs* rule internalises Lemma 3.40 and rule *tsubs* internalises Lemma 3.41.

Lemma 3.42 Let $\Gamma \rhd M : \sigma$ be a derivable type judgement in F_{es}. Then $ZT(\Gamma) \rhd ES(M) : \sigma$ is derivable in F.

Proof. By induction on the derivation of $\Gamma \rhd M : \sigma$ using Lemmas 3.40 and 3.41 and Lemma 3.29(2) and (3). We shall consider the interesting cases.

- case *subs*. Then the derivation runs

$$\frac{\Gamma, x : \rho \rhd P : \sigma \quad \Gamma \rhd Q : ZT(\rho)}{\Gamma \rhd P\langle x := Q\rangle : \sigma}\ subs$$

By induction hypothesis we have that $ZT(\Gamma), x : ZT(\rho) \rhd ES(P) : \sigma$ and $ZT(\Gamma) \rhd ES(Q) : ZT(\rho)$ are derivable type judgments in F. Then by Lemma 3.40 we have that $ZT(\Gamma) \rhd ES(P)\{x \leftarrow ES(Q)\} : \sigma$ is a derivable type judgement in F. Finally applying Lemma 3.29(3) we are done.

- case (*tsubs*). Then the derivation runs

$$\frac{\Delta \rhd P : \rho}{\Delta_{[t:=\tau]} \rhd P[t := \tau] : \rho\{t \leftarrow ZT(\tau)\}}\ tsubs$$

where $\Gamma = \Delta_{[t:=\tau]}$ and $\sigma = \rho\{t \leftarrow ZT(\tau)\}$. Remark that ρ is pure. Then by the induction hypothesis the type judgement $ZT(\Delta) \rhd ES(P) : \rho$ is derivable in F. Applying Lemma 3.41 we obtain that $ZT(\Delta)\{t \leftarrow ZT(\tau)\} \rhd ES(P)\{t \leftarrow ZT(\tau)\} : \rho\{t \leftarrow ZT(\tau)\}$ is derivable in F. We conclude by observing the following:

- By Lemma 3.29(4) we have $ES(P[t := \tau]) = ES(P)\{t \leftarrow ZT(\tau)\}$.
- On the other hand, suppose $x : \rho_1 \in \Delta$. Then we have $x : \rho_2 \in \Delta_{[t:=\tau]}$ with $\rho_2 =_{ZT} \rho_1[t := \tau]$. Since ZT is confluent and SN we have $ZT(\rho_2) = ZT(\rho_1[t := \tau]) =_{L\ 3.29(2)} ZT(\rho_1)\{t \leftarrow ZT(\tau)\}$, which concludes the case since we then have $ZT(\Delta_{[t:=\tau]}) = ZT(\Delta)\{t \leftarrow ZT(\tau)\}$.

∎

In order to prove the subject reduction property for F_{es} we need two auxiliary results, the weakening lemma and the context reduction lemma.

Lemma 3.43 (Weakening) If $\Gamma \rhd M : \sigma$ is a derivable type judgement, and $x : \tau$ is such that $x \notin Dom(\Gamma) \cup BV(M)$, and $BTV(M) \cap FTV(\tau) = \emptyset$ then $\Gamma, x : \tau \rhd M : \sigma$ is a derivable type judgement.

Proof. By induction on the size of the derivation of $\Gamma \rhd M : \sigma$. The interesting cases are

- case (abs). Then the derivation runs

$$\frac{\Gamma, y : \rho \rhd P : \rho'}{\Gamma \rhd \lambda y : \rho.P : ZT(\rho) \to \rho'}\ abs$$

Now by the conditions of the lemma we have $x \neq y$ and thus by induction hypothesis we obtain

$$\frac{\Gamma, y : \rho, x : \tau \rhd P : \rho'}{\Gamma, x : \tau \rhd \lambda y : \rho.P : ZT(\rho) \to \rho'}\ abs$$

- case $(tsubs)$. Then the derivation runs

$$\frac{\Delta \rhd P : \sigma'}{\Delta_{[t:=\rho]} \rhd P[t := \rho] : \sigma'\{t \leftarrow ZT(\rho)\}}\ tsubs$$

where $\Gamma = \Delta_{[t:=\rho]}$ and $\sigma = \sigma'\{t \leftarrow ZT(\rho)\}$. Then by induction hypothesis we have

$$\frac{\Delta, x : \tau \rhd P : \sigma'}{\Delta_{[t:=\rho]}, x : \tau \rhd P[t := \rho] : \sigma'\{t \leftarrow ZT(\rho)\}}\ tsubs$$

Note that since $t \in BTV(P[t := \rho])$ by the condition of the lemma we have $t \notin FTV(\tau)$ and therefore $\tau[t := \rho] \to_{ztgc} \tau$.

∎

We shall need the following lemma for the subject reduction property, it states that if a context types a term P with type σ then the context resulting from rewriting the original one also types P with σ.

Lemma 3.44 (Context reduction) Let $\Gamma, x : \tau \rhd P : \sigma$ be any derivable type judgement and suppose $\tau \to_{ZT} \tau'$, then $\Gamma, x : \tau' \rhd P : \sigma$ is a derivable type judgement.

Proof. By induction on the size of the derivation of $\Gamma, x : \tau \rhd P : \sigma$.

- case (var). We have two further subcases to consider.
  - $P = x$. Then $\sigma = ZT(\tau)$. And since $\tau \to_{ZT} \tau'$ we have $ZT(\tau) = ZT(\tau')$. Then $\Gamma, x : \tau' \rhd x : ZT(\tau)$ is a derivable type judgement.

  - $P = y \neq x$. Then $\sigma = ZT(\rho)$ with $y : \rho \in \Gamma$ and we are done.

- cases (app), (abs), $(tapp)$, $(subs)$ and $(tabs)$. We use the induction hypothesis. For the case $(tabs)$ note that if $\tau \rightarrow_{ZT} \tau'$ then $FTV(\tau') \subseteq FTV(\tau)$.

- case $(tsubs)$. Then the derivation runs

$$\frac{\Delta, x : \rho \rhd P' : \sigma_2}{\Delta_{[t:=\sigma_1]}, x : \tau \rhd P'[t := \sigma_1] : \sigma_2\{t \leftarrow ZT(\sigma_1)\}}\, tsubs$$

  where $\tau =_{ZT} \rho[t := \sigma_1]$. But then $\tau' =_{ZT} \rho[t := \sigma_1]$. And we may construct the following derivation (see Def. 3.36):

$$\frac{\Delta, x : \rho \rhd P' : \sigma_2}{\Delta_{[t:=\sigma_1]}, x : \tau' \rhd P'[t := \sigma_1] : \sigma_2\{t \leftarrow ZT(\sigma_1)\}}\, tsubs$$

∎

Lemma 3.45 (Subject reduction) Let $\Gamma \rhd M : \sigma$ be a derivable type judgement and suppose $M \rightarrow_{F_{es}} N$. Then $\Gamma \rhd N : \sigma$ is a derivable type judgement.

Proof. By induction on the reduction $M \rightarrow_{F_{es}} N$ using Lemmas 3.43 and 3.44 (see appendix).

∎

We finish this section with the counterexample [Blo97, Prop.7.27] mentioned in the introduction of the section that states that the *EPTS* Fx does not verify the subject reduction property and we show how this situation is remedied.

Example 3.46 Consider the pure term $M = (\Lambda s.(\lambda f : s \rightarrow s.\lambda x : s.fx))t$. It is typable in F_{es}

$$\dfrac{\dfrac{\dfrac{\dfrac{f : s \rightarrow s, x : s \rhd f : s \rightarrow s \quad f : s \rightarrow s, x : s \rhd x : s}{f : s \rightarrow s, x : s \rhd fx : s}\, app}{f : s \rightarrow s \rhd \lambda x : s.fx : s \rightarrow s}\, abs}{\rhd \lambda f : s \rightarrow s.\lambda x : s.fx : (s \rightarrow s) \rightarrow (s \rightarrow s)}\, abs}{\dfrac{\rhd \Lambda s.(\lambda f : s \rightarrow s.\lambda x : s.fx) : \forall s.(s \rightarrow s) \rightarrow (s \rightarrow s)}{\rhd (\Lambda s.(\lambda f : s \rightarrow s.\lambda x : s.fx))t : (t \rightarrow t) \rightarrow (t \rightarrow t)}\, tapp}\, tabs$$

Now we have the following reduction sequence:

$$\begin{array}{lll} M & \rightarrow_{\tau Beta} & (\lambda f : s \rightarrow s.\lambda x : s.fx)[s := t] \\ & \rightarrow_{ztlam} & \lambda f : (s \rightarrow s)[s := t].(\lambda x : s.fx)[s := t] \\ & \twoheadrightarrow_{ZT} & \lambda f : t \rightarrow t.(\lambda x : s.fx)[s := t] \end{array}$$

The last term is also typable as illustrated by Example 3.39 (as well as the intermediate term $\lambda f : (s \rightarrow s)[s := t].(\lambda x : s.fx)[s := t]$).

3.3.3 Strong Normalization of Typed F_{es}

In this subsection we prove the strong normalization property for the typed F_{es} calculus. We follow the presentation given by Gallier in [Gal90]. The idea is to define an erasing function *Erase*(•) that when applied to a typed term in F_{es} eliminates all typing information producing an untyped λx-term, and proceed as follows:

1. show that if a term resulting from erasing all type information, say *Erase*(M), is strongly λx-normalizing then the original term, i.e. M, is strongly F_{es}-normalizing.

2. thus, the result follows if we can show that *Erase*(M) is strongly λx-normalizing for typable M. We generalize this goal by showing that if M is typable then *Erase*(M) concatenated by a series of explicit substitutions is strongly λx-normalizing; clearly this implies that *Erase*(M) itself is strongly λx-normalizing.

3. this previous goal is proved by using the candidates of reducibility technique where the candidates are suitable subsets of $SN_{\lambda\mathrm{x}}$.

Although we have chosen to give a direct proof of strong normalization of F_{es} by means of the reducibility technique, other proofs relating strong normalization of F_{es} to that of F are possible [Blo99, Blo01].

We shall begin by considering the type information erasing function and then consider the reducibility proof.

Definition 3.47 (Erasing function) We define the function *Erase*$(\bullet) : \mathcal{T} \longrightarrow \mathcal{T}_{\lambda\mathrm{x}}$,

$$\begin{array}{llllll}
\mathit{Erase}(x) & \overset{\mathrm{def}}{=} & x & \mathit{Erase}(\Lambda t.M) & \overset{\mathrm{def}}{=} & \mathit{Erase}(M) \\
\mathit{Erase}(MN) & \overset{\mathrm{def}}{=} & \mathit{Erase}(M)\mathit{Erase}(N) & \mathit{Erase}(M\sigma) & \overset{\mathrm{def}}{=} & \mathit{Erase}(M) \\
\mathit{Erase}(M\langle x := N\rangle) & \overset{\mathrm{def}}{=} & \mathit{Erase}(M)\langle x := \mathit{Erase}(N)\rangle & \mathit{Erase}(M[t := \sigma]) & \overset{\mathrm{def}}{=} & \mathit{Erase}(M) \\
 & & & \mathit{Erase}(\lambda x : \sigma.(M)) & \overset{\mathrm{def}}{=} & \lambda x.(\mathit{Erase}(M))
\end{array}$$

We recall that x is the substitution calculus of λx. It propagates the substitution operators until they are performed or eliminated. The following lemma shows that a F_{es}-rewrite step may collapse to a λx-term via the erasing function or be simulated by a λx-rewrite step.

Lemma 3.48 Let $M, N \in \mathcal{T}^{\rightarrow}$.

1. If $M \rightarrow_r N$ with $r \in (F_{es} \setminus \lambda\mathrm{z})$ then *Erase*$(M) =$ *Erase*(N).
2. If $M \rightarrow_{\lambda\mathrm{z}} N$ then *Erase*$(M) \rightarrow_{\lambda\mathrm{x}}$ *Erase*(N).

Proof. Both items are proved by induction on the rewrite step $M \rightarrow_r N$ with $r \in (F_{es} \setminus \lambda\mathrm{z})$ and $r \in \lambda\mathrm{z}$, respectively. We prove the second item:

- The reduction takes place at the root.
  - $r = \mathit{Beta2}$. Then $M = (\lambda x : \sigma.(P))Q$ and $N = P\langle x := Q\rangle$, and therefore we have $\mathit{Erase}(M) = (\lambda x.(\mathit{Erase}(P)))\mathit{Erase}(Q)$ and $\mathit{Erase}(N) = \mathit{Erase}(P)\langle x := \mathit{Erase}(Q)\rangle$. Thus $\mathit{Erase}(M) \rightarrow_{\mathit{Beta}} \mathit{Erase}(N)$.
  - $r = \mathit{zapp}$. Then $M = (PQ)\langle x := R\rangle$ and $N = P\langle x := R\rangle Q\langle x := R\rangle$. And therefore on one hand $\mathit{Erase}(M) = (\mathit{Erase}(P)\mathit{Erase}(Q))\langle x := \mathit{Erase}(R)\rangle$ and on the other $\mathit{Erase}(N) = \mathit{Erase}(P)\langle x := \mathit{Erase}(R)\rangle\mathit{Erase}(Q)\langle x := \mathit{Erase}(R)\rangle$ and by the rule *app* we are done.
  - $r = \mathit{zlam}$. Then $M = (\lambda y : \sigma.P)\langle x := Q\rangle$ and $N = \lambda y : \sigma.P\langle x := Q\rangle$. And therefore $\mathit{Erase}(M) = (\lambda y.\mathit{Erase}(P))\langle x := \mathit{Erase}(Q)\rangle$ and we also have $\mathit{Erase}(N) = \lambda y.\mathit{Erase}(P)\langle x := \mathit{Erase}(Q)\rangle$ and by the rule *lam* we are done.
  - $r = \mathit{var}$ or $r = \mathit{gc}$. We resolve as above.
- The reduction is internal. In all cases we use the induction hypothesis and compatibility of $\rightarrow_{\lambda\mathrm{x}}$. Also note that in the cases where the context is $P\sigma$ or $P[t := \sigma]$ only reductions in P may have taken place.

▪

Proposition 3.49 Let M be a term in $\mathcal{T}^{\rightarrow}$. If there is an infinite F_{es}-derivation starting from M, then there is an infinite λx-derivation starting from *Erase*(M).

Proof. Suppose we have an infinite F_{es}-derivation starting from M. Then since the rewrite system $S = F_{es} \setminus \lambda\mathrm{z}$ is strongly normalizing (as a consequence of Corollary 3.32) the derivation must have the form $M = M_1 \twoheadrightarrow_S M_2 \rightarrow_{\lambda\mathrm{z}} M_3 \twoheadrightarrow_S M_4 \rightarrow_{\lambda\mathrm{z}} M_5 \ldots$ where the reductions $\rightarrow_{\lambda\mathrm{z}}$ occur infinitely many times. Then by Lemma 3.48 we obtain an infinite λx-derivation starting from *Erase*(M).

▪

Next, our aim shall be to show that if M is a typable term in the polymorphic lambda calculus with explicit substitutions then the untyped λx-term *Erase*(M) is strongly λx-normalizing, thus allowing us to conclude that M is strongly F_{es}-normalizing. As already mentioned, we shall use Tait's version of Girard's candidates of reducibility technique [Tai75].

The proof of strong normalization of F_{es}

In this section we apply Tait's version of the technique of candidates of reducibility in which the candidates are untyped λx-terms.

Definition 3.50 (Substitution) A *substitution* is a function $\phi : \mathcal{V} \longrightarrow \mathcal{T}_{\lambda x}$ such that $\phi(x) \neq x$ for only finitely many $x \in \mathcal{V}$. The finite set $\{x \mid \phi(x) \neq x\}$ is called the *domain* of ϕ and is denoted $Dom(\phi)$. When ϕ has domain $Dom(\Gamma)$ for some type assignment Γ of interest we write $\phi : \Gamma \longrightarrow \mathcal{T}_{\lambda x}$.

Definition 3.51 (P-substitution) A substitution ϕ is called a *p(arallel)-substitution* if for every $x_i \in Dom(\phi)$ we have $x_i \notin \bigcup_{x \in Dom(\phi)} FV(\phi(x))$.

Definition 3.52 (Explicification of a substitution) Let ϕ be a (possibly p-) substitution and let $[i_1, \ldots, i_n]$ be any ordering of the variables in $Dom(\phi)$. Then an *explicification* $\overline{\phi}$: $\mathcal{T}_{\lambda x} \longrightarrow \mathcal{T}_{\lambda x}$ of ϕ is defined as $\overline{\phi}(M) \overset{\text{def}}{=} M\langle x_{i_1} := \phi(x_{i_1})\rangle \ldots \langle x_{i_n} := \phi(x_{i_n})\rangle$.

Intuitively a p-substitution represents a parallel substitution. Usually when one defines a substitution ϕ and applies it to a term, the substitution process takes place 'in parallel'. Since λx incorporates sequential substitution into the object-level and in order to apply the usual reducibility technique we give conditions (definition of p-substitutions) on these sequential substitutions so that they may behave as desired. It should be noted that the explicification of a substitution ϕ is not the usual notion of universal algebra that we are accustomed to, rather $\overline{\phi}$ takes a term and applies the substitution ϕ in an 'explicit' way be means of pending explicit substitutions. The notion of explicification depends on the ordering of the variables of $Dom(\phi)$ given by its user.

Definition 3.53 Let C and D be sets of untyped terms in $\mathcal{T}_{\lambda x}$. Then we define the *function space of C in D*, denoted $[C \rightarrow D]$, as $[C \rightarrow D] = \{M \in \mathcal{T}_{\lambda x} \mid \forall N \in C,\ MN \in D\}$. We refer to $[\bullet \rightarrow \bullet]$ as the *function space constructor*.

Definition 3.54 A nonempty family of sets of (untyped) terms in $\mathcal{T}_{\lambda x}$, $\mathcal{C}$, is called *type closed* if it verifies the following properties

1. every $C \in \mathcal{C}$ is nonempty,
2. $\mathcal{C}$ is closed under the function space constructor,
3. given any $\mathcal{C}$-indexed family $(A_C)_{C \in \mathcal{C}}$ of sets in $\mathcal{C}$, then $\bigcap_{C \in \mathcal{C}} A_C \in \mathcal{C}$, and
4. every $C \in \mathcal{C}$ is closed under α-equivalence.

Note that since we identify α-equivalent terms at the metalevel the last item of this definition is trivially satisfied. We shall use the notation $M \in \bigcup \mathcal{C}$ to say that M is an untyped λx-term in C for some member C of the family $\mathcal{C}$.

Definition 3.55 (Assignment) Let $\mathcal{C}$ be a type closed family. An *assignment* is a function $\eta : \mathcal{V} \longrightarrow \mathcal{C}$. Given a set $C \in \mathcal{C}$ and a type variable t we use $\eta[t := C]$ to denote the assignment such that for all $v \in \mathcal{V}$ we have $\eta[t := C](v) = C$ if $v = t$ and $\eta[t := C](v) = \eta(v)$ otherwise. Although the $\bullet[\bullet := \bullet]$ symbol has already been used for the closure operator we expect the overloading not to cause any confusion.

Definition 3.56 Given an assignment $\eta : \mathcal{V} \longrightarrow \mathcal{C}$, for every pure type σ, the set $[\![\sigma]\!]\eta$ is defined as

$$\begin{array}{lll} [\![t]\!]\eta & \overset{\text{def}}{=} & \eta(t) \ \textit{if}\ t \in \mathcal{V} \\ [\![(\sigma_1 \rightarrow \sigma_2)]\!]\eta & \overset{\text{def}}{=} & [[\![\sigma_1]\!]\eta \rightarrow [\![\sigma_2]\!]\eta] \\ [\![\forall t.\sigma]\!]\eta & \overset{\text{def}}{=} & \bigcap_{C \in \mathcal{C}} [\![\sigma]\!]\eta[t := C] \end{array}$$

In the following lemma we assume by the variable convention that the bound type variables in σ do not occur free in τ. The third item is referred to as "Girard's trick" [Gal90]. The proof may be found in [Gal90].

Lemma 3.57

1. Given two assignments $\eta_1 : \mathcal{V} \longrightarrow \mathcal{C}$ and $\eta_2 : \mathcal{V} \longrightarrow \mathcal{C}$, for every pure type σ, if η_1 and η_2 agree on $FTV(\sigma)$ then $[\![\sigma]\!]\eta_1 = [\![\sigma]\!]\eta_2$.

2. Let σ and τ be pure types. Then for every assignment $\eta : \mathcal{V} \longrightarrow \mathcal{C}$ we have $[\![\sigma\{t \leftarrow \tau\}]\!]\eta = [\![\sigma]\!]\eta[t := [\![\tau]\!]\eta]$.

3. Let $\mathcal{C}$ be a type closed family. Then for every assignment $\eta : \mathcal{V} \longrightarrow \mathcal{C}$, for every pure type σ, we have $[\![\sigma]\!]\eta \in \mathcal{C}$.

In order to prove strong normalization on typable terms we impose some conditions on the members of a type closed family. Before proceeding we recall some notation: if $M, N_1, \ldots, N_m \in \mathcal{T}_{\lambda\mathbf{x}}$ and $x_1, \ldots, x_m \in \mathcal{V}$ then we use M^* to denote the term $M\langle x_1 := N_1\rangle \ldots \langle x_m := N_m\rangle$ for $m \geq 0$.

Definition 3.58 We say that a family $\mathcal{C}$ of sets of untyped λx-terms is a *family of candidates of reducibility* iff it is type closed and satisfies the following. For every set $C \in \mathcal{C}$ we have,

R1. For every variable $x \in \mathcal{V}$, $x \in C$.

R2. For all $M, N \in \mathcal{T}_{\lambda\mathbf{x}}$, if $M\langle x := N\rangle \in C$ then $(\lambda x.M)\ N \in C$.

R3. For all $M \in \bigcup\mathcal{C}$ and for all $m \geq 0$, and $N_1, \ldots, N_m \in \bigcup\mathcal{C}$ if $M^* \in C$ then $x\langle x := M\rangle^* \in C$

R4. For all $M, N \in \bigcup\mathcal{C}$ and for every $x \in \mathcal{V}$ and all $m \geq 0$ and $N_1, \ldots, N_m \in \bigcup\mathcal{C}$ if $M^* \in C$ and $x \notin FV(M)$ then $M\langle x := N\rangle^* \in C$.

R5. For all $M_1, M_2 \in \mathcal{T}_{\lambda\mathbf{x}}$, for every $x \in \mathcal{V}$ and all $m \geq 0$, and $N, N_1, \ldots, N_m \in \bigcup\mathcal{C}$ if $(M_1\langle x := N\rangle M_2\langle x := N\rangle)^* \in C$ then $(M_1M_2)\langle x := N\rangle^* \in C$.

R6. For all $M \in \mathcal{T}_{\lambda\mathbf{x}}$, for every $x \in \mathcal{V}$, and all $m \geq 0$, and $N, N_1, \ldots, N_m \in \bigcup\mathcal{C}$, if $(\lambda y.M\langle x := N\rangle)^* \in C$ and $y \notin FV(N)$ then $(\lambda y.M)\langle x := N\rangle^* \in C$.

R7. For all $M, N \in \mathcal{T}_{\lambda\mathbf{x}}$, for all $m \geq 0$, for all $P, N_1, \ldots, N_m \in \bigcup\mathcal{C}$, if $y \notin FV(P)$ and $M\langle x := P\rangle\langle y := N\langle x := P\rangle\rangle^* \in C$ then $M\langle y := N\rangle\langle x := P\rangle^* \in C$.

Condition (R7) is the explicit version of the substitution lemma (Lemma A.4).

Now we may prove (roughly) that if a term in $\mathcal{T}$ is typable with type σ then its image via any explicification of a p-substitution ϕ (satisfying the conditions of the lemma) is in the member set of the family $\mathcal{C}$ interpreting the type σ.

Lemma 3.59 Let $\mathcal{C}$ be a family of candidates of reducibility. For every derivation of $\Gamma \rhd M : \sigma$ for some term $M \in \mathcal{T}$, for every assignment $\eta : \mathcal{V} \longrightarrow \mathcal{C}$, for every p-substitution $\phi : \Gamma \longrightarrow \mathcal{T}_{\lambda\mathbf{x}}$, if $\phi(x) \in [\![ZT(\Gamma(x))]\!]\eta$ for every $x \in Dom(\Gamma)$, then for every explicification $\overline{\phi}$ of ϕ we have $\overline{\phi}(Erase(M)) \in [\![\sigma]\!]\eta$.

Proof. By induction on the size of the derivation of $\Gamma \rhd M : \sigma$.

- Base case. Then the derivation consists solely of $\Gamma \rhd x : ZT(\rho)$ where $x : \rho \in \Gamma$. Now by hypothesis we know that $\phi(x) \in [\![ZT(\rho)]\!]\eta$. Let $[x_1, \ldots, x_n]$ be any ordering of the variables in $Dom(\Gamma)$ and suppose $x_j = x$. Then we reason as follows

$$\begin{array}{ll}
\phi(x) \in [\![ZT(\rho)]\!]\eta & \text{hypothesis} \\
\phi(x)\langle x_n := \phi(x_n)\rangle \in [\![ZT(\rho)]\!]\eta & \text{(R4), } \phi \text{ p-subs} \\
\vdots & \vdots \\
\phi(x)\langle x_{j+1} := \phi(x_{j+1})\rangle \ldots \langle x_n := \phi(x_n)\rangle \in [\![ZT(\rho)]\!]\eta & \text{(R4), } \phi \text{ p-subs} \\
x\langle x := \phi(x)\rangle\langle x_{j+1} := \phi(x_{j+1})\rangle \ldots \langle x_n := \phi(x_n)\rangle \in [\![ZT(\rho)]\!]\eta & \text{(R3)} \\
\vdots & \vdots \\
x\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle \in [\![ZT(\rho)]\!]\eta & \text{(R4), } \Gamma \text{ type judg.}
\end{array}$$

- Inductive case. Let $k + 1$ be the length of this derivation. We analyse the last derivation rule applied,
  - *app*. Thus the derivation ends as follows,

$$\frac{\Gamma \rhd M : \tau \to \sigma \quad \Gamma \rhd N : \tau}{\Gamma \rhd MN : \sigma}\ app$$

Then by the induction hypothesis we have that $\overline{\phi}(\mathit{Erase}(M)) \in [\![\tau \to \sigma]\!]\eta$ and also that $\overline{\phi}(\mathit{Erase}(N)) \in [\![\tau]\!]\eta$ for any explicification $\overline{\phi}$ of ϕ. By the definition of $[\![\tau \to \sigma]\!]\eta$ we have $\overline{\phi}(\mathit{Erase}(M))\overline{\phi}(\mathit{Erase}(N)) \in [\![\sigma]\!]\eta$. That is,
$\mathit{Erase}(M)\langle x_1 := \phi(x_1)\rangle..\langle x_n := \phi(x_n)\rangle \mathit{Erase}(N)\langle x_1 := \phi(x_1)\rangle..\langle x_n := \phi(x_n)\rangle \in [\![\sigma]\!]\eta$. Then applying condition (R5) repeatedly we obtain,

$$\begin{array}{ll} & (\mathit{Erase}(M)\mathit{Erase}(N))\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle \\ = & \mathit{Erase}(MN)\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle \\ \in & [\![\sigma]\!]\eta \end{array}$$

Since this applies for any explicification $\overline{\phi}$ of ϕ meeting the conditions of the lemma we are done.

- *abs*. Then the derivation ends as follows:

$$\frac{\Gamma, x : \tau \rhd M : \rho}{\Gamma \rhd \lambda x : \tau.M : ZT(\tau) \to \rho}\ \mathit{abs}$$

Let N be any term in $[\![ZT(\tau)]\!]\eta$ and $\overline{\phi}$ any explicification of ϕ with ordering of variables $[x_1, \ldots, x_n]$. We choose a representative in the α-equivalence class $[(\lambda x.\mathit{Erase}(M))\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle]_\alpha$, say $(\lambda x'.\mathit{Erase}(M'))\langle x'_1 := \phi(x_1)\rangle \ldots \langle x'_n := \phi(x_n)\rangle$ (note that renaming commutes with the erasing function) such that

1. $x' \notin FV(\phi(x_i))$ for $i \in 1..n$ and $x' \notin FV(N)$ and $x_i \neq x'$ for $i \in 1..n$, and
2. $x'_i \notin FV(\phi(x_j))$ and $x'_i \notin FV(N)$, for $i, j \in 1..n$.

We have to rename x in order to avoid name clashes (first item). But since we are working with p-substitutions this entails more renaming (second item) so that we may be able to apply the induction hypothesis when it is needed. Note that since ϕ is a p-substitution no renaming takes place inside the $\phi(x_i)$s in the representative of the α-equivalence class chosen.

Let ϕ' be the resulting p-substitution, i.e. the substitution defined as

$$\phi'(z) \overset{\text{def}}{=} \begin{cases} \phi(x_i) & if\ z = x'_i \\ N & if\ z = x' \end{cases}$$

Since the induction hypothesis holds for every derivation of length $\leq k$ and for every assignment η satisfying the conditions of the lemma, we have that for every explicification $\overline{\phi'}$ of ϕ', $\overline{\phi'}(\mathit{Erase}(M')) \in [\![\rho]\!]\eta$. In particular, $\mathit{Erase}(M')\langle x'_1 := \phi(x_1)\rangle \ldots \langle x'_n := \phi(x_n)\rangle\langle x' := N\rangle \in [\![\rho]\!]\eta$.

Then by condition (R2) we obtain that $(\lambda x'.\mathit{Erase}(M')\langle x'_1 := \phi(x_1)\rangle \ldots \langle x'_n := \phi(x_n)\rangle)N \in [\![\rho]\!]\eta$

Since this is valid for every $N \in [\![ZT(\tau)]\!]\eta$ then by definition of $[\![ZT(\tau) \to \rho]\!]\eta$ we have that, $\lambda x'.\mathit{Erase}(M')\langle x'_1 := \phi(x_1)\rangle \ldots \langle x'_n := \phi(x_n)\rangle \in [\![ZT(\tau) \to \rho]\!]\eta$

Finally since $x' \notin FV(\phi(x_i))$ for $i \in 1..n$ (first item above) we may apply condition (R6) repeatedly obtaining, $(\lambda x'.\mathit{Erase}(M'))\langle x'_1 := \phi(x_1)\rangle \ldots \langle x'_n := \phi(x_n)\rangle \in [\![ZT(\tau) \to \rho]\!]\eta$

And by closure under α-equivalence we conclude,

$$\begin{array}{ll} & (\lambda x.\mathit{Erase}(M))\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle \\ = & \mathit{Erase}(\lambda x : \tau.M)\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle \\ = & \overline{\phi}(\mathit{Erase}(\lambda x : \tau.M)) \\ \in & [\![ZT(\tau) \to \rho]\!]\eta \end{array}$$

- *subs*. Then the derivation ends as follows

$$\frac{\Gamma, x : \tau \rhd M_1 : \sigma \quad \Gamma \rhd M_2 : ZT(\tau)}{\Gamma \rhd M_1\langle x := M_2\rangle : \sigma}\ \mathit{subs}$$

Let ϕ be any p-substitution with domain Γ satisfying the conditions of the lemma. Now we choose a representative of the equivalence class
$[\mathit{Erase}(M_1)\langle x := \mathit{Erase}(M_2)\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle\rangle]_\alpha$
say $\mathit{Erase}(M'_1)\langle x' := \mathit{Erase}(M_2)\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle\rangle$, such that

1. $x' \notin \mathit{Dom}(\Gamma)$, and

2. $x' \notin FV(\overline{\phi}(\mathit{Erase}(M_2)))$ (and thus $x' \notin FV(\phi(x_j))$ with $j \in 1..n$).

Let ϕ' be the substitution $\phi' : (\Gamma, x' : \tau) \longrightarrow \mathcal{T}_{\lambda\mathbf{x}}$ defined as

$$\phi'(z) \stackrel{\text{def}}{=} \begin{cases} \phi(x_i) & \mathit{if}\ z = x_i \\ \overline{\phi}(\mathit{Erase}(M_2)) & \mathit{if}\ z = x' \end{cases}$$

Note that due to the first and second item, and that $x_i \notin FV(\overline{\phi}(\mathit{Erase}(M_2))$ with $i \in 1..n$ (since they are all bound and ϕ is a p-substitution) the resulting substitution is a p-substitution.

Since the induction hypothesis holds for any derivation of length $\leq k$ and for every assignment satisfying the conditions of the lemma, on one hand for every explicification $\overline{\phi}$ of ϕ we have $\overline{\phi}(\mathit{Erase}(M_2)) \in \llbracket ZT(\tau)\rrbracket\eta$. Another application of the induction hypothesis to the derivation ending in the left premise of the application of *subs* allows us to infer that for every explicification $\overline{\phi'}$ of ϕ' we have $\overline{\phi'}(\mathit{Erase}(M_1')) \in \llbracket\sigma\rrbracket\eta$. In particular we have,

$\mathit{Erase}(M_1')\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle\langle x' := \mathit{Erase}(M_2)\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle\rangle \in \llbracket\sigma\rrbracket\eta$.

Since $x' \notin FV(\phi(x_i))$ (item 2) we may apply condition (R7) repeatedly and obtain $\mathit{Erase}(M_1')\langle x' := \mathit{Erase}(M_2)\rangle\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle \in \llbracket\sigma\rrbracket\eta$.

And since $\llbracket\sigma\rrbracket\eta$ is closed under α-equivalence

$$\begin{array}{rl} & \mathit{Erase}(M_1)\langle x := \mathit{Erase}(M_2)\rangle\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle \\ = & \mathit{Erase}(M_1\langle x := M_2\rangle)\langle x_1 := \phi(x_1)\rangle \ldots \langle x_n := \phi(x_n)\rangle \\ = & \overline{\phi}(\mathit{Erase}(M_1\langle x := M_2\rangle)) \in \llbracket\sigma\rrbracket\eta \end{array}$$

– *tapp*. Then the derivation ends as follows

$$\frac{\Gamma \rhd M : \forall t.\rho}{\Gamma \rhd M\tau : \rho\{t \leftarrow ZT(\tau)\}}\ \mathit{tapp}$$

By the induction hypothesis, for any explicification $\overline{\phi}$ of ϕ we have $\overline{\phi}(\mathit{Erase}(M)) \in \llbracket\forall t.\rho\rrbracket\eta$. Now since $\llbracket\forall t.\rho\rrbracket\eta = \bigcap_{C \in \mathcal{C}}\llbracket\rho\rrbracket\eta[t := C]$, we have $\overline{\phi}(\mathit{Erase}(M)) \in \llbracket\rho\rrbracket\eta[t := C]$ for every $C \in \mathcal{C}$. In particular if we take $C = \llbracket ZT(\tau)\rrbracket\eta \in \mathcal{C}$, we have $\overline{\phi}(\mathit{Erase}(M)) \in \llbracket\rho\rrbracket\eta[t := \llbracket ZT(\tau)\rrbracket\eta]$. By Lemma 3.57(2) we have that $\llbracket\rho\{t \leftarrow ZT(\tau)\}\rrbracket\eta = \llbracket\rho\rrbracket\eta[t := \llbracket ZT(\tau)\rrbracket\eta]$, and therefore $\overline{\phi}(\mathit{Erase}(M\tau)) = \overline{\phi}(\mathit{Erase}(M)) \in \llbracket\rho\{t \leftarrow ZT(\tau)\}\rrbracket\eta$.

– *tabs*. The derivation ends as follows.

$$\frac{\Gamma \rhd M : \rho}{\Gamma \rhd (\Lambda t.M) : \forall t.\rho}\ \mathit{tabs}$$

where $t \notin FTV(\Gamma(x))$ with $x \in \mathit{Dom}(\Gamma)$.

Then $t \notin FTV(ZT(\Gamma(x)))$ for every $x \in \mathit{Dom}(\Gamma)$ (since if $\tau \twoheadrightarrow_{ZT} \tau'$ then $FTV(\tau') \subseteq FTV(\tau)$) and by Lemma 3.57(1) we have $\llbracket ZT(\Gamma(x))\rrbracket\eta = \llbracket ZT(\Gamma(x))\rrbracket\eta[t := C]$ for every $C \in \mathcal{C}$. Since the induction hypothesis holds for every derivation of length $\leq k$, for every η, and for every p-substitution ϕ satisfying the conditions of the lemma, it holds for every $C \in \mathcal{C}$ when applied to the derivation $\Gamma \rhd M : \rho$, to every $\eta[t := C]$ and to every p-substitution ϕ such that $\phi(x) \in \llbracket ZT(\Gamma(x))\rrbracket\eta$. Therefore, for every $C \in \mathcal{C}$ and for every explicification $\overline{\phi}$ of ϕ we have $\overline{\phi}(\mathit{Erase}(M)) \in \llbracket\rho\rrbracket\eta[t := C]$. And thus $\overline{\phi}(\mathit{Erase}(\Lambda t.M)) = \overline{\phi}(\mathit{Erase}(M)) \in \bigcap_{C \in \mathcal{C}}\llbracket\rho\rrbracket\eta[t := C] = \llbracket\forall t.\rho\rrbracket\eta$.

– *tsubs*. The derivation ends as follows.

$$\frac{\Delta \rhd P : \rho}{\Delta_{[t:=\tau]} \rhd P[t := \tau] : \rho\{t \leftarrow ZT(\tau)\}}\ \mathit{tsubs}$$

Let η be any assignment and ϕ any p-substitution with domain $\Delta_{[t:=\tau]}$ meeting the conditions of the lemma. Consider any $x : \sigma_1 \in \Delta_{[t:=\tau]}$ and its corresponding $x : \sigma_2 \in \Delta$. Then by hypothesis we have $\phi(x) \in \llbracket ZT(\sigma_1)\rrbracket\eta$ and since $\sigma_1 =_{ZT} \sigma_2[t := \tau]$ then $ZT(\sigma_1) = ZT(\sigma_2)\{t \leftarrow ZT(\tau)\}$ using the fact that ZT is complete and Lemma 3.29(2). Thus $\llbracket ZT(\sigma_1)\rrbracket\eta = \llbracket ZT(\sigma_2)\{t \leftarrow ZT(\tau)\}\rrbracket\eta =_{L\ 3.57(2)} \llbracket ZT(\sigma_2)\rrbracket\eta[t := \llbracket ZT(\tau)\rrbracket\eta]$.

We apply then the induction hypothesis with p-substitution ϕ and assignment $\eta[t := \llbracket ZT(\tau)\rrbracket\eta]$ and obtain that for every explicification $\overline{\phi}$ of ϕ we have $\overline{\phi}(\mathit{Erase}(P)) \in \llbracket\rho\rrbracket\eta[t := \llbracket ZT(\tau)\rrbracket\eta]$. But since $\mathit{Erase}(P) = \mathit{Erase}(P[t := \tau])$ and by Lemmas 3.57(2) and 3.29(2) we may conclude that $\overline{\phi}(\mathit{Erase}(P[t := \tau])) \in \llbracket\rho\{t \leftarrow ZT(\tau)\}\rrbracket\eta$.

■

In order to obtain families of candidates of reducibility for proving properties of *Erase*(M) where $M \in \mathcal{T}$ Girard has shown that conditions R1-R7 must be strengthened.

Definition 3.60 (Saturated set) Let S be a nonempty set of terms in $\mathcal{T}_{\lambda x}$.

1. We say that S is *closed* iff whenever $Mx \in S$ then $M \in S$ where $x \in \mathcal{V}$.
2. Let S be closed. A subset $C \subseteq S$ which is closed under α-equivalence is called *saturated* iff the following conditions hold

S1. For every variable $x \in \mathcal{V}$, for all $n \geq 0$ and all $P_1 \ldots P_n \in S$, $xP_1 \ldots P_n \in C$.

S2. For all $M, N \in \mathcal{T}_{\lambda x}$, all $n \geq 0$ and all $P_1, \ldots, P_n \in S$, if $M\langle x := N\rangle P_1 \ldots P_n \in C$ then $(\lambda x.M)\ NP_1 \ldots P_n \in C$.

S3. For all $M \in S$ and for all $m \geq 0$,$n \geq 0$ and all $N_1, \ldots, N_m \in S$ and $P_1, \ldots, P_n \in S$ if $M^*P_1 \ldots P_n \in C$ then $x\langle x := M\rangle^* P_1 \ldots P_n \in C$.

S4. For all $M, N \in S$, all $m \geq 0$, $n \geq 0$, all $N_1, \ldots, N_m$, all $P_1, \ldots, P_n \in S$, and every variable x in $\mathcal{V}$ such that $x \notin FV(M)$ if $M^*P_1 \ldots P_n \in C$ then $M\langle x := N\rangle^* P_1 \ldots P_n \in C$.

S5. For all $M_1, M_2 \in \mathcal{T}_{\lambda x}$, for every variable $x \in \mathcal{V}$ and for all $m \geq 0$, and $N, N_1, \ldots, N_m \in S$ and all $n \geq 0$ and $P_1, \ldots, P_n \in S$ if $(M_1\langle x := N\rangle M_2\langle x := N\rangle)^* P_1 \ldots P_n \in C$ then $(M_1M_2)\langle x := N\rangle^* P_1 \ldots P_n \in C$.

S6. For all $M \in \mathcal{T}_{\lambda x}$, for every $x \in \mathcal{V}$, and all $m \geq 0$, and $N, N_1, \ldots, N_m \in S$, and all $n \geq 0$ and $P_1, \ldots, P_n \in S$ if $(\lambda y.M\langle x := N\rangle)^* P_1 \ldots P_n \in C$ then $(\lambda y.M)\langle x := N\rangle^* P_1 \ldots P_n \in C$.

S7. For all $M, N \in \mathcal{T}_{\lambda x}$, for all $m \geq 0$, for all $P, N_1, \ldots, N_m \in S$, for all $n \geq 0$ and $P_1, \ldots, P_n \in S$ if $y \notin FV(P)$ and $M\langle x := P\rangle\langle y := N\langle x := P\rangle\rangle^* P_1 \ldots P_n \in C$ then $M\langle y := N\rangle\langle x := P\rangle^* P_1 \ldots P_n \in C$.

Lemma 3.61 Let S be a nonempty closed set of terms in $\mathcal{T}_{\lambda x}$, let $\mathcal{C}$ be the family of saturated subsets of S, and assume that S is saturated. Then $\mathcal{C}$ is a family of candidates of reducibility.

Proof. Note that conditions (S1)-(S7) imply conditions (R1)-(R7), respectively. Therefore we are left to verify that $\mathcal{C}$ is a type closed family. First of all note that the family $\mathcal{C}$ is nonempty since $S \in \mathcal{C}$. Also,

- By condition (S1) each saturated subset is nonempty since it contains all variables.
- Let C and D be saturated subsets of S. We must show that $[C \rightarrow D] = \{M \in \mathcal{T}_{\lambda x} \mid \forall N \in C,\ MN \in D\}$ is a saturated subset of S.
  - $[C \rightarrow D]$ is a subset of S. Let $M \in [C \rightarrow D]$. Since there is some variable $x \in C$, we have $Mx \in D$. And since S is closed, $M \in S$.
  - (S1). Since D is a saturated subset of S then by (S1) for every variable $x \in \mathcal{V}$ and for all $n \geq 0$ and $P_1, \ldots, P_n, P \in S$ we have $xP_1 \ldots P_nP \in D$. Since this is valid for every $P \in C$, we have $xP_1 \ldots P_n \in [C \rightarrow D]$.
  - (S2). Let M and N be any terms in $\mathcal{T}_{\lambda x}$. Let $P_1 \ldots P_n$ be terms in S. Suppose that $M\langle x := N\rangle P_1 \ldots P_n \in [C \rightarrow D]$. Then for every $P \in C$ we have $M\langle x := N\rangle P_1 \ldots P_nP \in D$. Since D is saturated, by (S2) we have $(\lambda x.M)\ NP_1 \ldots P_nP \in D$. Since this holds for every $P \in C$, $(\lambda x.M)\ NP_1 \ldots P_n \in [C \rightarrow D]$.
  - (S3). Let M be a term in S. Let $N_1, \ldots, N_m$ and $P_1 \ldots P_n$ be terms in S. Suppose that $M\langle x_1 := N_1\rangle \ldots \langle x_m := N_m\rangle P_1 \ldots P_n \in [C \rightarrow D]$. Then for every $P \in C$ we have $M\langle x_1 := N_1\rangle \ldots \langle x_m := N_m\rangle P_1 \ldots P_nP \in D$ and since D is a saturated subset of S by (S3) $x\langle x := M\rangle\langle x_1 := N_1\rangle \ldots \langle x_m := N_m\rangle P_1 \ldots P_nP \in D$. Since this holds for every $P \in C$ we have $x\langle x := M\rangle\langle x_1 := N_1\rangle \ldots \langle x_m := N_m\rangle P_1 \ldots P_n \in [C \rightarrow D]$.

  The remaining cases are dealt with likewise.
- Note that properties (S1)-(S7) of saturated subsets of S are preserved under arbitrary intersections, and thus for the $\mathcal{C}$-indexed family of saturated subsets of S, $(A_C)_{C \in \mathcal{C}}$ we have $\bigcap_{C \in \mathcal{C}} A_C$ is a saturated subset of S.

- A saturated subset of S is closed under α-equivalence by definition.

∎

Proposition 3.62 Let S be a nonempty closed set of terms in $\mathcal{T}_{\lambda x}$, let $\mathcal{C}$ be the family of all saturated subsets of S, and assume that S is saturated. Let $M \in \mathcal{T}$ that type checks under the type assignment $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ and let $z_1, \ldots, z_n$ be fresh variables. Then $\mathit{Erase}(M)\langle x_{i_1} := z_1\rangle \ldots \langle x_{i_n} := z_n\rangle \in S$ where $i_1, \ldots, i_n$ is any ordering of the variables in $\mathit{Dom}(\Gamma)$.

Proof. By Lemma 3.61 we have that $\mathcal{C}$ is a family of candidates of reducibility. Finally, we conclude by Lemma 3.57(3) and Lemma 3.59. ∎

Therefore in order to prove properties of $\mathit{Erase}(M)$ when M is a typable polymorphic term we need saturated sets satisfying these properties. In particular, since we are interested in strongly λx-normalizing terms ($SN_{\lambda x}$) we shall prove that the set $SN_{\lambda x}$ is a closed saturated set. The key ingredient in the proof of this result is the Perpetuality Proposition.

Lemma 3.63 The set $SN_{\lambda x}$ is a closed saturated set.

Proof. We must verify that $SN_{\lambda x}$ is closed and verifies (S1)-(S7). Note that if Mx is strongly λx-normalizable then M also, so $SN_{\lambda x}$ is closed. We are left to check that $SN_{\lambda x}$ verifies

- (S1). Suppose $P_1 \ldots P_n \in SN_{\lambda x}$ then for every variable $x \in \mathcal{V}$, $xP_1 \ldots P_n$ is strongly λx-normalizing.
- (S2). By the Perpetuality Proposition 3.11(1).
- (S3)-(S6). By the Perpetuality Proposition 3.11(2),(3).
- (S7). By Remark 3.12.

∎

Therefore since $SN_{\lambda x}$ is a nonempty closed subset of $\mathcal{T}_{\lambda x}$ and is itself saturated, if we construct the family of saturated subsets of $SN_{\lambda x}$ we obtain, by Lemma 3.61, a family of candidates of reducibility. This allows us to prove the following corollary.

Corollary 3.64 Let M be a term in $\mathcal{T}^{\rightarrow}$. Then M is strongly F_{es}-normalizing.

Proof. Suppose $M \in \mathcal{T}^{\rightarrow}$. Then there is a type assignment $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ and a pure type σ such that $\Gamma \rhd M : \sigma$ is a derivable type judgement.

Let $z_1, \ldots, z_n$ be fresh variables. Then by Proposition 3.62 we have $\mathit{Erase}(M)\langle x_{i_1} := z_1\rangle \ldots \langle x_{i_n} := z_n\rangle \in SN_{\lambda x}$ where $i_1, \ldots, i_n$ is any ordering of the variables in $\mathit{Dom}(\Gamma)$. But then we also have $\mathit{Erase}(M) \in SN_{\lambda x}$ and finally by Proposition 3.49 we may conclude that M is strongly F_{es}-normalizing. ∎

Chapter 4

Perpetuality in $\lambda\mathtt{ws}$

This chapter studies perpetuality in the $\lambda\mathtt{ws}$-calculus. It does so by exploiting the proof technique introduced in [DG99, DG01] for proving preservation of strong normalization of $\lambda\mathtt{ws}$. The $\lambda\mathtt{ws}$-calculus is a calculus of explicit substitutions based on de Bruijn indices notation which is considerably more involved than $\lambda\mathtt{x}$ due to the presence of substitution composition. Indeed, although in contrast to the case of $\lambda\mathtt{x}$-terms the $\lambda\mathtt{ws}$-terms are decorated with updating tags (Def. 2.53), it is the presence of rewrite rules allowing substitutions to be composed which introduces technical difficulties which are not present in $\lambda\mathtt{x}$. A term with an update tag is of the form $\langle i\rangle M$, indicating that all free indices in M should be incremented by i units. Substitutions have the form $M[i/N,j]$ indicating that indices i must be replaced by $\langle i\rangle N$ and that there was a tag $\langle j\rangle$ embracing the *Beta*-redex that fired the substitution. The following rewrite step illustrates how substitutions may be composed in $\lambda\mathtt{ws}$:

$$4\underbrace{[0/00,0]}_{s}\underbrace{[0/\lambda 00,0]}_{t} \rightarrow_{\lambda\mathtt{ws}} 4\underbrace{[0/(00)[0/\lambda 00,0],1]}_{s o t}$$

Note that, as originally introduced in [DG99, DG01], de Bruijn indices start from 0 instead of 1 as introduced in Section 2.2.2. There are no updating tags in the above two terms. The example below shall include updating tags.

The $\lambda\mathtt{ws}$-calculus is the first[1] lambda calculus with explicit substitutions to satisfy simulation of one-step β-rewrite reduction[2], confluence on open terms and preservation of strong normalization. It is often mentioned that an advantage of calculi of explicit substitutions is that substitutions may be executed in a controlled manner. They may be delayed, for instance, in order to avoid unwanted duplication of the body of the substitution. However, in such a case all substitutions 'above' the delayed instance are blocked. This is witnessed in the $\lambda\mathtt{x}$-calculus for example. The interest in $\lambda\mathtt{ws}$ is that these substitutions 'above' the delayed one may jump (provided certain conditions are fulfilled) over it thanks to rewrite rules for substitution composition. This constitutes an interesting scenario for the study of rewrite strategies.

As already mentioned in Chapter 3 the literature on perpetual rewrite strategies requires orthogonality, a property which is not fulfilled by $\lambda\mathtt{ws}$. Hence these results do not apply. Furthermore, the closure tracing technique used in Chapter 3 is not applicable either due to the presence of substitution composition. Indeed, although as in $\lambda\mathtt{x}$, rewrite steps in infinite $\lambda\mathtt{ws}$-derivations must be void (Def. 2.38) from some point on, the substitution body which is source of an infinite $\lambda\mathtt{ws}$-derivation may have been created by composing substitutions. An example follows.

$$4[0/00,0][0/\lambda(00),0] \rightarrow_{\lambda\mathtt{ws}} 4[0/(00)[0/\lambda 00,0],1] \twoheadrightarrow_{\mathtt{ws}} 4[0/\langle 0\rangle(\lambda 00)\ \langle 0\rangle(\lambda 00),1] \twoheadrightarrow_{\lambda\mathtt{ws}} \cdots$$

The substitution body $(00)[0/\lambda 00,0]$ shall be the source of an infinite $\lambda\mathtt{ws}$-derivation. It does not seem possible to trace back this body and make use of the minimality argument.

Recently, an extension of uniform normalization (all redexes are perpetual) to non-orthogonal systems was presented in [KOO01b]. Due to the fact that decent terms are not preserved by $\mathtt{ws}$-reduction (see Section 4.2.4) the technique developed in that work does not seem to be directly applicable either.

[1] Together, it seems [DG01], with the work by H.Goguen and J.Goubault-Larrecq [GGL00].

[2] However, see Section 4.2.

In this chapter we define a perpetual rewrite strategy for λws and use it to prove that an inductive characterization of a class of terms in λws captures exactly those that are strongly λws-normalizing. The results build on the so called *constricting*, and in particular *zoom-in* strategies of the lambda calculus [Gra96, RSSX99, Mel96, KOO01a]. The strategy may be summarized as follows:

- Step 1. Given a λws-term M admitting an infinite λws-derivation we obtain a subterm N of M still having an infinite λws-derivation, but such that every strict subterm of N is strongly λws-normalizing. Thus N is a 'minimal' subterm of M admitting an infinite λws-derivation.
- Step 2. The resulting term N may have a *Beta*-redex as head redex[3] or a ws-redex, where ws is the substitution calculus of λws. Now in the former case the *Beta*-redex is easily seen to be perpetual, that is to say, contracting this redex does not result in a term which is strongly λws-normalizing. However in the latter case it is not clear that the head ws-redex (or any ws-redex for that matter) is perpetual. This problem is due to the non-orthogonality of λws. We solve it by introducing a *labelling strategy*, described briefly in Step 3.
- Step 3. Given a minimal (in the sense discussed above) λws-term N admitting an infinite λws-derivation, the labelling strategy labels an explicit substitution operator in N, yielding $\underline{N}$. The resulting labelled substitution operator enjoys the property that it may be eliminated from $\underline{N}$ (by computing it) via a notion of labelled rewriting that preserves the possibility of infinite λws-derivations. The proof that this notion of labelled rewriting is perpetual relies on the proof technique used for proving preservation of strong normalization of λws [DG99, DG01].

The perpetual rewrite strategy shall then be used to show that a certain set of terms determined by four inductive rule schemas are exactly the set of strongly λws-normalizing terms.

Structure of the chapter

Due to the technical complications arising in λws, partly because of the 'noise' introduced by the updating tags, but more fundamentally due to the presence of substitution composition, we have chosen to present the basic ideas in the context of the λx-calculus. Of course, substitutions may not be composed in λx, however we believe that the effort is nonetheless worthwhile from an expository point of view. It helps pinpoint the main steps outlined above, and prepares the reader for the material on λws that follows. After introducing the labelled λws-calculus we study the labelling strategy for λws. This shall constitute the core of the chapter. Finally, we formulate the rewrite strategy and prove it perpetual. We end the chapter with the characterization of the strongly λws-normalizing terms.

The material presented in this chapter is joint work with A.Arbiser and A.Ríos [ABR00].

4.1 Zooming-in on λx

In this section we formulate a perpetual zoom-in λx-strategy, in preparation for Section 4.2 on λws.

Definition 4.1 Let $M \in \mathcal{T}_{\lambda x}$ such that $\infty_{\lambda x}(M)$. A subterm N of M such that $\infty_{\lambda x}(N)$ and every proper subterm of N is strongly λx-normalizing, is called a *minimal perpetual subterm*.

An infinite λx-derivation is called *constricting* [Gra96, RSSX99] if it is of the form

$$C_1[M_1] \rightarrow_{\lambda x} C_1[C_2[M_2]] \rightarrow_{\lambda x} C_1[C_2[C_3[M_3]]] \rightarrow_{\lambda x} \dots$$

where the M_i are minimal perpetual subterms and the redex contracted in the step $C_1[\dots C_i[M_i]\dots] \rightarrow_{\lambda x} C_1[\dots C_i[C_{i+1}[M_{i+1}]]\dots]$ is a subterm of M_i. A (one-step) perpetual rewrite strategy $\mathcal{F}(\bullet)$ for λx is called constricting if any infinite $\mathcal{F}(\bullet)$-derivation $M \rightarrow_{\lambda x} \mathcal{F}(M) \rightarrow_{\lambda x} \mathcal{F}(\mathcal{F}(M)) \rightarrow_{\lambda x} \dots$ is constricting. In the case that $\mathcal{F}(\bullet)$ is a many-step rewrite strategy then it is constricting if any infinite $\mathcal{F}(\bullet)$-derivation $M \xrightarrow{+}_{\lambda x} \mathcal{F}(M) \xrightarrow{+}_{\lambda x} \mathcal{F}(\mathcal{F}(M)) \xrightarrow{+}_{\lambda x} \dots$ is constricting. A *zoom-in* rewrite strategy is a constricting strategy which in

[3] Actually, there are two '*Beta*' rules in λws however at the moment it is the intuitive grasp we are seeking so we shall ignore this issue for the time being.

each term contracts the leftmost redex of a minimal subterm with an infinite derivation. The strategies $\mathcal{F}_{\infty}(\bullet)$ and $\mathcal{F}(\bullet)$ of Def. 3.16 and 3.19, respectively, are not zoom-in strategies since in $M = (\lambda x.x)(\Delta\Delta)$ they contract the leftmost *Beta*-redex (yielding $x\langle x := \Delta\Delta\rangle$). However, this redex is not the leftmost redex of a minimal subterm of M admitting an infinite λx-derivation since $\Delta\Delta$ is a subterm of M with this property. For the same reason, neither of the strategies are constricting either.

The chapter aims at introducing a constricting strategy which is partly zoom-in in the sense that in a term M, if the leftmost redex of a minimal subterm N with an infinite derivation is a *Beta*-redex, then it contracts it. If not, then the leftmost λx-redex of this subterm is an x-redex. At this point we need some indication of which redex to contract. For this we introduce a *labelling strategy* which amounts to a function which selects an x-redex in N to compute. By compute we mean label or mark the x-redex in N and then contract, repeatedly, all marked x-redexes until none are left. The whole process shall yield a many-step λx-rewrite strategy.

Of course, from the Perpetuality Proposition 3.11 we know that contracting any x-redex (and in particular the leftmost one) in N shall yield a full perpetual zoom-in strategy. However, this result does not hold for λws, but rather a more restricted one (Corollary 4.40). Therefore, since this section's aim is to introduce the work done for λws in the context of λx in order to get a grip of the main ideas, we shall use the labelling strategy also for λx, as discussed above.

The reader not familiar with λx is referred to Section 2.3.1 of Chapter 2.

4.1.1 The Labelled Substitution Calculus

We shall begin by defining labelled terms and labelled λx-rewriting. Labels shall allow us to mark x-redexes, *Beta*-redexes shall not be labelled. Compare the definition of labelled terms below with that of the λx-terms, denoted $\mathcal{T}_{\lambda x}$ (Def. 2.29).

Definition 4.2 (Labelled λx-terms) The labelled λx-terms, denoted $\mathcal{T}_{\underline{\lambda x}}$, are given by the following grammar:

$$M \quad ::= \quad x \mid \lambda x.M \mid MM \mid M\langle x := M\rangle \mid M\langle\!\langle x := N\rangle\!\rangle \text{ where } N \in \mathcal{T}_{\lambda x}$$

The $\bullet\langle\!\langle \bullet := \bullet\rangle\!\rangle$ operator is called the *labelled substitution operator*. Note that the body of a labelled substitution operator is a λx-term (and thus contains no occurrences of the labelled substitution operator). The set of free variables of a term M is denoted $FV(M)$ and defined as usual. In the sequel we shall refer to $\bullet\langle \bullet := \bullet\rangle$ as the *unlabelled substitution operator*.

So for instance $x\langle\!\langle x := y\rangle\!\rangle\langle\!\langle y := z\rangle\!\rangle$ is a valid labelled λx-term, however $x\langle\!\langle x := y\langle\!\langle y := z\rangle\!\rangle\rangle\!\rangle$ is not since the term $y\langle\!\langle y := z\rangle\!\rangle$ is not a λx-term. The labels shall allow us to trace the computation of substitutions. We now present the rewrite rules that compute labelled substitutions.

Definition 4.3 (The $\underline{\mathrm{x}}$, $\underline{\mathrm{xc}}$ and $\underline{\lambda\mathrm{x}}$-rewrite systems) The $\underline{\lambda\mathrm{x}}$-rewrite system is given by the λx-rewrite system together with the $\underline{\mathrm{x}}$-rewrite system. The $\underline{\mathrm{x}}$-rewrite system is defined by the following rewrite rules:

$$\begin{array}{llll}
(MN)\langle\!\langle x := P\rangle\!\rangle & \rightarrow_{lApp} & M\langle\!\langle x := P\rangle\!\rangle N\langle\!\langle x := P\rangle\!\rangle & \\
(\lambda y.(M))\langle\!\langle x := P\rangle\!\rangle & \rightarrow_{lLam} & \lambda y.(M\langle\!\langle x := P\rangle\!\rangle) & \\
x\langle\!\langle x := P\rangle\!\rangle & \rightarrow_{lVar} & P & \\
M\langle\!\langle x := P\rangle\!\rangle & \rightarrow_{lGc} & M & x \notin FV(M)
\end{array}$$

Reduction in the $\underline{\lambda\mathrm{x}}$-rewrite system is called *labelled reduction*. The $\underline{\mathrm{xc}}$-rewrite system is the $\underline{\mathrm{x}}$-rewrite system together with the following *composition rule*:

$$P\langle x := Q\rangle\langle\!\langle y := R\rangle\!\rangle \rightarrow_c P\langle\!\langle y := R\rangle\!\rangle\langle x := Q\langle\!\langle y := R\rangle\!\rangle\rangle$$

Note that the c-rewrite rule allows labelled substitutions to jump over unlabelled substitutions. However, no jumping is allowed between substitutions of the same kind, i.e. labelled or not.

Lemma 4.4 (Properties of $\underline{\mathrm{xc}}$) The $\underline{\mathrm{xc}}$-rewrite system is strongly normalizing and confluent. The set of $\underline{\mathrm{xc}}$-normal forms is $\mathcal{T}_{\lambda x}$.

Proof. Confluence is a consequence of strong normalization and local confluence of $\underline{\mathrm{xc}}$ (all three critical pairs are joinable), by applying Newman's Lemma (Lemma 2.4). That the set of $\underline{\mathrm{xc}}$-normal forms is $\mathcal{T}_{\lambda x}$ follows from a close study of the rewrite rules of $\underline{\mathrm{xc}}$.

We now prove strong normalization by means of a translation to a set of ground terms equipped with a well-founded ordering obtained by the recursive path ordering (RPO) method [Der82]. Consider the alphabet $\{\star, @, \bullet\langle\bullet\rangle, \bullet\langle\!\langle\bullet\rangle\!\rangle, \lambda(\bullet)\}$ and the terms S over this alphabet defined as $a ::= \star \mid @(a,a) \mid \lambda(a) \mid a\langle a\rangle \mid a\langle\!\langle a\rangle\!\rangle$. Then the well-founded precedence $\langle\!\langle\rangle\!\rangle \gg \langle\rangle \gg @, \lambda(), \star$ induces a well-founded order on S, denoted $\succ_{\mathcal{T}_l}$, as dictated by the RPO method (where all function symbols are assigned multiset status). Finally we show, by induction on M, that if $M \to_{\underline{\mathtt{xc}}} N$ then $T(M) \succ_{\mathcal{T}_l} T(N)$ where T is the translation defined as follows:

$$\begin{array}{lcl}
T(x) & \stackrel{\text{def}}{=} & \star \\
T(MN) & \stackrel{\text{def}}{=} & @(T(M), T(N)) \\
T(\lambda x.M) & \stackrel{\text{def}}{=} & \lambda(T(M)) \\
T(M\langle x := N\rangle) & \stackrel{\text{def}}{=} & T(M)\langle T(N)\rangle \\
T(M\langle\!\langle x := N\rangle\!\rangle) & \stackrel{\text{def}}{=} & T(M)\langle\!\langle T(N)\rangle\!\rangle
\end{array}$$

As an example consider the case $M = P\langle x := Q\rangle\langle\!\langle y := R\rangle\!\rangle \to_c P\langle\!\langle y := R\rangle\!\rangle\langle x := Q\langle\!\langle y := R\rangle\!\rangle\rangle = N$. Then $T(M) = (T(P)\langle T(Q)\rangle)\langle\!\langle T(R)\rangle\!\rangle$ and $T(N) = (T(P)\langle\!\langle T(R)\rangle\!\rangle)\langle T(Q)\langle\!\langle T(R)\rangle\!\rangle\rangle$. Now since $\langle\!\langle\rangle\!\rangle \gg \langle\rangle$ we must verify two subcases:

1. $T(M) \succ_{\mathcal{T}_l} T(P)\langle\!\langle T(R)\rangle\!\rangle$. According to the equal heads case we must verify that $\langle T(P)\langle T(Q)\rangle, T(R)\rangle \succ'_{\mathcal{T}_l} \langle T(P), T(R)\rangle$ where $\succ'_{\mathcal{T}_l}$ is the usual multiset extension of $\succ_{\mathcal{T}_l}$. Since $T(P)$ is a subterm of $T(P)\langle T(Q)\rangle$ we are done.

2. $T(M) \succ_{\mathcal{T}_l} T(Q)\langle\!\langle T(R)\rangle\!\rangle$. Similar to the case above.

■

Labelled substitutions may be eliminated in two ways: by erasing all labelling information (Def. 4.5) or by executing the labelled substitutions (Lemma 4.4). The process of erasing labelling information shall be referred to as 'unlabelling'.

Definition 4.5 (Unlabelling for $\mathcal{T}_{\underline{\lambda\mathtt{x}}}$) For $M \in \mathcal{T}_{\underline{\lambda\mathtt{x}}}$ we define $\lfloor M\rfloor \in \mathcal{T}_{\lambda\mathtt{x}}$ as:

$$\begin{array}{lcl}
\lfloor x\rfloor & \stackrel{\text{def}}{=} & x \\
\lfloor \lambda x.P\rfloor & \stackrel{\text{def}}{=} & \lambda x.\lfloor P\rfloor \\
\lfloor PQ\rfloor & \stackrel{\text{def}}{=} & \lfloor P\rfloor\lfloor Q\rfloor \\
\lfloor P\langle x := Q\rangle\rfloor & \stackrel{\text{def}}{=} & \lfloor P\rfloor\langle x := \lfloor Q\rfloor\rangle \\
\lfloor P\langle\!\langle x := Q\rangle\!\rangle\rfloor & \stackrel{\text{def}}{=} & \lfloor P\rfloor\langle x := Q\rangle
\end{array}$$

Let us now see how labelled rewriting and unlabelled rewriting may be related. The following result may be proved by induction on M.

Lemma 4.6 (Unlabelling rewrite projection and lifting) Let $M, N \in \mathcal{T}_{\underline{\lambda\mathtt{x}}}$ and $M', N' \in \mathcal{T}_{\lambda\mathtt{x}}$. The following diagrams hold:

$$\begin{tikzcd}
M \arrow[r, "\underline{\mathtt{x}}"'] \arrow[d, dotted, "\lfloor\bullet\rfloor"'] & N \arrow[d, dotted, "\lfloor\bullet\rfloor"'] \\
\lfloor M\rfloor \arrow[r, dotted, "\mathtt{x}"'] & \lfloor N\rfloor
\end{tikzcd}
\qquad
\begin{tikzcd}
M \arrow[r, "\underline{\lambda\mathtt{x}}"'] \arrow[d, dotted, "\lfloor\bullet\rfloor"'] & N \arrow[d, dotted, "\lfloor\bullet\rfloor"'] \\
\lfloor M\rfloor \arrow[r, dotted, "\lambda\mathtt{x}"'] & \lfloor N\rfloor
\end{tikzcd}
\qquad
\begin{tikzcd}
M \arrow[r, dotted, "\underline{\lambda\mathtt{x}}"'] \arrow[d, "\lfloor\bullet\rfloor"'] & N \arrow[d, dotted, "\lfloor\bullet\rfloor"'] \\
M' \arrow[r, "\lambda\mathtt{x}"'] & N'
\end{tikzcd}$$

(a) (b) (c)

Definition 4.7 (The $\underline{\lambda\mathtt{x}}^i$ and $\lambda\mathtt{x}^e$-rewrite systems) Reduction in the $\underline{\lambda\mathtt{x}}$-rewrite system may be partitioned into reduction in the $\underline{\lambda\mathtt{x}}^i$ and $\lambda\mathtt{x}^e$-rewrite systems, i.e. $\underline{\lambda\mathtt{x}} = \underline{\lambda\mathtt{x}}^i \uplus \lambda\mathtt{x}^e$, where:

1. The $\underline{\lambda\mathtt{x}}^i$-rewrite system is $\lambda\mathtt{x}$-reduction in the bodies (denoted $\lambda\mathtt{x}^i$-reduction, i.e. the contextual closure of $\to_R$: if $M \to_{\lambda\mathtt{x}} N$ then $P\langle\!\langle x := M\rangle\!\rangle \to_R P\langle\!\langle x := N\rangle\!\rangle$) of labelled substitutions together with $\underline{\mathtt{x}}$-reduction.

2. The $\lambda\mathtt{x}^e$-rewrite system is $\lambda\mathtt{x}$-reduction over $\mathcal{T}_{\underline{\lambda\mathtt{x}}}$ *except inside the bodies of labelled substitution operators.*

Remark 4.8 Let us denote with $\mathcal{T}^{\underline{snb}}_{\underline{\lambda\mathtt{x}}}$ those terms in $\mathcal{T}_{\underline{\lambda\mathtt{x}}}$ such that all bodies of labelled substitution operators are strongly $\lambda\mathtt{x}$-normalizing terms[4]; thus $\mathcal{T}^{\underline{snb}}_{\underline{\lambda\mathtt{x}}} \subset \mathcal{T}_{\underline{\lambda\mathtt{x}}}$. Note that $\mathcal{T}^{\underline{snb}}_{\underline{\lambda\mathtt{x}}}$ is closed with respect to $\underline{\lambda\mathtt{x}} \cup c$-reduction. This may be verified by a close inspection of the rewrite rules and observing that no new bodies of substitutions are created. In particular, $\mathcal{T}^{\underline{snb}}_{\underline{\lambda\mathtt{x}}}$ is closed with respect to $\underline{\lambda\mathtt{x}}^i$-reduction.

So we know that any $\underline{\lambda\mathtt{x}}$-derivation consists of $\underline{\lambda\mathtt{x}}^i$ and $\lambda\mathtt{x}^e$ steps. If the derivation starts from a term in $\mathcal{T}^{\underline{snb}}_{\underline{\lambda\mathtt{x}}}$ and is infinite then in fact we shall conclude that there are infinitely many $\lambda\mathtt{x}^e$-rewrite steps. We shall reach this conclusion by showing that $\underline{\lambda\mathtt{x}}^i$ is terminating on the set of terms in $\mathcal{T}^{\underline{snb}}_{\underline{\lambda\mathtt{x}}}$.

Lemma 4.9 The $\underline{\lambda\mathtt{x}}^i$-rewrite system is strongly normalizing on $\mathcal{T}^{\underline{snb}}_{\underline{\lambda\mathtt{x}}}$.

Proof. The idea is to define a strictly positive total function h such that $h(M_1) > h(M_2)$ if $M_1 \rightarrow_{\underline{\lambda\mathtt{x}}^i} M_2$. This function results from merging the original interpretation used to show that the $\mathtt{x}$-calculus is SN and the idea that each $\lambda\mathtt{x}$-rewrite step in the body of a labelled substitution must be taken into account.

Consider the strictly positive total functions $g : \mathcal{T}_{\lambda\mathtt{x}} \longrightarrow \mathbb{N}^{>0}$ and $h : \mathcal{T}_{\underline{\lambda\mathtt{x}}} \longrightarrow \mathbb{N}^{>0}$

$$\begin{array}{lcl@{\qquad}lcl}
g(x) & \stackrel{\text{def}}{=} & 1 & h(x) & \stackrel{\text{def}}{=} & 1 \\
g(MN) & \stackrel{\text{def}}{=} & g(M)+g(N)+1 & h(MN) & \stackrel{\text{def}}{=} & h(M)+h(N)+1 \\
g(\lambda x.M) & \stackrel{\text{def}}{=} & g(M)+1 & h(\lambda x.M) & \stackrel{\text{def}}{=} & h(M)+1 \\
g(M\langle x := N\rangle) & \stackrel{\text{def}}{=} & g(M)+g(N)+1 & h(M\langle x := N\rangle) & \stackrel{\text{def}}{=} & h(M)+h(N)+1 \\
 & & & h(M\langle\!\langle x := N\rangle\!\rangle) & \stackrel{\text{def}}{=} & h(M)(\mathit{maxred}_{\lambda\mathtt{x}}(N)+\mathit{maxg}(N)+1)
\end{array}$$

where for $N \in \mathcal{SN}_{\lambda\mathtt{x}}$ we have $\mathit{maxg}(N) \stackrel{\text{def}}{=} \mathit{max}\{g(P) \mid N \twoheadrightarrow_{\lambda\mathtt{x}} P\}$. We now proceed to verify that $h(M_1) > h(M_2)$ if $M_1 \rightarrow_{\underline{\lambda\mathtt{x}}^i} M_2$ by induction on the position where the rewrite step takes place.

- The rewrite step is at the root. Let $\mathit{maxred}_{\lambda\mathtt{x}}(P) = n$ and $\mathit{maxg}(P) = m$ in the following rewrite steps:
  - (*lApp*). Then $h((MN)\langle\!\langle x := P\rangle\!\rangle) = (h(M)+h(N)+1)(n+m+1) > h(M)(m+n+1)+h(N)(m+n+1)+1 = h((M\langle\!\langle x := P\rangle\!\rangle)(N\langle\!\langle x := P\rangle\!\rangle))$.
  - (*lLam*). Then $h((\lambda y.M)\langle\!\langle x := P\rangle\!\rangle) = (h(M)+1)(n+m+1) > h(M)(m+n+1)+1 = h(\lambda y.M\langle\!\langle x := P\rangle\!\rangle)$.
  - (*lVar*). Then $h(x\langle\!\langle x := P\rangle\!\rangle) = n+m+1 > h(P)$ by definition of m. Indeed, note that $m = \mathit{maxg}(P) \geq g(P) = h(P)$ since $P \in \mathcal{T}_{\lambda\mathtt{x}}$.
  - (*lGc*). Then $h(M\langle\!\langle y := P\rangle\!\rangle) = h(M)(n+m+1) > h(M)$, as in the previous case.
  - ($\lambda\mathtt{x}^i$). Suppose $M\langle\!\langle x := P\rangle\!\rangle \rightarrow_{\lambda\mathtt{x}^i} M\langle\!\langle x := P'\rangle\!\rangle$ (recall that a $\lambda\mathtt{x}^i$-rewrite step at the root is a $\lambda\mathtt{x}$-rewrite step in P).

    Then $h(M)(\mathit{maxred}_{\lambda\mathtt{x}}(P)+\mathit{maxg}(P)+1) > h(M)(\mathit{maxred}_{\lambda\mathtt{x}}(P')+\mathit{maxg}(P')+1)$ since $\mathit{maxred}_{\lambda\mathtt{x}}(P) > \mathit{maxred}_{\lambda\mathtt{x}}(P')$ and $\mathit{maxg}(P) \geq \mathit{maxg}(P')$.
- The rewrite step is internal. Then we consider each possible context: if $M_1 = M_3M_4$ or $M_1 = \lambda x.M_3$ or $M_1 = M_3\langle x := M_4\rangle$ then we use the induction hypothesis. If $M_1 = M_3\langle\!\langle x := P\rangle\!\rangle$ then the $\underline{\lambda\mathtt{x}}^i$-rewrite step must be in M_3 (it cannot be in P since there are no labelled substitution operators in P) and we may also use the induction hypothesis yielding $h(M_3) > h(M_3')$ and thus $h(M_3)(n+m+1) > h(M_3')(n+m+1)$.

∎

Lemma 4.10 (Labelled rewrite projection) Let $M, N \in \mathcal{T}_{\underline{\lambda\mathtt{x}}}$. The following diagram holds:

$$\begin{tikzcd}
M \arrow[r, "\underline{\lambda\mathtt{x}}"'] \arrow[d, "\underline{\mathtt{xc}}"'] & N \arrow[d, "\underline{\mathtt{xc}}"] \\
\underline{\mathtt{xc}}(M) \arrow[r, dotted, two heads, "\lambda\mathtt{x}"'] & \underline{\mathtt{xc}}(N)
\end{tikzcd}$$

[4] The letters '*snb*' in $\mathcal{T}^{\underline{snb}}_{\underline{\lambda\mathtt{x}}}$ stand for 'strongly normalizing bodies', and the underlining in $\underline{snb}$ recalls the reader that it is just the bodies of labelled substitutions which are referred to.

The proof of Lemma 4.10 is by induction on M and requires using metalevel substitution (Def. 2.32). Also, Lemma 4.11 is by induction on M.

Lemma 4.11 ($\lambda\mathtt{x}^e$-rewrite projection) Let $M, N \in \mathcal{T}_{\underline{\lambda \mathtt{x}}}$. The following diagram holds:

$$\begin{tikzcd}
M \arrow[r, "\lambda\mathtt{x}^e"'] \arrow[d, "\underline{\mathtt{xc}}"'] & N \arrow[d, "\underline{\mathtt{xc}}"] \\
\underline{\mathtt{xc}}(M) \arrow[r, dotted, "+", "\lambda\mathtt{x}"'] & \underline{\mathtt{xc}}(N)
\end{tikzcd}$$

We would like to bring the reader's attention to the fact that for the proofs of Lemmas 4.10 and 4.11 to go through the full power of the garbage collection rule (Gc-rule) is needed. Consider the outcome if the restricted garbage collection rule ($x\langle y := P\rangle \rightarrow_{rGc} x$) were used instead: $M = x\langle y := z\rangle\langle\langle x := P\rangle\rangle \rightarrow_{\lambda\mathtt{x}^e} x\langle\langle x := P\rangle\rangle = N$ yet $\underline{\mathtt{xc}}(M) = P\langle y := z\rangle$ and $\underline{\mathtt{xc}}(N) = P$ but in general we do not have $\underline{\mathtt{xc}}(M) \rightarrow_{rGc} \underline{\mathtt{xc}}(N)$.

As a consequence of Lemmas 4.10 and 4.11 we have the following result:

Corollary 4.12 (Perpetuality of labelled rewriting) Let $M \in \mathcal{T}^{snb}_{\underline{\lambda\mathtt{x}}}$. If there is an infinite $\underline{\lambda\mathtt{x}}$-derivation from M then there is an infinite $\lambda\mathtt{x}$-derivation from $\underline{\mathtt{xc}}(M)$.

Proof. Suppose there is an infinite $\underline{\lambda\mathtt{x}}$-derivation starting from M. Then since the $\underline{\lambda\mathtt{x}}^i$-rewrite system is strongly normalizing on $\mathcal{T}^{snb}_{\underline{\lambda\mathtt{x}}}$ (Lemma 4.9) this derivation must have the form:

$$M = M_0 \twoheadrightarrow_{\underline{\lambda\mathtt{x}}^i} M_1 \rightarrow_{\lambda\mathtt{x}^e} M_2 \twoheadrightarrow_{\underline{\lambda\mathtt{x}}^i} M_3 \rightarrow_{\lambda\mathtt{x}^e} M_4 \ldots$$

Then applying Lemma 4.10 (extended to many step reduction and noting that $\underline{\lambda\mathtt{x}}^i \subset \underline{\lambda\mathtt{x}}$) and Lemma 4.11 we may construct the following diagram and conclude the proof:

$$\begin{tikzcd}
M_0 \arrow[r, two heads, "\underline{\lambda\mathtt{x}}^i"'] \arrow[d, "\underline{\mathtt{xc}}"] & M_1 \arrow[r, "\lambda\mathtt{x}^e"'] \arrow[d, "\underline{\mathtt{xc}}"] & M_2 \arrow[r, two heads, "\underline{\lambda\mathtt{x}}^i"'] \arrow[d, "\underline{\mathtt{xc}}"] & M_3 \arrow[r, "\lambda\mathtt{x}^e"'] \arrow[d, "\underline{\mathtt{xc}}"] & M_4 \arrow[r, two heads, "\underline{\lambda\mathtt{x}}^i"'] \arrow[d, "\underline{\mathtt{xc}}"] & \cdots \\
\underline{\mathtt{xc}}(M_0) \arrow[r, dotted, two heads, "\lambda\mathtt{x}"'] & \underline{\mathtt{xc}}(M_1) \arrow[r, dotted, "+", "\lambda\mathtt{x}"'] & \underline{\mathtt{xc}}(M_2) \arrow[r, dotted, two heads, "\lambda\mathtt{x}"'] & \underline{\mathtt{xc}}(M_3) \arrow[r, dotted, "+", "\lambda\mathtt{x}"'] & \underline{\mathtt{xc}}(M_4) \arrow[r, dotted, two heads, "\lambda\mathtt{x}"'] & \cdots
\end{tikzcd}$$

■

Note that since $\underline{\lambda\mathtt{x}}$-reduction includes $\lambda\mathtt{x}$-reduction Corollary 4.12 also holds if $\underline{\lambda\mathtt{x}}$ is replaced by $\lambda\mathtt{x}$ in its statement. It is possible to strengthen this result and prove that *each* $\underline{\mathtt{xc}}$-reduction step is $\underline{\lambda\mathtt{x}}$-perpetual. This leads to a new proof of perpetuality of the strategies presented in Section 3.2.2 and allows to infer the other results presented in that chapter, such as the inductive characterization of the set $SN_{\lambda\mathtt{x}}$ and termination of typed F_{es}. The reader may find further details in [ABR00].

4.1.2 A Digression: PSN

Corollary 4.12 shall be the main result used in proving perpetual our rewrite strategy (Def. 4.26). Although in this chapter we are not interested in proving preservation of strong normalization, we would like to show how Corollary 4.12 may be used for this purpose.

We shall use $M \in \bowtie_{\lambda\mathtt{x}}$ as an abbreviation for $M \in \mathcal{T}_\lambda \cap \infty_{\lambda\mathtt{x}}$, i.e. the set of pure terms that admit infinite $\lambda\mathtt{x}$-derivations. Given a pure term M such that $\infty_{\lambda\mathtt{x}}(M)$ (hence $M \in \bowtie_{\lambda\mathtt{x}}$) the following *zoom-in function* $V(\bullet)$ allows us to obtain a *'minimal'* (in the sense that all strict subterms are strongly $\lambda\mathtt{x}$-normalizing terms) pure term N such that also $N \in \bowtie_{\lambda\mathtt{x}}$ and $N \subseteq M$. This definition is a variant of Definition 4.8 in [RSSX99].

Definition 4.13 (Zoom-in function on pure terms) $V(\bullet) : \bowtie_{\lambda\mathtt{x}} \longrightarrow \mathcal{T}_\lambda$ is defined:

$$\begin{array}{llll}
V(x\vec{P}Q\vec{R}) & \stackrel{\text{def}}{=} & V(Q) & \text{if } \vec{P} \in SN_{\lambda\mathtt{x}}, Q \notin SN_{\lambda\mathtt{x}} \\
V(\lambda x.P) & \stackrel{\text{def}}{=} & V(P) & \\
V((\lambda x.P)Q\vec{R}) & \stackrel{\text{def}}{=} & (\lambda x.P)Q\vec{R} & \text{if } P, Q, \vec{R} \in SN_{\lambda\mathtt{x}} \\
V((\lambda x.P)Q\vec{R}) & \stackrel{\text{def}}{=} & V(P) & \text{if } P \notin SN_{\lambda\mathtt{x}} \\
V((\lambda x.P)\vec{Q}S\vec{R}) & \stackrel{\text{def}}{=} & V(S) & \text{if } P, \vec{Q} \in SN_{\lambda\mathtt{x}}, S \notin SN_{\lambda\mathtt{x}}
\end{array}$$

Lemma 4.14 (Properties of $V(\bullet)$) For all $M \in \bowtie_{\lambda x}$ we have:

1. $V(M) = (\lambda x.P)Q\vec{R}$ for some $P, Q, \vec{R} \in SN_{\lambda x}$.
2. $V(M) \subseteq M$.
3. $V(M) \in \bowtie_{\lambda x}$.

Lemma 4.15 (Lifting lemma) Let $M \in \bowtie_{\lambda x}$ with $V(M) = (\lambda x.P)Q\vec{R}$. Then there exists an infinite $\underline{\lambda x}$-derivation starting from $\underline{V(M)} \stackrel{\text{def}}{=} P\langle\!\langle x := Q\rangle\!\rangle\vec{R}$, i.e. the following diagram holds:

$$\begin{tikzcd}
V(M) \arrow[r, "Beta_l"'] \arrow[dr, "\underline{Beta_l}"'] & M_0 \arrow[r, "\lambda x"'] & M_1 \arrow[r, "\lambda x"'] & M_2 \arrow[r, "\lambda x"'] & M_3 \arrow[r, "\lambda x"'] & \cdots \\
& \underline{V(M)} \arrow[u, "\lfloor\bullet\rfloor"] \arrow[r, dotted, "\underline{\lambda x}"'] & M'_1 \arrow[u, "\lfloor\bullet\rfloor"] \arrow[r, dotted, "\underline{\lambda x}"'] & M'_2 \arrow[u, "\lfloor\bullet\rfloor"] \arrow[r, dotted, "\underline{\lambda x}"'] & M'_3 \arrow[u, "\lfloor\bullet\rfloor"] \arrow[r, dotted, "\underline{\lambda x}"'] & \cdots
\end{tikzcd}$$

where $\underline{Beta_l}$ just stands for leftmost $Beta$-reduction where a labelled substitution operator is introduced instead of an unlabelled one.

Proof. The proof follows from Lemma 4.6(c) and the fact that any infinite λx-derivation starting from $V(M)$ may be transformed into another one in which the first reduction step is a $Beta_l$-reduction step. ▪

Lemma 4.16 Let $M \in \bowtie_{\lambda x}$. Then there exists $N \in \bowtie_{\lambda x}$ such that $M \rightarrow_\beta N$.

Proof. Let $M \in \bowtie_{\lambda x}$. Then by Lemma 4.14(3) we have $V(M) \in \bowtie_{\lambda x}$. Using the Lifting Lemma (Lemma 4.15) we may construct the top part of the following diagram:

$$\begin{tikzcd}
V(M) \arrow[r, "Beta_l"'] \arrow[dr, "\underline{Beta_l}"'] \arrow[dd, "\beta_l"'] & M_0 \arrow[r, "\lambda x"'] & M_1 \arrow[r, "\lambda x"'] & M_2 \arrow[r, "\lambda x"'] & M_3 \arrow[r, "\lambda x"'] & \cdots \\
& \underline{V(M)} \arrow[u, "\lfloor\bullet\rfloor"] \arrow[r, "\underline{\lambda x}"'] \arrow[d, "\underline{xc}"] & M'_1 \arrow[u, "\lfloor\bullet\rfloor"] \arrow[r, "\underline{\lambda x}"'] & M'_2 \arrow[u, "\lfloor\bullet\rfloor"] \arrow[r, "\underline{\lambda x}"'] & M'_3 \arrow[u, "\lfloor\bullet\rfloor"] \arrow[r, "\underline{\lambda x}"'] & \cdots \\
N' \arrow[r, phantom, "="] & \underline{xc(V(M))} \arrow[r, dotted, "+", "\lambda x"'] & {} \arrow[r, dotted, two heads, "\lambda x"'] & {} \arrow[r, dotted, "+", "\lambda x"'] & {} \arrow[r, dotted, two heads, "\lambda x"'] & \cdots
\end{tikzcd}$$

The bottom part is completed by using Corollary 4.12. Finally, since $V(M) \subseteq M$ there is a pure context (a context without occurrences of the substitution operator) C such that $M = C[V(M)]$ and $N = C[N']$ with $N \in \bowtie_{\lambda x}$.

▪

The following zoom-in strategy was proved perpetual for the λ-calculus by Melliès [Mel96]. Here we present the *zoom-in strategy on pure terms* based on our Def. 4.13. It is perpetual for the λx-calculus too (see proof of Corollary 4.18).

Definition 4.17 (Zoom-in strategy for pure terms) Let $M \in \bowtie_{\lambda x}$. The zoom-in strategy $\mathrm{Z}(\bullet)\colon \bowtie_{\lambda x} \longrightarrow \mathcal{T}_\lambda$ is defined as:

$$\begin{array}{lcll}
\mathrm{Z}(x\vec{P}Q\vec{R}) & \stackrel{\text{def}}{=} & x\vec{P}\mathrm{Z}(Q)\vec{R} & \text{if } \vec{P} \in SN_{\lambda x}, Q \notin SN_{\lambda x} \\
\mathrm{Z}(\lambda x.P) & \stackrel{\text{def}}{=} & \lambda x.\mathrm{Z}(P) & \\
\mathrm{Z}((\lambda x.P)Q\vec{R}) & \stackrel{\text{def}}{=} & P\{x \leftarrow Q\}\vec{R} & \text{if } P, Q, \vec{R} \in SN_{\lambda x} \\
\mathrm{Z}((\lambda x.P)Q\vec{R}) & \stackrel{\text{def}}{=} & (\lambda x.\mathrm{Z}(P))Q\vec{R} & \text{if } P \notin SN_{\lambda x} \\
\mathrm{Z}((\lambda x.P)\vec{Q}S\vec{R}) & \stackrel{\text{def}}{=} & (\lambda x.P)\vec{Q}\mathrm{Z}(S)\vec{R} & \text{if } P, \vec{Q} \in SN_{\lambda x}, S \notin SN_{\lambda x}
\end{array}$$

Finally, we conclude the section by showing PSN for λx.

Corollary 4.18 (PSN for λx) Let $M \in \bowtie_{\lambda x}$ then $M \in \infty_\beta$.

Proof. By applying repeatedly Lemma 4.16 we obtain an infinite β-derivation starting from M. See [ABR00] for details.

$$\begin{tikzcd}
M \arrow[d, "\beta"'] & \supseteq & V(M) \in \bowtie_{\lambda x} \arrow[d, "\beta_I"'] & & & & \\
\mathrm{Z}(M) \arrow[d, "\beta"'] & \supseteq & N_1 & \supseteq & V(\mathrm{Z}(M)) \in \bowtie_{\lambda x} \arrow[d, "\beta_I"'] & & \\
\mathrm{Z}(\mathrm{Z}(M)) \arrow[d, "\beta"'] & \supseteq & & & N_2 & \supseteq & V(\mathrm{Z}(\mathrm{Z}(M))) \in \bowtie_{\lambda x} \arrow[d, "\beta_I"'] \\
\mathrm{Z}(\mathrm{Z}(\mathrm{Z}(M))) & \supseteq & & & & & N_3
\end{tikzcd}$$

4.1.3 The Zx(•) Perpetual Strategy

In this section we introduce a substitution labelling strategy for λx. This consists in defining a strategy over a class of terms (the domain of the labelling strategy) which selects an occurrence of a substitution operator that may safely be computed while preserving the possibility of infinite λx-derivations. But before proceeding we present the *zoom-in function* for λx.

Given a term M admitting an infinite λx-derivation, the strategy Zx(•) shall zoom-in into the term in order to select the redex to contract. The same effect is obtained by the following zoom-in function, the only difference being that this function just selects the minimal subterm where the strategy Zx(•) shall find its redex without actually contracting any redex at all.

Definition 4.19 (Zoom-in function) Let $M \in \mathcal{T}_{\lambda x}$ such that $\infty_{\lambda x}(M)$. We define $Vx(M)$ by induction on M:

$$\begin{array}{lll}
Vx(x\vec{P}Q\vec{R}) & \stackrel{\text{def}}{=} Vx(Q) & \text{if } \vec{P} \in SN_{\lambda x}, Q \notin SN_{\lambda x} \\
Vx(\lambda x.P) & \stackrel{\text{def}}{=} Vx(P) & \\
Vx((\lambda x.P)Q\vec{R}) & \stackrel{\text{def}}{=} (\lambda x.P)Q\vec{R} & \text{if } P, Q, \vec{R} \in SN_{\lambda x} \\
Vx((\lambda x.P)Q\vec{R}) & \stackrel{\text{def}}{=} Vx(P) & \text{if } P \notin SN_{\lambda x} \\
Vx((\lambda x.P)\vec{Q}S\vec{R}) & \stackrel{\text{def}}{=} Vx(S) & \text{if } P, \vec{Q} \in SN_{\lambda x}, S \notin SN_{\lambda x} \\
Vx(P\langle x := Q\rangle\vec{R}) & \stackrel{\text{def}}{=} P\langle x := Q\rangle\vec{R} & \text{if } P, Q, \vec{R} \in SN_{\lambda x} \\
Vx(P\langle x := Q\rangle\vec{R}) & \stackrel{\text{def}}{=} Vx(P) & \text{if } P \notin SN_{\lambda x} \\
Vx(P\langle x := Q\rangle\vec{R}) & \stackrel{\text{def}}{=} Vx(Q) & \text{if } P \in SN_{\lambda x}, Q \notin SN_{\lambda x} \\
Vx(P\langle x := Q\rangle\vec{Q}S\vec{R}) & \stackrel{\text{def}}{=} Vx(S) & \text{if } P, Q, \vec{Q} \in SN_{\lambda x}, S \notin SN_{\lambda x}
\end{array}$$

$Vx(\bullet)$ satisfies some properties for which we shall require the following definition of decent terms which first appeared in [Blo97].

Definition 4.20 (Decent terms) We define the set of *decent terms* $\mathcal{T}^{snb}_{\lambda x} \subset \mathcal{T}_{\lambda x}$ as: $M \in \mathcal{T}^{snb}_{\lambda x}$ iff the bodies of the substitutions in M are λx-strongly normalizing[5] (i.e. if $N_1\langle x := N_2\rangle \subseteq M$ then $N_2 \in SN_{\lambda x}$).

The set of decent terms is closed under x-rewriting, in other words, x-rewriting does not introduce substitutions whose bodies possess infinite λx-derivations if there were none in the first place. This fails for the λws-calculus, as we shall see.

Lemma 4.21 (Preservation of decent terms by x-reduction) Let $M \in \mathcal{T}^{snb}_{\lambda x}$. If $M \rightarrow_{x} N$ then $N \in \mathcal{T}^{snb}_{\lambda x}$.

The proof follows from a close inspection of the x-rewrite rules. Now back to the properties of $Vx(\bullet)$.

Lemma 4.22 (Properties of $Vx(\bullet)$) For all $M \in \mathcal{T}_{\lambda x}$ such that $\infty_{\lambda x}(M)$ we have:

1. $Vx(M) = (\lambda x.P)Q\vec{R}$ or $Vx(M) = P\langle x := Q\rangle\vec{R}$ for some $P, Q, \vec{R} \in SN_{\lambda x}$ (hence $Vx(M) \in \mathcal{T}^{snb}_{\lambda x}$).

[5] The letters '*snb*' in $\mathcal{T}^{snb}_{\lambda x}$ stand for 'strongly normalizing bodies'. Note that they are not underlined so as to remind the reader that it is the bodies of unlabelled substitution operators which are referred to. Likewise, λx is not underlined to remind the reader that we are dealing with λx-terms (no labels are present).

2. $V\mathtt{x}(M) \subseteq M$.

3. $\infty_{\lambda\mathtt{x}}(V\mathtt{x}(M))$.

Now depending on the result of $V\mathtt{x}(M)$ two situations may arise. The action to be taken by $\mathtt{Zx}(\bullet)$ shall depend on this result.

1. If $V\mathtt{x}(M) = (\lambda x.P)Q\vec{R}$ with $P, Q, \vec{R} \in SN_{\lambda\mathtt{x}}$, then $\mathtt{Zx}(M)$ shall simply contract the leftmost *Beta*-redex of $V\mathtt{x}(M)$. Hence constituting a zoom-in strategy as commented in the introduction of this section.

2. However, if $V\mathtt{x}(M) = P\langle x := Q\rangle\vec{R}$ for some $P, Q, \vec{R} \in SN_{\lambda\mathtt{x}}$, then $\mathtt{Zx}(\bullet)$ must decide which x-redex to contract in $V\mathtt{x}(M)$. This task shall be derived to the labelling strategy. The latter shall select, in this case, the innermost-leftmost substitution, label it, and compute it by means of the $\underline{\mathtt{xc}}$-rewrite system.

This division of the strategy $\mathtt{Zx}(\bullet)$ shall prove convenient for analysing the more complex case of $\lambda\mathtt{ws}$.

Let us now address the labelling strategy. First some notation: we call an occurrence $N_1\langle x := N_2\rangle \subseteq M$ of a substitution operator *innermost* if $N_1 \in \mathcal{T}_\lambda$, that is to say, if N_1 is a pure term.

Definition 4.23 (Innermost substitution labelling) Let $M \in \mathcal{T}^{snb}_{\lambda\mathtt{x}}$ then $\underline{M}$ denotes the term M where the innermost-leftmost substitution has been labelled if M is not pure, and M otherwise. Thus $\underline{\bullet} : \mathcal{T}^{snb}_{\lambda\mathtt{x}} \longrightarrow \mathcal{T}^{\underline{snb}}_{\underline{\lambda\mathtt{x}}}$ (see Remark 4.8 for the definition of $\mathcal{T}^{\underline{snb}}_{\underline{\lambda\mathtt{x}}}$).

We now define a (many-step) x-reduction strategy that reduces innermost substitution operators.

Definition 4.24 (Substitution labelling strategy) The substitution labelling strategy for $\lambda\mathtt{x}$ is given by the (many-step) x-reduction strategy $\mathtt{Lx}(\bullet) : \mathcal{T}^{snb}_{\lambda\mathtt{x}} \longrightarrow \mathcal{T}^{snb}_{\lambda\mathtt{x}}$ as $\mathtt{Lx}(M) \overset{\text{def}}{=} \underline{\mathtt{xc}}(\underline{M})$. The domain of the labelling strategy is the set $\mathcal{T}^{snb}_{\lambda\mathtt{x}}$.

Note that we may just as well have defined $\mathtt{Lx}(\bullet)$ as $\mathtt{Lx}(M) \overset{\text{def}}{=} \underline{\mathtt{x}}(\underline{M})$ since the composition rule is not used in the reduction $\underline{M} \twoheadrightarrow_{\underline{\mathtt{xc}}} \underline{\mathtt{xc}}(\underline{M})$ due to the fact that it is innermost substitutions that are labelled. Note that $\mathtt{Lx}(\bullet)$ is an x-strategy, indeed: if M is not pure then $\underline{M} \xrightarrow{+}_{\underline{\mathtt{x}}} \underline{\mathtt{xc}}(\underline{M})$ and hence by Lemma 4.6(a) $M = \lfloor \underline{M} \rfloor \xrightarrow{+}_{\mathtt{x}} \lfloor \underline{\mathtt{xc}}(\underline{M}) \rfloor = \underline{\mathtt{xc}}(\underline{M})$. Also, $\mathtt{Lx}(M) \in \mathcal{T}^{snb}_{\lambda\mathtt{x}}$ by Lemma 4.21.

Lemma 4.25 (Properties of $\mathtt{Lx}(\bullet)$) 1. The $\mathtt{Lx}(\bullet)$ strategy is x-normalizing: for all $M \in \mathcal{T}^{snb}_{\lambda\mathtt{x}}$ there exists $n \geq 0$ such that $\mathtt{Lx}^n(M)$ is in x-normal form.

2. The $\mathtt{Lx}(\bullet)$ strategy is $\lambda\mathtt{x}$-perpetual: let $M \in \mathcal{T}^{snb}_{\lambda\mathtt{x}}$ and also $\infty_{\lambda\mathtt{x}}(M)$. Then $\infty_{\lambda\mathtt{x}}(\mathtt{Lx}(M))$.

Proof. The first item is a consequence of strong normalization of the x-rewrite system. The second item follows from Corollary 4.12. Indeed, since $\infty_{\lambda\mathtt{x}}(M)$ then by Lemma 4.6(c) $\underline{M}$ has an infinite $\underline{\lambda\mathtt{x}}$-derivation. Corollary 4.12 yields $\infty_{\lambda\mathtt{x}}(\underline{\mathtt{xc}}(\underline{M}))$. ▪

Note that if $M \in \mathcal{T}^{snb}_{\lambda\mathtt{x}}$ such that $\infty_{\lambda\mathtt{x}}(M)$, then from repeated application of the $\mathtt{Lx}(\bullet)$ strategy to $V\mathtt{x}(M)$ we obtain $\mathtt{x}(V\mathtt{x}(M)) \in \bowtie_{\lambda\mathtt{x}}$.

Definition 4.26 (The $\mathtt{Zx}(\bullet)$ Strategy) Let $M \in \mathcal{T}_{\lambda\mathtt{x}}$ such that $\infty_{\lambda\mathtt{x}}(M)$, and let C be the context such that $M = C[V\mathtt{x}(M)]$. We define $\mathtt{Zx}(M)$ as:

$$\mathtt{Zx}(M) \overset{\text{def}}{=} \begin{cases} C[P\langle x := Q\rangle\vec{R}] & \text{if } V\mathtt{x}(M) = (\lambda x.P)Q\vec{R} \\ C[\mathtt{Lx}(P\langle x := Q\rangle\vec{R})] & \text{if } V\mathtt{x}(M) = P\langle x := Q\rangle\vec{R} \end{cases}$$

Note how in the sixth clause, the $\mathtt{Lx}(\bullet)$ strategy is in charge of selecting the closure to compute. By the observation just before Lemma 4.25, $\mathtt{Zx}(\bullet)$ is indeed a many-step $\lambda\mathtt{x}$-rewrite strategy. It is also perpetual.

Lemma 4.27 The $\mathtt{Zx}(\bullet)$ strategy is perpetual.

This may be proved for $\mathtt{Zx}(M)$ by induction on M using Lemma 4.25(2).

4.2 Zooming-in on λws

Next we shall consider the setting of the λws-calculus. In the lines of the work developed in the previous section we shall formulate a perpetual rewrite strategy for λws. Also, an inductive characterization of the terms in $SN_{\lambda ws}$ shall be given. As already mentioned, due to the presence of rewrite rules allowing the composition of substitutions the problem is technically more demanding. A brief introduction to the λws-calculus may be found in Section 2.3.4 of Chapter 2. Although the rewrite rules of λws are reproduced below for the reader's convenience she/he is advised to take a quick look at the aforementioned section.

Warning on notation to the reader familiar with λws: in order to be consistent with the nomenclature used in Chapter 3 we have adopted the term *labelled* instead of tagged as used in [DG01]. Thus, for example, well-tagged terms in the terminology of [DG01] shall be referred to as well-labelled here.

4.2.1 The labelled λws-calculus

We begin this section by characterizing the set of λws-terms as given in Def. 2.56 of Chapter 2.

Lemma 4.28 Every term M in $\mathcal{T}_{\lambda ws}$ has exactly one of the following forms:

1. $\langle\vec{k}\rangle n\vec{R}$
2. $\langle\vec{k}\rangle\lambda P$
3. $(\langle\vec{k}\rangle\lambda P)Q\vec{R}$
4. $(\langle\vec{k}\rangle P[i/Q,j])\vec{R}$
5. $(\langle\vec{k}^{+}\rangle(PQ))\vec{R}$

We recall from Def. 2.58 the rewrite rules of the λws-calculus.

$$\mathtt{B}\left\{\begin{array}{lll} (\lambda M)N & \rightarrow_{b1} & M[0/N,0] \\ (\langle k\rangle\lambda M)N & \rightarrow_{b2} & M[0/N,k] \end{array}\right.$$

$$\mathtt{ws}\left\{\begin{array}{llll} (\lambda M)[i/N,j] & \rightarrow_{l} & \lambda(M[i+1/N,j]) & \\ (M_1M_2)[i/N,j] & \rightarrow_{a} & M_1[i/N,j]M_2[i/N,j] & \\ (\langle k\rangle M)[i/N,j] & \rightarrow_{e1} & \langle k+j-1\rangle M & i<k \\ (\langle k\rangle M)[i/N,j] & \rightarrow_{e2} & \langle k\rangle M[i-k/N,j] & i\geq k \\ n[i/N,j] & \rightarrow_{n1} & n & n<i \\ n[i/N,j] & \rightarrow_{n2} & \langle i\rangle N & n=i \\ n[i/N,j] & \rightarrow_{n3} & n+j-1 & n>i \\ M[k/N,l][i/P,j] & \rightarrow_{c1} & M[k/N[i-k/P,j],j+l-1] & k\leq i<k+l \\ M[k/N,l][i/P,j] & \rightarrow_{c2} & M[i-l+1/P,j][k/N[i-k/P,j],l] & k+l\leq i \\ \langle i\rangle\langle j\rangle M & \rightarrow_{m} & \langle i+j\rangle M & \end{array}\right.$$

The B-calculus is just rules $b1$ and $b2$. The substitution ws-calculus is the λws-calculus without the rules $b1$ and $b2$. The p-calculus is the ws-calculus without the m-rule.

Before recalling the labelled λws-calculus from [DG01] we define the labelled λws-terms.

Definition 4.29 (Labelled terms) The set of labelled terms, denoted $\mathcal{T}_{\underline{\lambda ws}}$, is defined as:

$$M \quad ::= \quad n \mid \lambda M \mid MM \mid \langle k\rangle M \mid M[i/M,j] \mid N[\![i/P,j]\!] \quad \text{where } n,i,j,k\in\mathbb{N}_0 \text{ and } N,P\in\mathcal{T}_{\lambda ws}$$

The $\bullet[\![\bullet/\bullet,\bullet]\!]$ operator is called the *labelled substitution operator*. Note that the target and body of a labelled substitution operator are terms without occurrences of labelled substitution operators.

Definition 4.30 (Labelled rewriting) The labelled $\lambda\mathtt{ws}$-calculus, denoted $\underline{\lambda\mathtt{ws}}$, consists of the $\lambda\mathtt{ws}$-calculus together with the following set of rewrite rules (called the $\underline{\mathtt{ws}}$-calculus):

$$\begin{array}{llll}
(\lambda(M))\llbracket i/N,j\rrbracket & \rightarrow_{ll} & \lambda(M\llbracket i+1/N,j\rrbracket) & \\
(M_1M_2)\llbracket i/N,j\rrbracket & \rightarrow_{la} & M_1\llbracket i/N,j\rrbracket M_2\llbracket i/N,j\rrbracket & \\
(\langle k\rangle M)\llbracket i/N,j\rrbracket & \rightarrow_{le1} & \langle k+j-1\rangle M & i<k \\
(\langle k\rangle M)\llbracket i/N,j\rrbracket & \rightarrow_{le2} & \langle k\rangle M\llbracket i-k/N,j\rrbracket & i\geq k \\
n\llbracket i/N,j\rrbracket & \rightarrow_{ln1} & n & n<i \\
n\llbracket i/N,j\rrbracket & \rightarrow_{ln2} & \langle i\rangle N & n=i \\
n\llbracket i/N,j\rrbracket & \rightarrow_{ln3} & n+j-1 & n>i \\
M[k/N,l]\llbracket i/P,j\rrbracket & \rightarrow_{lc1} & M[k/N\llbracket i-k/P,j\rrbracket,j+l-1] & k\leq i<k+l \\
M[k/N,l]\llbracket i/P,j\rrbracket & \rightarrow_{lc2} & M\llbracket i-l+1/P,j\rrbracket[k/N\llbracket i-k/P,j\rrbracket,l] & k+l\leq i
\end{array}$$

Note that there is no lm-rule (labelled version of the m-rewrite rule) in the $\underline{\mathtt{ws}}$-calculus (this simplifies some proofs).

In our exposition on $\underline{\lambda\mathtt{x}}$, the labelled $\lambda\mathtt{x}$-calculus of Section 4.1.1, after introducing it we immediately considered properties of its substitution calculus ($\underline{\mathtt{x}}$) and showed how $\lambda\mathtt{x}$-rewriting related to $\underline{\lambda\mathtt{x}}$-rewriting by an unlabelling function. Here we shall do the same. However, in contrast to $\lambda\mathtt{x}$ we shall not rewrite all labelled terms, but instead consider labelled rewriting on a restricted set of terms of the set $\mathcal{T}_{\underline{\lambda\mathtt{ws}}}$. The reason is that substitution composition must be taken into account. This restricted set of labelled $\lambda\mathtt{ws}$-terms shall be seen to behave correctly in the presence of substitution composition when perpetuality is under the microscope.

Let us consider an example in order to shed further light on this issue. Let M be a $\lambda\mathtt{x}$-term such that $\infty_{\lambda\mathtt{x}}(M)$. From Corollary 4.12 it follows that any x-redex in M such that the body of the substitution involved in this x-redex (denoted by the letter P in Def. 2.34) is strongly $\lambda\mathtt{x}$-normalizing, is a $\lambda\mathtt{x}$-perpetual redex. Indeed, it suffices to label this substitution, apply Corollary 4.12, and project the resulting $\underline{\lambda\mathtt{x}}$-derivation with the help of items (a) and (b) of Lemma 4.6. However, this does not hold in $\lambda\mathtt{ws}$. Consider the term $M = 4[0/00,0][0/\lambda 00,0]$. Then, as illustrated in the introduction of this chapter, $\infty_{\lambda\mathtt{ws}}(M)$. Also, the term 00 is clearly strongly $\lambda\mathtt{ws}$-normalizing (in fact it is in normal form). However, the ws-redex (more precisely, the $n3$-redex) involving the only substitution whose body is the term 00 is not perpetual since the resulting term is $3[0/\lambda 00,0]$, which no longer admits an infinite $\lambda\mathtt{ws}$-derivation. The problem is that we have erased a substitution that could potentially be combined with some other substitution by means of substitution composition, on its way to constructing an infinite derivation.

The restricted set of $\lambda\mathtt{ws}$-terms are called *well-labelled terms*. This notion is due to R.David and B.Guillaume and is the key to the proof of PSN for $\lambda\mathtt{ws}$ [DG01].

Remark 4.31 In order to convey the intuition behind Def. 4.32 assume we are given a non-pure term $M \in \mathcal{T}_{\lambda\mathtt{ws}}$. We would like to know if the term resulting from labelling some substitution operator in M occurring at position p is well-labelled. For this, and before labelling any substitution, three conditions must be met:

SN body. The body of the substitution operator at p is a $\lambda\mathtt{ws}$-strongly normalizing term. We do not wish the labelling strategy to erase terms having an infinite derivation.

Safe propagation. The substitution at p, once labelled, may safely be propagated by the $\underline{\mathtt{ws}}$-calculus until it is completely executed. In other words, there are no substitutions 'below' p *and* which could potentially block the propagation of the labelled substitution operator.

Non-interaction. Substitution operators above p or substitution operators which may be created above p may not interact with (the substitution operator at) p in the sense that they may not compose with p. Intuitively, this seeks to uphold compliance with the **SN body** condition.

The predicate $\mathcal{B}(\bullet,\bullet)$ verifies the **Safe propagation** condition and $\mathcal{H}(\bullet,\bullet)$ all three of them.

Definition 4.32 (Well-labelled terms) A term M is called *well-labelled* if there exists $m \in \mathbb{N}$ such that $\mathcal{H}(M,m)$. We use WL to denote the set of well-labelled terms. The $\mathcal{H}(\bullet,\bullet) \subseteq \mathcal{T}_{\underline{\lambda\mathtt{ws}}} \times \mathbb{N}$ relation makes use of the $\mathcal{B}(\bullet,\bullet) \subseteq \mathcal{T}_{\lambda\mathtt{ws}} \times \mathbb{N}$ relation. Both are defined below:

$$
\begin{array}{lll}
\mathcal{B}(n,m) & & \\
\mathcal{B}(\lambda(M),m) & \text{iff} & \mathcal{B}(M,m+1) \\
\mathcal{B}(MN,m) & \text{iff} & \mathcal{B}(M,m) \text{ and } \mathcal{B}(N,m) \\
\mathcal{B}(\langle i\rangle M,m) & \text{iff} & \begin{cases} i \leq m \text{ and } \mathcal{B}(M,m-i) \\ \text{or} \\ i > m \end{cases} \\
\mathcal{B}(M[i/N,j],m) & \text{iff} & \begin{cases} i \leq m < i+j \text{ and } \mathcal{B}(N,m-i) \\ \text{or} \\ i+j \leq m \text{ and } \mathcal{B}(N,m-i) \text{ and } \mathcal{B}(M,m-j+1) \end{cases}
\end{array}
$$

$$
\begin{array}{lll}
\mathcal{H}(n,m) & & \\
\mathcal{H}(\lambda(M),m) & \text{iff} & \mathcal{H}(M,m+1) \\
\mathcal{H}(MN,m) & \text{iff} & \mathcal{H}(M,m) \text{ and } \mathcal{H}(N,m) \\
\mathcal{H}(\langle i\rangle M,m) & \text{iff} & \begin{cases} i \leq m \text{ and } \mathcal{H}(M,m-i) \\ \text{or} \\ i > m \text{ and } M \in \mathcal{T}_{\lambda\mathtt{ws}} \end{cases} \\
\mathcal{H}(M[i/N,j],m) & \text{iff} & \begin{cases} m < i \text{ and } M,N \in \mathcal{T}_{\lambda\mathtt{ws}} \\ \text{or} \\ i \leq m < i+j \text{ and } \mathcal{H}(N,m-i) \text{ and } M \in \mathcal{T}_{\lambda\mathtt{ws}} \\ \text{or} \\ i+j \leq m \text{ and } \mathcal{H}(N,m-i) \text{ and } \mathcal{H}(M,m-j+1) \end{cases} \\
\mathcal{H}(M\llbracket i/N,j\rrbracket,m) & \text{iff} & i = m,\ M,N \in \mathcal{T}_{\lambda\mathtt{ws}},\ N \in SN_{\lambda\mathtt{ws}} \text{ and } \mathcal{B}(M,m)
\end{array}
$$

Example 4.33 Let $M = 4[0/\langle 0\rangle(\lambda 00)\ \langle 0\rangle(\lambda 00),1]$. Then $4\llbracket 0/\langle 0\rangle(\lambda 00)\ \langle 0\rangle(\lambda 00),1\rrbracket$ is not well-labelled since the **SN body** condition fails.

Let $N = 4[2/00,0][0/\lambda(00),0]$. Then the term $4[2/00,0]\llbracket 0/\lambda(00),0\rrbracket$ is not well-labelled since **safe propagation** fails. Indeed, the labelled substitution may not be computed by $\underline{\mathtt{ws}}$-rewriting because it is 'blocked' by the innermost substitution.

Let $O = 4[0/00,0][0/\lambda(00),0]$. Then $4\llbracket 0/00,0\rrbracket[0/\lambda(00),0]$ is not well labelled since the **non-interaction** condition fails: $O \rightarrow_{c_1} 4[0/00[0/\lambda 00,0],1]$. However, $4[0/00,0]\llbracket 0/\lambda(00),0\rrbracket$ is well-labelled.

Now that we have restricted the terms and singled out the 'good' ones we continue by defining an appropriate unlabelling function and relating labelled rewriting to unlabelled rewriting, as done for $\lambda\mathtt{x}$ in the previous chapter.

Definition 4.34 (Unlabelling for $\mathcal{T}_{\underline{\lambda\mathtt{ws}}}$) We define $\lfloor\bullet\rfloor : \mathcal{T}_{\underline{\lambda\mathtt{ws}}} \to \mathcal{T}_{\lambda\mathtt{ws}}$ given by

$$
\begin{array}{lll}
\lfloor n\rfloor & \stackrel{\text{def}}{=} & n \\
\lfloor(MN)\rfloor & \stackrel{\text{def}}{=} & \lfloor M\rfloor\lfloor N\rfloor \\
\lfloor\lambda(M)\rfloor & \stackrel{\text{def}}{=} & \lambda(\lfloor M\rfloor) \\
\lfloor\langle k\rangle M\rfloor & \stackrel{\text{def}}{=} & \langle k\rangle\lfloor M\rfloor \\
\lfloor M[i/P,j]\rfloor & \stackrel{\text{def}}{=} & \lfloor M\rfloor[i/\lfloor P\rfloor,j] \\
\lfloor M\llbracket i/P,j\rrbracket\rfloor & \stackrel{\text{def}}{=} & M[i/P,j]
\end{array}
$$

Note that in the last clause it is unnecessary to define $\lfloor M\llbracket i/P,j\rrbracket\rfloor$ as $\lfloor M\rfloor[i/\lfloor P\rfloor,j]$ since the term $M\llbracket i/P,j\rrbracket$ requires that M, $P \in \mathcal{T}_{\lambda\mathtt{ws}}$ (so in this case $\lfloor M\rfloor = M$ and $\lfloor P\rfloor = P$).

Having defined labelled and unlabelled $\lambda\mathtt{ws}$-rewriting the following lemma relates these notions.

Lemma 4.35 (Unlabelling rewrite projection and lifting) Let $M,N,Q \in \mathcal{T}_{\underline{\lambda\mathtt{ws}}}$ and $P' \in WL$. The following diagrams hold:

$$\begin{tikzcd}
M \arrow[r, "\underline{\mathtt{ws}}"'] \arrow[d, dotted, "\lfloor\bullet\rfloor"'] & N \arrow[d, dotted, "\lfloor\bullet\rfloor"'] & M \arrow[r, "\underline{\lambda\mathtt{ws}}"'] \arrow[d, dotted, "\lfloor\bullet\rfloor"'] & N \arrow[d, dotted, "\lfloor\bullet\rfloor"'] & P' \arrow[r, dotted, "\underline{\lambda\mathtt{ws}}"'] \arrow[d, "\lfloor\bullet\rfloor"'] & Q' \arrow[d, dotted, "\lfloor\bullet\rfloor"'] \\
\lfloor M\rfloor \arrow[r, dotted, "\mathtt{p}"'] & \lfloor N\rfloor & \lfloor M\rfloor \arrow[r, dotted, "\lambda\mathtt{ws}"'] & \lfloor N\rfloor & P \arrow[r, "\lambda\mathtt{ws}"'] & Q \in WL
\end{tikzcd}$$

(a) (b) (c)

See [ABR00] for the proof.

Lemma 4.36 (Properties of well-labelled terms) 1. If $M \in NF_{\mathtt{p}}$ (i.e. M is a pure term) then $\mathcal{B}(M, m)$ holds for all $m \geq 0$.

2. If $M \in \mathcal{T}_{\lambda\mathtt{ws}}$ (i.e. M contains no labelled substitution) then $\mathcal{H}(M, m)$ holds for all $m \geq 0$.

3. If $M \in WL$ and $N \subseteq M$ then $N \in WL$.

4. The set WL is closed under $\underline{\lambda\mathtt{ws}}$-rewriting; i.e. for all $M, N \in \mathcal{T}_{\underline{\lambda\mathtt{ws}}}$, if $M \in WL$ and $M \rightarrow_{\underline{\lambda\mathtt{ws}}} N$, then $N \in WL$.

As in the $\underline{\lambda\mathtt{x}}$-calculus labelled substitutions may be eliminated by executing them (Lemma 4.37) or by erasing them (Def. 4.34).

Lemma 4.37 (Properties of $\underline{\mathtt{ws}}$) The $\underline{\mathtt{ws}}$-rewrite system is strongly normalizing and confluent on the set of well-labelled terms. The $\underline{\mathtt{ws}}$-normal forms of well-labelled terms do not contain occurrences of the labelled substitution operator, i.e. if $M \in WL$ then $\underline{\mathtt{ws}}(M) \in \mathcal{T}_{\lambda\mathtt{ws}}$.

Definition 4.38 (The $\underline{\lambda\mathtt{ws}}^i$ and $\lambda\mathtt{ws}^e$-rewrite systems) Reduction in the $\underline{\lambda\mathtt{ws}}$-rewrite system may be partitioned into reduction in the $\underline{\lambda\mathtt{ws}}^i$ and $\lambda\mathtt{ws}^e$-rewrite systems, i.e. $\underline{\lambda\mathtt{ws}} = \underline{\lambda\mathtt{ws}}^i \uplus \lambda\mathtt{ws}^e$, where :

1. The $\underline{\lambda\mathtt{ws}}^i$-rewrite system is $\lambda\mathtt{ws}$-rewriting in the bodies of labelled substitutions (i.e. the contextual closure of $\rightarrow_R$: if $M \rightarrow_{\lambda\mathtt{ws}} N$ then $P[i/M, j] \rightarrow_R P[i/N, j]$) together with $\underline{\mathtt{ws}}$-rewriting.

2. The $\lambda\mathtt{ws}^e$-rewrite system is $\lambda\mathtt{ws}$-rewriting over $\mathcal{T}_{\underline{\lambda\mathtt{ws}}}$ *except inside the bodies of labelled substitution operators.*

Lemma 4.39 (labelled rewrite projection) Let M be a well-labelled term. The following diagrams hold:

$$\begin{tikzcd}
M \arrow[r, "\underline{\lambda\mathtt{ws}}^i"'] \arrow[d, "\underline{\mathtt{ws}}"', two heads] & N \arrow[d, "\underline{\mathtt{ws}}", two heads] \\
\underline{\mathtt{ws}}(M) \arrow[r, dotted, two heads, "\lambda\mathtt{ws}"'] & \underline{\mathtt{ws}}(N)
\end{tikzcd}
\qquad
\begin{tikzcd}
M \arrow[r, "\lambda\mathtt{ws}^e"'] \arrow[d, "\underline{\mathtt{ws}}"', two heads] & N \arrow[d, "\underline{\mathtt{ws}}", two heads] \\
\underline{\mathtt{ws}}(M) \arrow[r, dotted, "+", "\lambda\mathtt{ws}"'] & \underline{\mathtt{ws}}(N)
\end{tikzcd}$$

(1) (2)

The following key result analogous to Corollary 4.12 in the setting of $\lambda\mathtt{x}$ is formulated in [DG01]. Its proof follows from Lemma 4.39 and the fact that $\underline{\lambda\mathtt{ws}}^i$-rewriting is strongly normalizing on the set of well-labelled terms [DG01, Lemma 8.19]

Corollary 4.40 (Perpetuality of labelled rewriting) Let $M \in WL$. If M has an infinite $\underline{\lambda\mathtt{ws}}$-derivation then $\underline{\mathtt{ws}}(M)$ has an infinite $\lambda\mathtt{ws}$-derivation.

Note how, in Corollary 4.40, the set of terms is restricted to the well-labelled terms. Compare this with Corollary 4.12 for $\lambda\mathtt{x}$ where the full set of labelled terms is considered.

4.2.2 The Labelling Strategy for $\lambda\mathtt{ws}$

We shall now study how to replay the analysis behind the $\mathtt{Zx}(\bullet)$ $\lambda\mathtt{x}$-perpetual strategy in the $\lambda\mathtt{ws}$-calculus. We shall first begin by introducing the zoom-in function which selects the minimal subterm where $\mathtt{Zws}(\bullet)$ shall find its redex. If the leftmost redex of the resulting term is a $b1$ or $b2$-redex then all works smoothly. However, if this is not the case then we shall be in the need of introducing a labelling strategy.

Definition 4.41 (Decent $\lambda\mathtt{ws}$-terms) We define the set of *decent terms* $\mathcal{T}^{snb}_{\lambda\mathtt{ws}} \subset \mathcal{T}_{\lambda\mathtt{ws}}$ as: $M \in \mathcal{T}^{snb}_{\lambda\mathtt{ws}}$ if the bodies of the substitutions in M are strongly $\lambda\mathtt{ws}$-normalizing (i.e. if $N_1[i/N_2, j] \subseteq M$ then $N_2 \in SN_{\lambda\mathtt{ws}}$).

Note that contrary to the x-calculus (Lemma 4.21), the set $\mathcal{T}^{snb}_{\lambda\mathtt{ws}}$ is not closed under ws-reduction nor p-reduction as explained when introducing the well-labelled terms (also see Section 4.2.4).

Definition 4.42 (Zoom-in function for λws) Let $M \in \mathcal{T}_{\lambda ws}$ such that $\infty_{\lambda ws}(M)$. We define $\mathtt{Vws}(M)$ by induction on M as:

$$\begin{array}{lcll}
\mathtt{Vws}(\langle\vec{k}\rangle n\vec{P}Q\vec{R}) & \overset{\text{def}}{=} & \mathtt{Vws}(Q) & \text{if } \vec{P} \in SN_{\lambda ws}, Q \notin SN_{\lambda ws} \\
\mathtt{Vws}(\langle\vec{k}\rangle(\lambda P)) & \overset{\text{def}}{=} & \mathtt{Vws}(P) & \\
\mathtt{Vws}((\langle\vec{k}\rangle\lambda P)Q\vec{R}) & \overset{\text{def}}{=} & \mathtt{Vws}(P) & \text{if } P \notin SN_{\lambda ws} \\
\mathtt{Vws}((\langle\vec{k}\rangle\lambda P)\vec{Q}S\vec{R}) & \overset{\text{def}}{=} & \mathtt{Vws}(S) & \text{if } P, \vec{Q} \in SN_{\lambda ws}, S \notin SN_{\lambda ws} \\
\mathtt{Vws}((\langle\vec{k}\rangle\lambda P)Q\vec{R}) & \overset{\text{def}}{=} & (\langle\vec{k}\rangle\lambda P)Q\vec{R} & \text{if } P, Q, \vec{R} \in SN_{\lambda ws} \\
 & & & \\
\mathtt{Vws}(\langle\vec{k}\rangle(P[i/Q,j])\vec{R}) & \overset{\text{def}}{=} & \mathtt{Vws}(P) & \text{if } P \notin SN_{\lambda ws} \\
\mathtt{Vws}(\langle\vec{k}\rangle(P[i/Q,j])\vec{R}) & \overset{\text{def}}{=} & \mathtt{Vws}(Q) & \text{if } P \in SN_{\lambda ws}, Q \notin SN_{\lambda ws} \\
\mathtt{Vws}(\langle\vec{k}\rangle(P[i/Q,j])\vec{R_1}S\vec{R_2}) & \overset{\text{def}}{=} & \mathtt{Vws}(S) & \text{if } P, Q, \vec{R_1} \in SN_{\lambda ws}, S \notin SN_{\lambda ws} \\
\mathtt{Vws}(\langle\vec{k}\rangle(P[i/Q,j])\vec{R}) & \overset{\text{def}}{=} & \langle\vec{k}\rangle(P[i/Q,j])\vec{R} & \text{if } P, Q, \vec{R} \in SN_{\lambda ws} \\
 & & & \\
\mathtt{Vws}(\langle\vec{k}^+\rangle(PQ)\vec{R}) & \overset{\text{def}}{=} & \mathtt{Vws}(P) & \text{if } P \notin SN_{\lambda ws} \\
\mathtt{Vws}(\langle\vec{k}^+\rangle(PQ)\vec{R}) & \overset{\text{def}}{=} & \mathtt{Vws}(Q) & \text{if } P \in SN_{\lambda ws} \text{ and } Q \notin SN_{\lambda ws} \\
\mathtt{Vws}(\langle\vec{k}^+\rangle(PQ)\vec{R_1}S\vec{R_2}) & \overset{\text{def}}{=} & \mathtt{Vws}(S) & \text{if } P, Q, \vec{R_1} \in SN_{\lambda ws}, S \notin SN_{\lambda ws} \\
\mathtt{Vws}(\langle\vec{k}^+\rangle(PQ)\vec{R}) & \overset{\text{def}}{=} & \langle\vec{k}^+\rangle(PQ)\vec{R} & \text{if } P, Q, \vec{R} \in SN_{\lambda ws}
\end{array}$$

Lemma 4.43 (Properties of $\mathtt{Vws}(\bullet)$) For all $M \in \mathcal{T}_{\lambda ws}$ such that $\infty_{\lambda ws}(M)$ we have:

1. $\mathtt{Vws}(M) = ((\langle\vec{k}\rangle\lambda P)Q\vec{R}$ or $\mathtt{Vws}(M) = \langle\vec{k}\rangle(P[i/Q,j])\vec{R}$ or $\mathtt{Vws}(M) = \langle\vec{k}^+\rangle(PQ)\vec{R}$ for some $P, Q, \vec{R} \in SN_{\lambda ws}$ (hence $\mathtt{Vws}(M) \in \mathcal{T}^{snb}_{\lambda ws}$).
2. $\mathtt{Vws}(M) \subseteq M$.
3. $\infty_{\lambda ws}(\mathtt{Vws}(M))$.

In the case that the zoom-in function yields $\langle\vec{k}^+\rangle(PQ)\vec{R}$ we follow [DG01] and use the context notation below to access its leftmost redex.

Definition 4.44 (Body context) The set of *body contexts* is given by the grammar: $B ::= \Box \mid \langle k\rangle(B) \mid BM$ where $\Box$ denotes a hole. We use letters $B, B', \ldots$ to denote body contexts. The *depth* of a body context B, denoted $\mathtt{depth}(B)$, is defined as: $\mathtt{depth}(\Box)=0$, $\mathtt{depth}(\langle k\rangle(B')) = k + \mathtt{depth}(B'), \mathtt{depth}(B'M) = \mathtt{depth}(B')$.

Lemma 4.45 If $M = (\langle\vec{k}^+\rangle(PQ))\vec{R}$ for $P, Q, \vec{R} \in SN_{\lambda ws}$ then either

- $M = B[(\langle\vec{k'}\rangle\lambda P')Q']$ for some body context B and $P', Q', \vec{k'}$, or
- $M = B[\langle\vec{k'}\rangle(P'[i/Q',j])]$ for some body context B and $P', Q', i, j, \vec{k'}$.

Moreover, the body context B in both items is unique.

Proof. We prove the following more general result: if $N = \langle\vec{k}\rangle(\langle\vec{l}\rangle PQ)\vec{R}$ where $\infty_{\lambda ws}(N)$, the outermost symbol of P is not an update tag (i.e. $P \neq \langle l'\rangle S$ for some $l' \geq 0$ and λws-term S) and $P, Q, \vec{R} \in SN_{\lambda ws}$ then either

- $N = B[(\langle\vec{k'}\rangle\lambda P')Q']$ for some body context B and $P', Q', \vec{k'}$, or
- $N = B[\langle\vec{k'}\rangle(P'[i/Q',j])]$ for some body context B and $P', Q', i, j, \vec{k'}$.

The proof is by structural induction on P. Note that P must be of one of the following forms:

- $P = n$. Then we contradict the assumption that $\infty_{\lambda ws}(M)$ so this case holds trivially.
- $P = P_1P_2$. Then $N = \langle\vec{k}\rangle(\langle\vec{l}\rangle(P_1P_2)Q)\vec{R}$. The induction hypothesis may be applied to the subterm of N: $\langle\vec{l}\rangle(P_1P_2)Q$ where $\vec{l}$ replaces $\vec{k}$, P_1 replaces P, P_2 replaces Q, and Q replaces $\vec{R}$. Thus two situations may arise:

- $\langle\vec{l}\rangle(P_1P_2)Q = B'[(\langle\vec{k''}\rangle\lambda P'')Q'']$ for some $P'', Q'', \langle\vec{k''}\rangle$.
- $\langle\vec{l}\rangle(P_1P_2)Q = B'[\langle\vec{k''}\rangle(P''[i'/Q'', j'])]$ for some $P'', Q'', i', j', \langle\vec{k''}\rangle$.

In either case let $B'' = \langle\vec{k}\rangle(\Box)\vec{R}$ and define $B = B''[B'[]]$ and conclude.

- $P = \lambda P_1$. Then by taking $B = \langle\vec{k}\rangle(\Box)\vec{R}$ we may conclude directly.
- $P = P_1[i/P_2, j]$. Then by taking $B = \langle\vec{k}\rangle(\Box Q)\vec{R}$ we may conclude directly.

. Uniqueness of B follows by contruction. •

Consider the following problem: given a non-pure λws-term M, is it always possible to label some substitution operator in M such that the resulting term is well-labelled? By Remark 4.31 there are at least three reasons why this is not necessarily true. The term $4[0/(\lambda 00)(\lambda 00), 1]$ is an example since the body of the only substitution is not a strongly λws-normalizing term, so the **SN body** condition fails. We may refine the question further and ask: given a *decent* λws-term M, is it always possible to label some substitution operator in M such that the resulting term is well-labelled? The answer is, once again, no. Consider for example the term $M = (\lambda(0[0/0, 0]))1$; here the problem is that the substitution operator is not 'interaction-free' (hence **non-interaction** shall fail) in the following sense: a $b1$ step may create a substitution that is composable with the one under the λ in M, possibly creating a source of an infinite derivation.

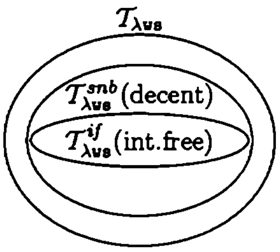
We shall prove the following result: if a decent λws-term M is *interaction-free* (Def. 4.47) then it admits a well-labelling, i.e. there is some substitution in M that can be labelled such that the resulting term is a well-labelled term. In the case that M has an infinite λws-derivation Lemma 4.35(c) and Lemma 4.40 shall complete our labelling strategy. The set of decent interaction-free terms shall constitute the domain of the labelling strategy.

Definition 4.46 (Substitution occurrences) Let $M \in \mathcal{T}_{\lambda ws}$. We shall use $\mathcal{SO}(M)$ to denote the set of positions of substitution operators in M.

Definition 4.47 (Interaction-free substitutions and terms) Let $M \in \mathcal{T}_{\lambda ws}$, $p \in \mathcal{SO}(M)$ and $m \geq 0$. The substitution at occurrence p is an *interaction-free substitution of level m in M* if $IF(M, m, p)$ holds, where $IF(\bullet, \bullet, \bullet)$ is defined as:

$$\begin{array}{lll}
IF(\lambda N, m, 1.p) & \text{if} & IF(N, m+1, p) \\
IF(N_1N_2, m, 1.p) & \text{if} & IF(N_1, m, p) \\
IF(N_1N_2, m, 2.p) & \text{if} & IF(N_2, m, p) \\
IF(\langle i\rangle N, m, 1.p) & \text{if} & i \leq m \text{ and } IF(N, m-i, p) \\
IF(N_1[i/N_2, j], m, 1.p) & \text{if} & i+j \leq m \text{ and } IF(N_1, m-j+1, p) \\
IF(N_1[i/N_2, j], m, 2.p) & \text{if} & i \leq m \text{ and } IF(N_2, m-i, p) \\
IF(N_1[i/N_2, j], m, \epsilon) & \text{if} & m = i
\end{array}$$

We say $M \in \mathcal{T}_{\lambda ws}$ is an *interaction-free term* when $IF(M, m, p)$ holds for some $p \in \mathcal{SO}(M)$ and $m \geq 0$. Also, we use $\mathcal{T}^{if}_{\lambda ws} \subset \mathcal{T}^{snb}_{\lambda ws}$ to denote the set of decent λws-terms that are interaction-free.


$\mathcal{T}_{\lambda ws}$
$\mathcal{T}^{snb}_{\lambda ws}$(decent)
$\mathcal{T}^{if}_{\lambda ws}$(int.free)

The intuition behind interaction-free substitutions follows the lines of Remark 4.31. Consider a decent non-pure term M. If some substitution is labelled in M the resulting term shall be well-labelled if the remaining two conditions of Remark 4.31 are met: safe propagation and non-interaction. The interaction-free predicate of Def. 4.47 guarantees non-interaction. Furthermore, we shall see that this suffices for a decent term to admit a well-labelling, for the condition of safe propagation may always be met (Proposition 4.51).

The following result formalizes the idea that if $\mathcal{B}(M, m)$ does not hold, for some term M and $m \geq 0$, then there is an interaction-free substitution blocking m-level substitutions, in other words, there is an interaction-free substitution with strictly higher level in M.

Lemma 4.48 Suppose $\mathcal{B}(M, m)$ does not hold. Then there is a $p \in \mathcal{SO}(M)$ and a level $k > m$ such that $IF(M, k, p)$ holds.

Proof. By induction on M.

- $M = n$. Holds vacuously.
- $M = M_1M_2$. Then since $\mathcal{B}(M, m)$ does not hold either $\mathcal{B}(M_1, m)$ or $\mathcal{B}(M_2, m)$ does not hold. Then we apply the induction hypothesis.
- $M = \lambda(N)$. Then it must be that $\mathcal{B}(N, m+1)$ does not hold. Then by induction hypothesis there is a $p' \in \mathcal{SO}(N)$ of level k' such that $IF(N, k', p')$ with $k' > m+1$. But then $IF(\lambda(N), k'-1, 1.p')$ holds and we are done.
- $M = \langle i \rangle N$. Then it must be that $i \leq m$ and $\mathcal{B}(N, m-i)$ does not hold (the case $m < i$ is not possible since $\mathcal{B}(M, m)$ would thus hold). The induction hypothesis yields a $p' \in \mathcal{SO}(N)$ and a level k' with $k' > m - i$ such that $IF(N, k', p')$. Then by Def. 4.47 we have $IF(M, k'+i, 1.p')$ with $k' + i > m$ and we are done.
- $M = M_1[i/N, j]$. Then since $\mathcal{B}(M, m)$ does not hold we have three cases to consider:
  1. $m < i$. Then $IF(M, i, \epsilon)$ holds trivially.
  2. $i \leq m < i+j$ and $\mathcal{B}(N, m-i)$ does not hold. Then by the induction hypothesis there is a $p' \in \mathcal{SO}(N)$ and a level k' with $k' > m - i$ such that $IF(N, k', p')$ holds. Then by Def. 4.47 (sixth clause) also $IF(M_1[i/N, j], k'+i, 2.p')$ holds where $k' + i > m$.
  3. $i + j \leq m$ and two further subcases must be considered:
     (a) $\mathcal{B}(N, m - i)$ does not hold. Then by the induction hypothesis there is a $p' \in \mathcal{SO}(N)$ and a level k' with $k' > m - i$ such that $IF(N, k', p')$ holds. Then by Def. 4.47 (sixth clause) also $IF(M_1[i/N, j], k'+i, 2.p')$ holds where $k' + i > m$.
     (b) $\mathcal{B}(M_1, m-j+1)$ does not hold. Then by the induction hypothesis there is a $p' \in \mathcal{SO}(M_1)$ and a level k' with $k' > m-j+1$ such that $IF(M_1, k', p')$ holds. Then by Def. 4.47 (fifth clause) also $IF(M_1[i/N, j], k'+j-1, 1.p')$ holds where $k'+j-1 > m$. Note that $k'+j-1 > m \geq i+j$.

∎

Let $\underline{M}_p$ denote the term obtained from M by labelling the substitution operator at position p.

Lemma 4.49 Let $M \in \mathcal{T}_{\lambda ws}$. Let $p \in \mathcal{SO}(M)$ with $M|_p = N_1[i/N_2, j]$. Suppose furthermore that:

1. $N_2 \in \mathcal{SN}_{\lambda ws}$,
2. $\mathcal{B}(N_1, i)$, and
3. $IF(M, m, p)$ for some $m \geq 0$.

Then $\mathcal{H}(\underline{M}_p, m)$ holds.

Proof. By induction on M.

- $M = n$. Holds vacuously.
- $M = M_1M_2$. Suppose $p = 1.p'$ with $p' \in \mathcal{SO}(M_1)$ (the case $p = 2.p'$ with $p' \in \mathcal{SO}(M_2)$ is similar). Then by the induction hypothesis $\mathcal{H}(\underline{M_1}, m)$ holds. Also, since $M_2 \in \mathcal{T}_{\lambda ws}$ then $\mathcal{H}(M_2, m)$ holds (Lemma 4.36)(2). Thus since $\underline{M} = \underline{M_1}M_2$ we may conclude $\mathcal{H}(\underline{M}, m)$.
- $M = \lambda(N)$. Then $p = 1.p'$ with $p' \in \mathcal{SO}(N)$ and we must verify that $\mathcal{H}(\underline{N}, m+1)$ holds which follows from the induction hypothesis. Note that the induction hypothesis may be applied since $IF(N, m+1, p')$ follows from hypothesis 3.

- $M = \langle i \rangle N$. Note that since $IF(\langle i \rangle N, m, p)$ holds, we must have that $p = 1.p'$ with $p' \in \mathcal{SO}(N)$ and $m \geq i$ and $IF(N, m-i, p')$. We must thus verify that $\mathcal{H}(\underline{N}, m-i)$ holds. For this we apply the induction hypothesis taking N for M, p' for p and $m-i$ for m.

- $M = N[i'/P, j']$. We have three further cases to consider depending on the form of p:

  - $p = 1.p'$ with $p' \in \mathcal{SO}(N)$. Then since $IF(N[i'/P, j'], m, 1.p')$ holds by hypothesis 3 we have $i' + j' \leq m$ and $IF(N, m - j' + 1, p')$ holds. By induction hypothesis $\mathcal{H}(\underline{N}, m - j' + 1)$ holds. Also, since $P \in \mathcal{T}_{\lambda ws}$ the by Lemma 4.36(2) we have that $\mathcal{H}(P, m - i')$ holds. Thus by Def. 4.32 we conclude that $\mathcal{H}(\underline{M}, m)$ holds.
  - $p = 2.p'$ with $p' \in \mathcal{SO}(P)$. Then since $IF(N[i'/P, j'], m, 2.p')$ holds by hypothesis 3 we have $i' \leq m$ and $IF(P, m-i', p')$ holds. By induction hypothesis $\mathcal{H}(\underline{P}, m-i')$ holds. Now we consider two further subcases:
    1. $m < i' + j'$. Then since $N \in \mathcal{T}_{\lambda ws}$ we have $\mathcal{H}(\underline{M}, m)$.
    2. $m \geq i' + j'$. Then since $N \in \mathcal{T}_{\lambda ws}$ by Lemma 4.36(2) we have $\mathcal{H}(N, m - j' + 1)$. Then by Def. 4.32 we may conclude $\mathcal{H}(\underline{M}, m)$.
  - $p = \epsilon$ and hence $N[i'/P, j'] = N_1[i/N_2, j]$ and $m = i$. Then $\mathcal{H}(N_1[i/N_2, j], m)$ holds by hypothesis 1 and 2.

■

The following lemma states that if a substitution at position p with target N_1 is interaction-free in M, and N_1 itself has an interaction-free substitution at position q, then the latter is interaction-free in M too.

Lemma 4.50 Suppose that $M \in \mathcal{T}_{\lambda ws}$ and that there exist $p, q \in \mathcal{SO}(M)$, $k, l \geq 0$ such that

1. $M|_p = N_1[i_1/P_1, j_1]$ with $IF(M, k, p)$,
2. $N_1|_q = N_2[i_2/P_2, j_2]$ with $IF(N_1, l, q)$, and
3. $l > i_1$.

Then $IF(M, k + e + j_1, p.1.q)$ where $e = l - i_1 - 1$.

Proof. By induction on the length of p.

- $p = \epsilon$. Then $M = N_1[i_1/P_1, j_1]$ and $k = i_1$. Now $IF(N_1[i_1/P_1, j_1], i_1 + e + j_1, 1.q)$ iff $i_1 + e + j_1 \geq i_1 + j_1$ and $IF(N_1, i_1 + e + 1, q)$. Since $i_1 + e + 1 = l$ then by hypothesis 2 we are done.

- $p = 1.p'$. Then we must consider each possible M:

  - $M = \lambda(M')$. Then $IF(\lambda(M'), k+e+j_1, 1.p'.1.q)$ iff $IF(M', k+e+j_1+1, p'.1.q)$. But by hypothesis 1 we know $IF(M', k+1, p')$ holds. Then by induction hypothesis we have that $IF(M', k+1+e+j_1, p'.1.q)$ holds and we are done. Note that e remains unaltered since it depends solely on l and i_1 which remain unaltered when the induction hypothesis is applied.
  - $M = M_1 M_2$. Then $IF(M_1 M_2, k + e + j_1, 1.p'.1.q)$ iff $IF(M_1, k + e + j_1, p'.1.q)$. But by hypothesis 1 we know $IF(M_1, k, p')$ holds. Then by induction hypothesis we have that $IF(M_1, k + e + j_1, p'.1.q)$ holds and we are done.
  - $M = \langle i \rangle M'$. Then $IF(\langle i \rangle M', k + e + j_1, 1.p'.1.q)$ iff $k + e + j_1 \geq i$ and $IF(M', k + e + j_1 - i, p'.1.q)$. But by hypothesis 1 we know $k \geq i$ and $IF(M', k - i, p')$ holds. Thus $k + e + j_1 \geq i$. We are left to verify that $IF(M', k + e + j_1 - i, p'.1.q)$ holds, a result that follows by induction hypothesis.
  - $M = N'[i'/P', j']$. Then $IF(N'[i'/P', j'], k + e + j_1, 1.p'.1.q)$ iff $i' + j' \leq k + e + j_1$ and $IF(N', k + e + j_1 - j' + 1, p'.1.q)$. But by hypothesis 1 we know $k \geq i' + j'$ and $IF(N', k - j' + 1, p')$ holds. Thus $k + e + j_1 \geq i' + j'$. We are left to verify that $IF(N', k + e + j_1 - j' + 1, p'.1.q)$ holds. But by induction hypothesis we have that $IF(N', k - j' + 1 + e + j_1, p'.1.q)$ holds and we are done.

- $p = 2.p'$. Here we have two further cases to consider:

- $M = M_1 M_2$. Then $IF(M_1 M_2, k+e+j_1, 2.p'.1.q)$ iff $IF(M_2, k+e+j_1, p'.1.q)$. But by hypothesis 1 we know $IF(M_2, k, p')$ holds. Then by induction hypothesis we have that $IF(M_2, k+e+j_1, p'.1.q)$ holds and we are done.
- $M = N'[i'/P', j']$. Then $IF(N'[i'/P', j'], k+e+j_1, 2.p'.1.q)$ iff $i' \leq k+e+j_1$ and $IF(P', k+e+j_1-i', p'.1.q)$. But by hypothesis 1 we know $k \geq i'$ and $IF(P', k-i', p')$ holds. Thus $k+e+j_1 \geq i'$. We are left to verify that $IF(P', k+e+j_1-i', p'.1.q)$ holds. But by induction hypothesis we have that $IF(P', k-i'+e+j_1, p'.1.q)$ holds and we are done.

■

Proposition 4.51 If a decent $\lambda\mathtt{ws}$-term M is interaction-free then it admits a well-labelling, i.e. there is some substitution in M that can be labelled such that the resulting term is a well-labelled term.

Proof. Let $p \in \mathcal{SO}(M)$ with $M|_p = N[i/P, j]$ and such that $IF(M, m, p)$. We shall use induction on the number of substitution operators n occurring in N.

- $n = 0$. By Lemma 4.36(1) $\mathcal{B}(N, i)$ holds. Lemma 4.49 concludes the case.
- $n > 0$. If $\mathcal{H}(\underline{M}_p, m)$ holds then we are done. Otherwise, Lemma 4.49 reveals that $\mathcal{B}(N, i)$ does not hold. Lemma 4.48 then yields $q \in \mathcal{SO}(N)$ and $l > i$ such that $N|_q = N'[i'/P', j']$ and $IF(N, l, q)$. Now by Lemma 4.50 we have $IF(M, m+e+j, p.1.q)$ where $e \overset{\text{def}}{=} l-i-1$. Then we may apply the inductive hypothesis and conclude the case.

■

Definition 4.52 (Substitution labelling strategy for $\lambda\mathtt{ws}$) The substitution labelling strategy for $\lambda\mathtt{ws}$ is given by the (many-step) $\mathtt{ws}$-reduction strategy $\mathtt{Lws}(\bullet) : \mathcal{T}_{\lambda\mathtt{ws}} \longrightarrow \mathcal{T}_{\lambda\mathtt{ws}}$ defined as

$$\mathtt{Lws}(M) \overset{\text{def}}{=} \begin{cases} M & \text{if } M \notin \mathcal{T}^{if}_{\lambda\mathtt{ws}} \\ \underline{\mathtt{ws}}(\underline{M}_{\mathtt{labelSubs}(M,p)}) & \text{otherwise} \end{cases}$$

where $p \in \mathcal{SO}(M)$ is the outermost-leftmost interaction-free substitution operator in M. The domain of the labelling strategy is defined as the set $\mathcal{T}^{if}_{\lambda\mathtt{ws}}$.

The *labelling algorithm* $\mathtt{labelSubs}(\bullet, \bullet)$ is an algorithm that selects the appropriate substitution to label in order to guarantee that the resulting term, once labelled, is a well-labelled term. It shall build on the results developed above. Let $M \in \mathcal{T}^{if}_{\lambda\mathtt{ws}}$ and $p \in \mathcal{SO}(M)$ with $M|_p = N_0[i_0/P_0, j_0]$ and $IF(M, k_0, p)$. The algorithm $\mathtt{labelSubs}(\bullet, \bullet)$ on the input (M, p) is defined in Figure 4.1.

Example 4.53 Consider the term $N = ((\lambda(0[2/1,0]))3)[0/2,0] \in \mathcal{T}^{if}_{\lambda\mathtt{ws}}$. Then $\mathtt{labelSubs}(N, \epsilon)$ returns 1.1.1, i.e. the occurrence of the inner substitution. Indeed, the term $((\lambda(0[2/1,0]))3)\mathbf{[}0/2,0\mathbf{]}$ is not well-labelled (safe propagation fails). However, $((\lambda(0\mathbf{[}2/1,0\mathbf{]}))3)[0/2,0]$ is well-labelled.

For the sake of comparison with the results already introduced for $\lambda\mathtt{x}$ (Section 4.1.3), if $M \in \mathcal{T}^{if}_{\lambda\mathtt{ws}}$ then we shall abbreviate $\underline{M}_{\mathtt{labelSubs}(M,p)}$, where $p \in \mathcal{SO}(M)$ is the outermost-leftmost interaction-free substitution operator in M, by $\underline{M}$.

Note that $\mathcal{T}^{if}_{\lambda\mathtt{ws}}$ is not closed under $\mathtt{Lws}(\bullet)$-reduction. Let $M = ((\lambda 0[0/0,0])1)[2/1,0]$. Then $M \in \mathcal{T}^{if}_{\lambda\mathtt{ws}}$ since the outermost substitution is interaction-free yet $\mathtt{Lws}(M) = (\lambda 0[0/0,0])1 \notin \mathcal{T}^{if}_{\lambda\mathtt{ws}}$. Moreover, $\mathtt{Lws}(M)$ may not even be decent for $M \in \mathcal{T}^{if}_{\lambda\mathtt{ws}}$: indeed, $M = 4[0/00,0][0/\lambda(00),0] \in \mathcal{T}^{if}_{\lambda\mathtt{ws}}$, yet $\mathtt{Lws}(M) = 4[0/\langle 0\rangle(\lambda 00)\langle 0\rangle(\lambda 00),1] \notin \mathcal{T}^{snb}_{\lambda\mathtt{ws}}$.

Lemma 4.54 (Properties of $\mathtt{Lws}(\bullet)$) 1. $\mathtt{Lws}(\bullet)$ is a (many-step) p-(hence $\mathtt{ws}$-) rewrite strategy.

2. $\mathtt{Lws}(\bullet)$ is $\lambda\mathtt{ws}$-perpetual.

Proof. For the first item suppose $M \in \mathcal{T}^{if}_{\lambda\mathtt{ws}}$, since $\underline{M} \xrightarrow{+}_{\underline{\mathtt{ws}}} \underline{\mathtt{ws}}(\underline{M})$, then by Lemma 4.35(a) we have $M = \lfloor \underline{M} \rfloor \xrightarrow{+}_{\mathtt{p}} \lfloor \underline{\mathtt{ws}}(\underline{M}) \rfloor = \mathtt{ws}(\underline{M})$. For the second item suppose there is an infinite $\lambda\mathtt{ws}$-derivation starting from M, then by Lemma 4.35(c) there is an infinite $\underline{\lambda\mathtt{ws}}$-derivation starting from $\underline{M}$. Corollary 4.40 concludes the item.

■

```
labelSubs $(M,p)$;
begin
  $r := 0$;
  $l := k_0$;
  while $\neg\mathcal{H}(\underline{M}_p, l)$ do --Termination guaranteed by proof of Proposition 4.51
  begin
     --Note that $IF(M,l,p)$ holds by Lemma 4.50--
     Lemma 4.49 reveals that $\neg\mathcal{B}(N_r, i_r)$
     Apply Lemma 4.48 and obtain $q_{r+1} \in \mathcal{SO}(N_r)$ and a level $k_{r+1} > i_r$ with
         $IF(N_r, k_{r+1}, q_{r+1})$. Let $N_r|_{q_{r+1}} = N_{r+1}[i_{r+1}/P_{r+1}, j_{r+1}]$
     $p := p.1.q_{r+1}$;
     $l := l + e + j_r$ where  $e = k_{r+1} - i_r - 1$;
     $r := r + 1$;
  end;
  return $p$;
end;
```

Figure 4.1: The labelling algorithm

4.2.3 The Zws(•) Perpetual Strategy for λws

We are almost in condition of defining the zoom-in reduction strategy for λws. A final remark related to the use of the Lws(•) by the Zws(•) strategy is required. We have to guarantee that the term to which Lws(•) is applied is in the domain of this labelling strategy. Since Zws(•) shall recur to the labelling strategy twice we must verify that this condition is met.

Remark 4.55 1. Suppose $M = \langle\vec{k}\rangle(P[i/Q,j])\vec{R}$ with $P, Q, \vec{R} \in SN_{\lambda ws}$ then $M \in \mathcal{T}^{if}_{\lambda ws}$. Indeed, first note that M is decent. Also, if $l = (\sum\vec{k}) + i$ where $\vec{R} = R_1 \ldots R_n$ and $\langle\vec{k}\rangle = \langle k_1\rangle \ldots \langle k_m\rangle$ then $IF(M, l, 1^{n+m})$ holds.

2. Suppose $M = B[\langle\vec{k}\rangle(P[i/Q,j])]$ with $P, Q, \vec{R} \in SN_{\lambda ws}$ where $\vec{R}$ is the set of arguments of applications in B, then $M \in \mathcal{T}^{if}_{\lambda ws}$. Indeed, note that M is decent. Also, if $l = \mathtt{depth}(B) + i + \sum\vec{k}$ and p is the position of the hole in the body context B and $\langle\vec{k}\rangle = \langle k_1\rangle \ldots \langle k_m\rangle$ then $IF(M, l, p.1^m)$.

Definition 4.56 (The Zws(•) strategy for λws) Let $M \in \mathcal{T}_{\lambda ws}$ and let C be the context such that $M = C[\mathtt{Vws}(M)]$. We define the (many-step) λws-strategy $\mathtt{Zws}(M)$ by $\mathtt{Zws}(M) \overset{\text{def}}{=} M$ if $M \in SN_{\lambda ws}$ otherwise $\mathtt{Zws}(M)$ is defined as:

$$\mathtt{Zws}(M) \overset{\text{def}}{=} \begin{cases} C[(P[0/Q, \sum\vec{k}]\,\vec{R}] & \text{if } \mathtt{Vws}(M) = (((\langle\vec{k}\rangle\lambda P)\,Q\,\vec{R} \\ C[\mathtt{Lws}(\langle\vec{k}\rangle(P[i/Q,j])\,\vec{R})] & \text{if } \mathtt{Vws}(M) = \langle\vec{k}\rangle(P[i/Q,j])\,\vec{R} \\ C[B[P'[0/Q', \sum\vec{k'}]]] & \text{if } \mathtt{Vws}(M) = \langle\vec{k}^{+}\rangle(P\,Q)\,\vec{R} \text{ and } \langle\vec{k}^{+}\rangle(PQ)\vec{R} = B[(\langle\vec{k'}\rangle\lambda(P'))Q'] \\ & \text{for some body context } B \text{ and } P, Q, \vec{R} \in SN_{\lambda ws} \\ C[\mathtt{Lws}(B[\langle\vec{k'}\rangle P'[i/Q',j]])] & \text{if } \mathtt{Vws}(M) = \langle\vec{k}^{+}\rangle(P\,Q)\,\vec{R} \text{ and } \langle\vec{k}^{+}\rangle(PQ)\vec{R} = B[\langle\vec{k'}\rangle P'[i/Q',j]] \\ & \text{for some body context } B \text{ and } P, Q, \vec{R} \in SN_{\lambda ws} \end{cases}$$

Lemma 4.57 (Zws(•) is a (many-step) λws-strategy) For all $M \in \mathcal{T}_{\lambda ws}$ we have $M \twoheadrightarrow_{\lambda ws} \mathtt{Zws}(M)$. Moreover, if $M \notin SN_{\lambda ws}$ then $M \xrightarrow{+}_{\lambda ws} \mathtt{Zws}(M)$.

Proof. If $M \in SN_{\lambda ws}$ then we are done. So let us assume that $M \notin SN_{\lambda ws}$. According to Lemma 4.28 we have the following cases to consider.

- $M = \langle\vec{k}\rangle n\vec{R}$. Then $\vec{R} = \vec{R_1}P\vec{R_2}$ where $\vec{R_1} \in SN_{\lambda ws}$ and $P \notin SN_{\lambda ws}$. And $\mathtt{Zws}(M) = \langle\vec{k}\rangle n\vec{R_1}\mathtt{Zws}(P)\vec{R_2}$. By the induction hypothesis we have $P \xrightarrow{+}_{\lambda ws} \mathtt{Zws}(P)$ and therefore $M \xrightarrow{+}_{\lambda ws} \mathtt{Zws}(M)$.
- $M = \langle\vec{k}\rangle\lambda P$. Then $P \notin SN_{\lambda ws}$ and we use the induction hypothesis.

- $M = ((\vec{k})\lambda P)Q\vec{R}$. If P or Q or $\vec{R} \notin SN_{\lambda ws}$ then we use the induction hypothesis. Otherwise, $M \twoheadrightarrow_m ((\sum \vec{k})\lambda(P))Q\vec{R} \rightarrow_{b2} P[0/Q, \sum k]\vec{R} = \mathtt{Zws}(M)$.
- $M = ((\vec{k})P[i/Q, j])\vec{R}$. If P or Q or $\vec{R} \notin SN_{\lambda ws}$ then we use the induction hypothesis. Otherwise, we use Lemma 4.54(1).
- $M = \langle\vec{k}^+\rangle(PQ)\vec{R}$. If P or Q or $\vec{R} \notin SN_{\lambda ws}$ then we use the induction hypothesis. Otherwise we have two cases to consider as dictated by Lemma 4.45:
  1. $M = B[((\vec{k'})\lambda P_1)Q_1]$ for some body context B. Then
  $$B[((\vec{k'})\lambda P_1)Q_1] \twoheadrightarrow_m B[((\sum \vec{k'})\lambda P_1)Q_1] \rightarrow_{b2} B[P_1[0/Q_1, \sum \vec{k'}]]$$
  2. $M = B[(\vec{k'})P_1[i/Q_1, j]]$ for some body context B. Then we use Lemma 4.54(1).

Proposition 4.58 The $\mathtt{Zws}(\bullet)$ reduction strategy is λws-perpetual.

Proof. Suppose M has an infinite λws-derivation. We prove by induction on M (using Lemma 4.28) that $\mathtt{Zws}(M)$ also has an infinite λws-derivation.

- $M = \langle\vec{k}\rangle n\vec{R}$. Then $\mathtt{Zws}(M) = \langle\vec{k}\rangle n\vec{R}_1\mathtt{Zws}(R')\vec{R}_2$ where $\vec{R} = \vec{R}_1R'\vec{R}_2$, $\vec{R}_1 \in SN_{\lambda ws}$, and $R' \notin SN_{\lambda ws}$. Then by induction hypothesis we are done.
- $M = \langle\vec{k}\rangle\lambda P$. Then $\mathtt{Zws}(M) = \langle\vec{k}\rangle\lambda(\mathtt{Zws}(P))$ and $P \notin SN_{\lambda ws}$. Thus we apply the induction hypothesis.
- $M = ((\vec{k})\lambda P)Q\vec{R}$. If $P, Q \notin SN_{\lambda ws}$ or there is some $R' \notin SN_{\lambda ws}$ in $\vec{R}$ then we apply $\mathtt{Zws}(\bullet)$ and apply the induction hypothesis. So suppose that $P, Q, \vec{R} \in SN_{\lambda ws}$. Then any infinite λws-derivation starting from M must have the form:
  $$((\vec{k})\lambda P)Q\vec{R} \twoheadrightarrow_{\lambda ws} \langle k'\rangle(\lambda P')Q'\vec{R'} \rightarrow_{B} P'[0/Q, k']\vec{R'} \twoheadrightarrow_{\lambda ws} \cdots$$
  where $k' = \sum \vec{k}$. Thus we may construct an infinite λws-derivation from $\mathtt{Zws}(M)$ as follows:
  $$P[0/Q, k']\vec{R} \twoheadrightarrow_{\lambda ws} P'[0/Q', k']\vec{R'} \twoheadrightarrow_{\lambda ws} \cdots$$
  and conclude the case.
- $M = ((\vec{k})P[i/Q, j])\vec{R}$. If $P, Q \notin SN_{\lambda ws}$ or there is some $R' \notin SN_{\lambda ws}$ in $\vec{R}$ then we apply $\mathtt{Zws}(\bullet)$ and apply the induction hypothesis. So suppose that $P, Q, \vec{R} \in SN_{\lambda ws}$. Now since there is an infinite λws-derivation starting from M then by Lemma 4.35(c) there is an infinite $\underline{\lambda ws}$-derivation starting from $\underline{M}$ where $\underline{M}$ is defined in Remark 4.55(1). Then by Lemma 4.40 we are done.
- $M = \langle\vec{k}^+\rangle(PQ)\vec{R}$. If $P, Q \notin SN_{\lambda ws}$ or there is some $R' \notin SN_{\lambda ws}$ in $\vec{R}$ then we apply $\mathtt{Zws}(\bullet)$ and apply the induction hypothesis. So suppose that $P, Q, \vec{R} \in SN_{\lambda ws}$. We consider two further cases depending on the form of M, as dictated by Lemma 4.45:
  - $M = B[((\vec{k'})\lambda(P_1))Q_1]$ for some body context B. Then any infinite λws-derivation starting from M must be of the form:
  $$B[((\vec{k'})\lambda(P_1))Q_1] \twoheadrightarrow_{\lambda ws} B'[((l)\lambda(P_1'))Q_1'] \rightarrow_{B} B'[P_1'[0/Q_1', l]] \twoheadrightarrow_{\lambda ws} \cdots$$
  where $l = \sum \vec{k'}$. Then we may construct the following infinite λws-derivation starting from $\mathtt{Zws}(M)$:
  $$B[P_1[0/Q_1, l]] \twoheadrightarrow_{\lambda ws} B'[P_1'[0/Q_1', l]] \twoheadrightarrow_{\lambda ws} \cdots$$
  - $M = B[(\vec{k'})P_1[i/Q_1, j]]$ for some body context B. Now since there is an infinite λws-derivation starting from M then by Lemma 4.35(c) there is an infinite $\underline{\lambda ws}$-derivation starting from $\underline{M}$ where $\underline{M}$ is defined in Remark 4.55(2). Then by Lemma 4.40 we are done.

4.2.4 Characterizing Terminating Terms in $\lambda\mathtt{ws}$

We shall now formulate an inductive characterization of the terms in $SN_{\lambda\mathtt{ws}}$. The proof needed for the characterization makes use of the perpetuality of the strategy $\mathtt{Zws}(\bullet)$.

Note that Bloo and Geuvers' characterization of $SN_{\lambda\mathtt{x}}$ (Proposition 2.40) does not adapt straightforwardly to the $\lambda\mathtt{ws}$-calculus. In other words, the inclusion $\{M \mid \forall N \subseteq M,\ \mathtt{ws}(N) \in SN_{\lambda\mathtt{w}}\} \subset SN_{\lambda\mathtt{ws}}$ does *not* hold. Consider the term $M = 4[0/00,0][0/\lambda(00),0]$. The subterms of M are $\{4, 0, 00, \lambda(00), 4[0/00,0], M\}$. Note that the $\mathtt{ws}$-normal form of each of these terms is a strongly $\lambda\mathtt{w}$-normalizing term, however, M is not a strongly $\lambda\mathtt{ws}$-normalizing term:

$$M \to_{c_1} 4[0/00[0/\lambda 00,0],1] \twoheadrightarrow_{\mathtt{p}} 4[0/\langle 0\rangle(\lambda 00)\langle 0\rangle(\lambda 00),1] \twoheadrightarrow_{\lambda\mathtt{ws}} \cdots$$

This follows from the failure of preservation of decent terms by $\mathtt{ws}$-reduction. We shall use our labelling strategy in order to formulate a characterization of the terms in $SN_{\lambda\mathtt{ws}}$.

Definition 4.59 Let $\mathcal{SN} \subseteq \mathcal{T}_{\lambda\mathtt{ws}}$ be the smallest set closed under the clauses:

$$\frac{P_1,\ldots,P_n \in \mathcal{SN} \quad n \geq 0}{\langle\vec{k}\rangle m P_1 \ldots P_n \in \mathcal{SN}}\ Ind \qquad\qquad \frac{P \in \mathcal{SN}}{\langle\vec{k}\rangle\lambda P \in \mathcal{SN}}\ Abs$$

$$\frac{\underline{\mathtt{ws}}(\underline{B[P[i/Q,j]]}) \in \mathcal{SN} \quad P,Q \in \mathcal{SN}}{B[P[i/Q,j]] \in \mathcal{SN}}\ Subs \qquad \frac{j = \sum\vec{k} \quad B[P[0/Q,j]] \in \mathcal{SN}}{B[(\langle\vec{k}\rangle\lambda(P))Q] \in \mathcal{SN}}\ Betas$$

In clauses *Subs* and *Betas* recall that B ranges over body contexts (Def. 4.44).

Proposition 4.60 $SN_{\lambda\mathtt{ws}} = \mathcal{SN}$.

Proof.

- $SN_{\lambda\mathtt{ws}} \subseteq \mathcal{SN}$.

  Let $M \in SN_{\lambda\mathtt{ws}}$. We prove by induction on $(maxred_{\lambda\mathtt{ws}}(M), \| M \|)$ where $\| M \|$ denotes the size of M (i.e. the number of variables, applications, abstractions, updatings and substitution operators), using the usual lexicographic ordering, that $M \in \mathcal{SN}$. According to Lemma 4.28 we have the following cases to consider:

  - $M = \langle\vec{k}\rangle n\vec{R}$ where $\vec{R} = R_1 \ldots R_n$. Then since $M \in SN_{\lambda\mathtt{ws}}$ each $R_i \in SN_{\lambda\mathtt{ws}}$ with $i \in 1..n$. By the induction hypothesis $R_i \in \mathcal{SN}$ with $i \in 1..n$. Then clause *Ind* concludes the case.
  - $M = \langle\vec{k}\rangle\lambda P$. Then $P \in SN_{\lambda\mathtt{ws}}$ and we use the induction hypothesis and clause *Abs*.
  - $M = ((\langle\vec{k}\rangle\lambda P)Q\vec{R}$. Then $P[0/Q,j]\vec{R} \in SN_{\lambda\mathtt{ws}}$ where $j = \sum\vec{k}$. By the induction hypothesis $P[0/Q,j]\vec{R} \in \mathcal{SN}$. Now set $B \overset{\text{def}}{=} \square$ and apply clause *Betas* to conclude the case.
  - $M = ((\langle\vec{k}\rangle P[i/Q,j])\vec{R}$. Then $\underline{\mathtt{ws}}(\underline{(\langle\vec{k}\rangle P[i/Q,j]}\vec{R}) \in SN_{\lambda\mathtt{ws}}$ by Lemma 4.54(1). By the induction hypothesis we know $\underline{\mathtt{ws}}(\underline{(\langle\vec{k}\rangle P[i/Q,j]}\vec{R}) \in \mathcal{SN}$. Also, we have $P, Q \in \mathcal{SN}$ by the induction hypothesis. Set $B \overset{\text{def}}{=} \langle\vec{k}\rangle\square\vec{R}$ and apply clause *Subs* to conclude the case.
  - $M = \langle\vec{k}^{+}\rangle(PQ)\vec{R}$. We have two cases to consider as dictated by Lemma 4.45:
    1. $M = B'[((\langle\vec{k'}\rangle\lambda(P_1))Q_1]$ for some body context B'. Then $B'[P_1[0/Q_1,\sum\vec{k'}]] \in SN_{\lambda\mathtt{ws}}$. The induction hypothesis yields $B'[P_1[0/Q_1,\sum\vec{k'}]] \in \mathcal{SN}$. Set $B \overset{\text{def}}{=} B'$ and apply *Betas* to conclude the case.
    2. $M = B'[\langle\vec{k'}\rangle P_1[i/Q_1,j]]$ for some body context B'. Then we proceed as in the previous case: $\underline{\mathtt{ws}}(\underline{B'[\langle\vec{k'}\rangle P_1[i/Q_1,j]]}) \in \mathcal{SN}$ and $P_1, Q_1 \in \mathcal{SN}$ by induction hypothesis. Set $B \overset{\text{def}}{=} B'[\langle\vec{k'}\rangle\square]$ and apply clause *Subs* to conclude the case.

- $\mathcal{SN} \subseteq SN_{\lambda\mathtt{ws}}$.

  By induction on the derivation of $M \in \mathcal{SN}$. The cases of clauses *Ind* and *Abs* are direct; those of *Subs* and *Betas* follow from Proposition 4.58:

- $(Subs)$. Suppose $M = B[P[i/Q,j]] \notin SN_{\lambda ws}$. First note that the conditions $P, Q \in SN_{\lambda ws}$ together with $\underline{\mathtt{ws}}(\overline{B[P[i/Q,j]]}) \in SN_{\lambda ws}$ guarantee that M is decent. We proceed by a case analysis on the form of B indicating for each form the defining case of $\mathtt{Zws}(\bullet)$ which allows us to arrive at a contradiction.
  * $B = (..((\Box M_1)M_2)...M_n)$ with $n \geq 0$. Then by the ninth defining case we arrive at a contradiction.
  * $B = (..(((\langle\vec{l^+}\rangle\Box M_1)M_2)...M_n)$ with $n \geq 0$. Ninth defining case.
  * $B = (..(((\langle\vec{l^+}\rangle(B''M')M_1)M_2)...M_n)$ with $n \geq 0$. Last defining case.
- $(Betas)$. Suppose $M = B[(\langle\vec{k}\rangle\lambda(P))Q] \notin SN_{\lambda ws}$. We proceed by a case analysis on the form of B indicating for each form the defining case of $\mathtt{Zws}(\bullet)$ (Def. 4.56) which allows us to arrive at a contradiction.
  * $B = (..((\Box M_1)M_2)...M_n)$ with $n \geq 0$. Then by the fifth defining case we arrive at a contradiction.
  * $B = (..(((\langle\vec{l^+}\rangle\Box M_1)M_2)...M_n)$ with $n \geq 0$. Last defining case.
  * $B = (..(((\langle\vec{l^+}\rangle(B''M')M_1)M_2)...M_n)$ with $n \geq 0$. Last defining case.

■

Note that $\mathcal{SN}$ is *deterministic* in the sense that if $M \in SN_{\lambda ws}$ then there is a unique derivation of M in $\mathcal{SN}$.

Part II

Explicit Substitutions for a Calculus of Objects

Chapter 5

Fields and Explicit Substitutions for Objects and Functions

This chapter studies a calculus of explicit substitutions for modeling object-oriented languages: the ς-calculus of M.Abadi and L.Cardelli [AC96] is augmented with explicit substitutions. The ς-calculus is a formalism which is at the level of abstraction of that of the λ-calculus, but which is based on objects instead of functions. It may be regarded as a minimul calculus of objects in the sense that it is difficult to conceive a simpler calculus for modelling object-oriented language constructs. Objects are the only computational structures in the calculus. An *object* is a collection of *methods*; each method has a bound variable that represents self and a body that produces a result. Only two operations are present and both apply to objects: *method invocation* and *method update*. The calculus is Turing complete in the sense that lambda calculus may be encoded via suitable objects [AC96]. The simple and elegant translation achieving this encoding, the *function-object translation*, shall be seen shortly. Now assuming some calculus of explicit substitutions, say e, is used to render metalevel substitution in ς at the object-level, it is natural to expect the resulting calculus of explicit substitutions ςe to be able to encode λe. For example, if explicit substitutions à la υ [BBLRD96] are considered in order to augment the ς-calculus, obtaining $\varsigma\upsilon$, then we would want to verify that $\lambda\upsilon$ may be encoded in $\varsigma\upsilon$, and hence rest assured that $\varsigma\upsilon$ shall be at least as expressive as the lambda calculus. In a variable name setting D.Kesner and P.E.Martínez López [KML98] have verified that indeed this is so, more precisely, they have verified that λx may be simulated in ςx. However, they have observed that in a de Bruijn indices setting, in order to verify that this simulation property indeed holds by adapting the aforementioned function-object translation, a new substitution notion must be introduced: *invoke substitution*. Invoke substitution behaves differently from the usual notion of substitution as regards the way in which de Bruijn indices are adjusted. For example, in $\lambda\upsilon$-style calculi of explicit substitutions we have $n+1[a/] \rightarrow_{RVar} n$ since the substitution $a/$ was supposedly generated by a *Beta*-redex and hence a lambda binder has now disappeared. However, for invoke substitution we have $n+1[@l] \rightarrow_{RInv} n+1$. Note that the index $n+1$ suffers no alteration. In an explicit substitution setting, the author has verified that *fields* are an appropriate tool in order to encode λe in ςe via the corresponding function-object translation. This chapter shall study a first-order calculus of explicit substitutions for the ς-calculus with fields as primitive constructs. Simulation of the $\lambda\upsilon$-calculus, confluence and preservation of strong normalization shall be the focus of our attention.

As mentioned in the introduction the study of calculi of explicit substitution has arised in the setting of λ-calculus, however there have been attempts to study explicit substitutions in a more general setting such as Explicit Combinatory Reduction Systems [BR96], based on the higher-order rewriting formalism *CRS* [Klo80] and eXplicit Reduction Systems (*XRS*) of Pagano [Pag98]. These formalisms although defined in a higher-order rewriting setting deal with a fixed 'built-in' calculus of explicit substitutions (Σ in Explicit *CRS* and $\sigma_{\Uparrow}$ in *XRS*). We shall see below that our calculus of explicit substitutions implementing the ς-object calculus is an instance of neither of these formalisms. At the time of writing this thesis the author has learned of an independent formalization of the ς-calculus published by M-O.Stehr [Ste00] based on an alternative representation for terms called Berkling's notation. This notation may be seen as the result of fusing both de Bruijn indices and variable names. The variable name part of the notation is most appropriate for simulating fields (as done in the ς-calculus), however the de Bruijn indices part of the notation requires index adjustment to be brought into the scene. The latter implies that invoke substitution or some analoguous notion shall also be required in order for the function-object translation to succeed, hence switching to Berkling's notation does not solve our problems.

Further work merging calculi of explicit substitutions and calculi of objects is that of F.Lang et al [LLL98]. This work aims at providing a unifying framework for studying operational semantics of various object-calculi, so may be considered orthogonal to our approach. Moreover, the framework is based on an extension of the $\lambda\mathcal{O}bj$-calculus [FHM94] rather than the ς-calculus. Since the $\lambda\mathcal{O}bj$-calculus builds on the λ-calculus there is no need to consider function-object translations there.

Structure of the chapter

We begin by briefly recalling the main constructs of the ς-calculus and immediately go on to consider the ς_{db}-calculus, ς in a de Bruijn indices setting. After addressing some basic properties of the ς-calculus with de Bruijn indices, we augment it with fields. A field may be seen as a method which does not use its self parameter. In the ς-calculus fields and proper methods have been unified as methods. By declaring methods that have no occurrences of its self parameter, fields may be simulated. This behaviour may also be achieved in the ς_{db}-calculus. Nevertheless, we introduce fields as primitive constructs (hence methods and fields coexist) since when working in an explicit substitutions setting it shall be seen that this simulation is no longer possible. Having introduced fields into the ς_{db}-calculus we proceed to prove the confluence of the resulting calculus.

Section 5.4 introduces the main calculus of this chapter: the ς_{dbes}-calculus. The latter results from ς by introducing fields and explicit substitutions in the style of λv [BBLRD96]. In an attempt to encode λv via the function-object translation two issues appear as obstacles:

1. encoding application: explicit substitutions interfere with the encoding of fields as methods which do not use their self parameter.

2. encoding abstraction: the use of metalevel substitution in the function-object translation requires a new notion of explicit substitution in order to be encoded soundly in the explicit substitution setting.

The first issue is taken care of by introducing fields as primitive constructs, the second by introducing the notion of invoke substitution. Simulation of λv is then seen to hold. Finally, we focus on confluence of ς_{dbes} and preservation of strong normalization. We use the interpretation technique in order to prove confluence and the recursive path ordering-based technique due to R.Bloo and H.Geuvers [BH98] to prove PSN. The latter property requires detailed attention since two notions of explicit substitution coexist in the ς_{dbes}-calculus and a weak form of interaction (in the form of a rewrite rule) is present.

The work reported in this chapter has been published as [Bon99b].

5.1 The ς-calculus

We have at our disposal an infinite list of variables denoted $x, y, z, \ldots$, and an infinite list of labels denoted $l, l_i, l', \ldots$. The labels shall be used to reference methods. An object is represented as a collection of methods denoted $l_i \doteq \varsigma(x_i).a_i$. We use l_i for representing method names and $\varsigma(x_i).a_i$ for method bodies. The labels of an object's methods are assumed to be all distinct. Operations allowed on objects are *method invocation* and *method update*. A method invocation of the method l_j in an object $[l_i \doteq \varsigma(x_i).a_i{}^{i\in 1..n}]$ is represented by the term $[l_i \doteq \varsigma(x_i).a_i{}^{i\in 1..n}].l_j$. The order in which the methods appear does not matter. As a result of method invocation, not only the corresponding method body is returned but also, this method body is supplied with a copy of its host object. Thus method bodies are represented as $\varsigma(x_i).a_i$ where ς is a binder that binds the variable x_i in a_i. This variable called *self* will be replaced by the host object when the associated method is invoked. It is this notion of *self* captured by the ς-calculus that allows an object to operate on itself. The other valid operation on objects is method update. A method $l_j \doteq \varsigma(x_j).a_j$ in an object o may be replaced by a new method $l'_j \doteq \varsigma(x'_j).a'_j$, thus resulting in a new object o'.

The terms of the ς-calculus, denoted $\mathcal{T}_\varsigma$, may be described more precisely by the following grammar:

$$a \ ::= \ x \mid a.l \mid a \lhd < l \doteq \varsigma(x).a > \mid [l_i \doteq \varsigma(x_i).a_i{}^{i\in 1..n}]$$

We say that x is a *variable*, $a.l$ is a *method invocation*, $a \lhd < l \doteq \varsigma(x).a >$ is a *method update* and $[l_i \doteq \varsigma(x_i).a_i{}^{i\in 1..n}]$ is an *object*. Free and bound variables are defined as expected. We shall write $FV(a)$ for the free variables of a. A variable convention similar to the one present in lambda calculus is adopted: terms differing only in the names of their bound variables (i.e. α-equivalent) are considered identical. For example, $[l_1 \doteq \varsigma(x).(x.l_1)]$ and $[l_1 \doteq \varsigma(y).(y.l_1)]$ are identified.

Example 5.1 (Natural numbers object) The natural numbers may be represented as objects. Here we show the number *zero*. All other natural numbers may be obtained from this one by using its *succ* method.

$$\begin{array}{lll} \mathit{zero} & \stackrel{\text{def}}{=} & [\mathit{iszero} \doteq \varsigma(x).\mathit{true}, \\ & & \mathit{pred} \doteq \varsigma(x).x, \\ & & \mathit{succ} \doteq \varsigma(x).(((x \lhd < \mathit{iszero} \doteq \varsigma(x).\mathit{false} >) \lhd < \mathit{pred} \doteq \varsigma(y).x >)] \end{array}$$

Note how the *succ* method has two nested method override operations. It indicates that when called it must modify the method *iszero* of self by replacing it with the method which returns the constant *false*. Moreover, it must modify the method *pred* so that it answers correctly with the previous number object.

Before defining the rewrite rules of the ς-calculus we need to take a quick look at substitution. The result of substituting a free variable x in a term a for a term b shall be denoted $a\{x \leftarrow b\}$. It is defined as follows.

Definition 5.2 (Substitution) Let a and b be terms in $\mathcal{T}_\varsigma$ and let x be a variable. Then *the substitution of x by b in a*, denoted $a\{x \leftarrow b\}$, is defined as:

$$\begin{array}{llll} [l_i \doteq \varsigma(x_i).m_i{}^{i\in 1..n}]\{x \leftarrow b\} & \stackrel{\text{def}}{=} & [l_i \doteq \varsigma(x_i).m_i\{x \leftarrow b\}\ {}^{i\in 1..n}] & \textit{if } x \neq x_i,\ i \in \mathit{1..n} \\ a.l\{x \leftarrow b\} & \stackrel{\text{def}}{=} & a\{x \leftarrow b\}.l & \\ a \lhd < l \doteq \varsigma(z).c > \{x \leftarrow b\} & \stackrel{\text{def}}{=} & a\{x \leftarrow b\} \lhd < l \doteq \varsigma(z).c\{x \leftarrow b\} > & \textit{if } x \neq z \\ x\{x \leftarrow b\} & \stackrel{\text{def}}{=} & b & \\ y\{x \leftarrow b\} & \stackrel{\text{def}}{=} & y & \textit{if } x \neq y \end{array}$$

Note that by the variable convention in the first clause we assume that $x_i \notin FV(b)$ for every i in $1..n$, and likewise $z \notin FV(b)$ in the third clause.

The semantics of the ς-calculus, referred to as *primitive semantics* in [AC96], is defined by the following rewrite rules:

$$\begin{array}{llll} [l_i \doteq \varsigma(x_i).a_i{}^{i\in 1..n}].l_j & \to_\varsigma & a_j\{x_j \leftarrow [l_i \doteq \varsigma(x_i).a_i{}^{i\in 1..n}]\} & j \in 1..n \\ [l_i \doteq \varsigma(x_i).a_i{}^{i\in 1..n}] \lhd < l_j \doteq \varsigma(x).a > & \to_\varsigma & [l_j \doteq \varsigma(x).a,\ l_i \doteq \varsigma(x_i).a_i{}^{i\in 1..n, i\neq j}] & j \in 1..n \end{array}$$

The first rule defines the semantics of *method invocation*. The result of invoking the method $l_j \doteq \varsigma(x_j).a_j$ (a 'call' to method $\varsigma(x_j).a_j$) is the body of the method a_j where the self variable has been replaced by a copy of the host object. The second rule defines the semantics of *method update*. Note that the substitution operator is not part of the ς-calculus but rather a metaoperation.

Example 5.3 Consider the natural number object *zero* from Example 5.1. If we invoke the method *succ* of the object *zero* then we may obtain an object representing the natural number one.

$$\begin{array}{lll} \mathit{zero.succ} & \twoheadrightarrow_\varsigma & [\mathit{iszero} \doteq \varsigma(x).\mathit{false}, \\ & & \mathit{pred} \doteq \varsigma(x).\mathit{zero} \\ & & \mathit{succ} \doteq \varsigma(x).(((x \lhd < \mathit{iszero} \doteq \varsigma(x).\mathit{false} >) \lhd < \mathit{pred} \doteq \varsigma(y).x >)] \end{array}$$

Note how the *iszero* method now correctly returns *false*, and how the *pred* method has also been modified appropriately. Compare the resulting object with the *zero* object.

Example 5.4 The following example is that of an object with the capability of making a backup copy of itself.

$$\begin{array}{lll} \mathit{bkupObject} & \stackrel{\text{def}}{=} & [\mathit{retrieve} \doteq \varsigma(x_1).x_1, \\ & & \mathit{backup} \doteq \varsigma(x_2).x_2 \lhd < \mathit{retrieve} \doteq \varsigma(x_1).x_2 >, \\ & & \text{(possibly additional methods)}] \end{array}$$

Then a call to the *backup* method of *bkupObject* returns an new object *bkupObject′* which when calling its *retrieve* method shall return the original backup object *bkupObject*. In other words, $o.\mathit{backup}.\mathit{retrieve} \twoheadrightarrow_\varsigma o$.

As regards the expressive power of this calculus, it is shown in [AC96] that lambda terms can be encoded as objects and that β-reduction can be simulated by ς-reduction.

Definition 5.5 (The function-object translation) The translation $\prec\!\prec \bullet \succ\!\succ$ from λ-terms to $\mathcal{T}_\varsigma$ is defined:

$$\begin{array}{lll}
\prec\!\prec x \succ\!\succ & \overset{\text{def}}{=} & x \\
\prec\!\prec \lambda x.a \succ\!\succ & \overset{\text{def}}{=} & [arg \doteq \varsigma(z).z.arg, val \doteq \varsigma(x).\ \prec\!\prec a \succ\!\succ \{x \leftarrow x.arg\}] \\
\prec\!\prec ab \succ\!\succ & \overset{\text{def}}{=} & \prec\!\prec a \succ\!\succ \circledast \prec\!\prec b \succ\!\succ \\
 & & \text{where } c \circledast d \overset{\text{def}}{=} (c \lhd < arg \doteq \varsigma(y).d >).val \text{ with } y \notin FV(d)
\end{array}$$

For instance the λ-term $(\lambda x.x)y$ is encoded as the ς-term $([arg \doteq \varsigma(z).z.arg, val \doteq \varsigma(x).x.arg] \lhd < arg \doteq \varsigma(v).y >).val$.

It is then proved for λ-terms a and b that if $a \rightarrow_\beta b$ then $\prec\!\prec a \succ\!\succ \twoheadrightarrow_\varsigma \prec\!\prec b \succ\!\succ$. In the preceding example we have:

$$\begin{array}{ll}
 & ([arg \doteq \varsigma(z).z.arg, val \doteq \varsigma(x).x.arg] \lhd < arg \doteq \varsigma(v).y >).val \\
\rightarrow_\varsigma & [arg \doteq \varsigma(v).y, val \doteq \varsigma(x).x.arg].val \\
\rightarrow_\varsigma & (x.arg)\{x \leftarrow [arg \doteq \varsigma(v).y, val \doteq \varsigma(x).x.arg]\} \\
= & [arg \doteq \varsigma(v).y, val \doteq \varsigma(x).x.arg].arg \\
\rightarrow_\varsigma & y\{v \leftarrow [arg \doteq \varsigma(v).y, val \doteq \varsigma(x).x.arg]\} \\
= & y
\end{array}$$

5.2 The ς-calculus with de Bruijn Indices

We now shift to a de Bruijn indices setting. Instead of labelling bound variables with names (as above) variables are labelled with natural numbers (see Section 2.2.2). For example, the term $[l_1 \doteq \varsigma(x_1).[l_2 \doteq \varsigma(y_1).x_1, l_3 \doteq \varsigma(z_1).z_1], l_4 \doteq \varsigma(x_2).y_2]$ shall be represented as $[l_1 \doteq \varsigma([l_2 \doteq \varsigma(2), l_3 \doteq \varsigma(1)]), l_4 \doteq \varsigma(2)]$. As in the λ_{db}-calculus the advantage attained is that there is no longer any need to perform renaming of bound variables. Nevertheless we must take care of index adjustments: if a substitution drags a term under a binder, its indices must be adjusted in order to avoid unwanted capture of indices.

The terms of the ς-calculus with de Bruijn indices (the ς_{db}-calculus), denoted $\mathcal{T}_{\varsigma_{db}}$, are characterized by the following grammar:

$$a \quad ::= \quad m \mid a.l \mid a \lhd < l \doteq \varsigma(a) > \mid [l_i \doteq \varsigma(a_i)\ ^{i \in 1..n}]$$

where m is a natural number greater than zero. Since the order of methods in an object is not important we shall hereafter identify objects which differ only in the order of appearance of their methods.

An example of a term in $\mathcal{T}_{\varsigma_{db}}$ is the de Bruijn representation of the abovementioned ς-term resulting from applying the function-object translation, asuming that the index of the variable y in our reference context is 1: $([arg \doteq \varsigma.1.arg, val \doteq \varsigma.1.arg] \lhd < arg \doteq \varsigma.2 >).val$.

The set of free indices of a term $a \in \mathcal{T}_{\varsigma_{db}}$, denoted $FI(a)$ is defined inductively as follows:

$$\begin{array}{lll}
FI(m) & \overset{\text{def}}{=} & \{m\} \\
FI(b.l) & \overset{\text{def}}{=} & FI(b) \\
FI(b \lhd < l \doteq \varsigma(c) >) & \overset{\text{def}}{=} & FI(b) \cup (FI(c) \backslash\!\!\backslash 1) \\
FI([l_i \doteq \varsigma(a_i)\ ^{i \in 1..n}]) & \overset{\text{def}}{=} & (FI(a_1) \backslash\!\!\backslash 1) \cup \ldots \cup (FI(a_n) \backslash\!\!\backslash 1)
\end{array}$$

where for $S \subset \mathbb{N}$ and $k \in \mathbb{N}$ we have $S \backslash\!\!\backslash k = \{n - k : n \in S, n > k\}$.

We now define substitution in the setting of indices.

Definition 5.6 (Ordinary Substitution) Let a and b be pure terms and $n \geq 1$. The substitution of a by b at level n, denoted $a\{\!\!\{n \leftarrow b\}\!\!\}$, is defined as follows:

$$\begin{array}{lll}
[l_i \doteq \varsigma(a_i)\ ^{i \in 1..m}]\{\!\!\{n \leftarrow b\}\!\!\} & \overset{\text{def}}{=} & [l_i \doteq \varsigma(a_i\{\!\!\{n+1 \leftarrow b\}\!\!\})\ ^{i \in 1..m}] \\
(a.l)\{\!\!\{n \leftarrow b\}\!\!\} & \overset{\text{def}}{=} & a\{\!\!\{n \leftarrow b\}\!\!\}.l \\
a \lhd < l \doteq \varsigma(c) > \{\!\!\{n \leftarrow b\}\!\!\} & \overset{\text{def}}{=} & a\{\!\!\{n \leftarrow b\}\!\!\} \lhd < l \doteq \varsigma(c\{\!\!\{n+1 \leftarrow b\}\!\!\}) > \\
m\{\!\!\{n \leftarrow b\}\!\!\} & \overset{\text{def}}{=} & \begin{cases} m-1 & if\ m > n \\ \mathcal{U}_0^n(b) & if\ m = n \\ m & if\ m < n \end{cases}
\end{array}$$

where for every $i \geq 0$ and $n \geq 1$, $\mathcal{U}_i^n(\bullet)$ is an *updating function* from terms in $\mathcal{T}_{\varsigma_{db}}$ to terms in $\mathcal{T}_{\varsigma_{db}}$ defined as follows:

$$\begin{array}{lcl}
\mathcal{U}_i^n([l_i \doteq \varsigma(a_i)\ ^{i\in 1..m}]) & \overset{\mathrm{def}}{=} & [l_i \doteq \varsigma(\mathcal{U}_{i+1}^n(a_i))\ ^{i\in 1..m}] \\
\mathcal{U}_i^n(a.l) & \overset{\mathrm{def}}{=} & \mathcal{U}_i^n(a).l \\
\mathcal{U}_i^n(a \lhd < l \doteq \varsigma(c) >) & \overset{\mathrm{def}}{=} & \mathcal{U}_i^n(a) \lhd < l \doteq \varsigma(\mathcal{U}_{i+1}^n(c)) > \\
\mathcal{U}_i^n(m) & \overset{\mathrm{def}}{=} & \begin{cases} m+n-1 & if\ m > i \\ m & if\ m \leq i \end{cases}
\end{array}$$

Remark 5.7 Note that $\mathcal{U}_i^1(a) = a$. Also, $\mathcal{U}_0^2(a)$ increments all indices representing free variables (in the usual sense) by one.

Having defined our terms and substitution we may now define the appropriate rewrite rules.

Definition 5.8 (Reduction in the ς_{db}-calculus) Reduction in the ς_{db}-calculus is defined by the following rewrite rules:

$$\begin{array}{lll}
[l_i \doteq \varsigma(a_i)\ ^{i\in 1..n}].l_j & \to_{\varsigma_{db}} & a_j\{\!\{1 \leftarrow [l_i \doteq \varsigma(a_i)\ ^{i\in 1..n}]\}\!\} \\
[l_i \doteq \varsigma(a_i)\ ^{i\in 1..n}] \lhd < l_j \doteq \varsigma(c) > & \to_{\varsigma_{db}} & [l_j \doteq \varsigma(c),\ l_i \doteq \varsigma(a_i)\ ^{i\in 1..n, i\neq j}]
\end{array}$$

Notice that substitution is still a metaoperation in this calculus, completely external to the reduction rules of the formalism.

We now address some basic properties concerning the behaviour of reduction in the ς_{db}-calculus. The Substitution Lemma, preservation of reduction via the updating functions, and preservation of reduction via de Bruijn substitution are the properties we shall look at. We first require the following technical lemma.

Lemma 5.9 1. Let $a, b \in \mathcal{T}_{\varsigma_{db}}$. Then $\forall i, j, k$ such that $i > 0, j \geq 0$ and $j < i \leq j + k$ we have $\mathcal{U}_j^{k+1}(a)\{\!\{i \leftarrow b\}\!\} = \mathcal{U}_j^k(a)$.

2. Let $a, b \in \mathcal{T}_{\varsigma_{db}}$. Then $\forall i, n, k$ such that $i \leq n - k$ we have $\mathcal{U}_k^i(a)\{\!\{n \leftarrow b\}\!\} = \mathcal{U}_k^i(a\{\!\{n - i + 1 \leftarrow b\}\!\})$.

3. Let $a, b \in \mathcal{T}_{\varsigma_{db}}$. Then $\forall n, k$ such that $n \leq k + 1$ we have $\mathcal{U}_k^j(a\{\!\{n \leftarrow b\}\!\}) = \mathcal{U}_{k+1}^j(a)\{\!\{n \leftarrow \mathcal{U}_{k-n+1}^j(b)\}\!\}$.

Proof. By induction on a. •

As expected, the classical Substitution Lemma [Bar84, Lemma 2.1.16] also holds in the de Bruijn setting. The usual conditions demanded on free variables, in order for this result to hold in the variable name setting, are reflected as conditions on the indices in the de Bruijn setting.

Lemma 5.10 (Substitution Lemma) Let $a, b, c \in \mathcal{T}_{\varsigma_{db}}$. Then $\forall n, i \geq 1$ such that $i \leq n$ we have $a\{\!\{i \leftarrow b\}\!\}\{\!\{n \leftarrow c\}\!\} = a\{\!\{n + 1 \leftarrow c\}\!\}\{\!\{i \leftarrow b\{\!\{n - i + 1 \leftarrow c\}\!\}\}\!\}$

Proof. The proof is by induction on a, using Lemma 5.9 (items 1 and 2). •

The following results state that every rewrite step in the ς_{db}-calculus is preserved by the updating functions. Since the definition of de Bruijn substitution relies on that of the updating functions Lemma 5.11 shall be required in order to prove that also de Bruijn substitution preserves ς_{db}-rewrite steps. The latter result is presented as Lemma 5.12. Both are proved by induction on a; the first uses Lemma 5.9(3), and the second the Substitution Lemma and Lemma 5.11.

Lemma 5.11 Let $a, a' \in \mathcal{T}_{\varsigma_{db}}$. If $a \to_{\varsigma_{db}} a'$ then $\mathcal{U}_k^j(a) \to_{\varsigma_{db}} \mathcal{U}_k^j(a')$.

Lemma 5.12 Let $a, a' \in \mathcal{T}_{\varsigma_{db}}$ and $n \geq 1$. If $a \to_{\varsigma_{db}} a'$ then

1. $a\{\!\{n \leftarrow b\}\!\} \to_{\varsigma_{db}} a'\{\!\{n \leftarrow b\}\!\}$

2. $b\{\!\{n \leftarrow a\}\!\} \twoheadrightarrow_{\varsigma_{db}} b\{\!\{n \leftarrow a'\}\!\}$

Confluence of the ς_{db}-calculus may be proved by providing translation functions from terms with variable names to terms with de Bruijn indices and the corresponding results that these definitions are well-behaved with respect to the updating functions, substitution and reduction. The result then follows from confluence of ς. The details are rather straightforward and hence ommitted.

Lemma 5.13 (Confluence of the ς_{db}-calculus) The ς_{db}-calculus is confluent.

5.3 The ς-calculus with de Bruijn Indices and Fields

This section introduces de ς_{db}-calculus with fields, called ς^f_{db}-calculus, and proves its confluence. The ς^f_{db}-calculus is a straightforward extension of ς_{db} and is formulated in preparation for the introduction of explicit substitutions in Section 5.4.

From a general standpoint an object may be regarded as an entity encapsulating state (fields) and behaviour (methods) in an object-oriented language. These methods allow the object to modify its local state as well as interact with other objects. Let us concentrate on fields. Consider an object `calculator` that possesses a field which allows the user (another object) to store some intermediate result. For this the object interface includes a method `save(n)` where `n` is the number to be stored that returns a new calculator object where `n` has been saved as an intermediate result. Also, in order to retrieve this value it includes a method `recall`. Thus one would expect the equation `calculator.save(n).recall=n` to be true. This is characteristic of the behaviour of fields. As mentioned in [AC96] the ς-calculus does not include field contructs as primitive. Nevertheless, methods that do not use the self variable may be regarded as fields. Indeed, let b be a term in the ς-calculus such that it has no occurrence of a variable x. Then we have

$$[l \doteq \varsigma(x).b].l \to_{\varsigma} b\{x \leftarrow [l \doteq \varsigma(x).b]\} = b$$

Thus we obtain exactly b, the body of the method $l \doteq \varsigma(x).b$.

Now consider the setting where variables are represented no longer by variable names but by de Bruijn indices. Then we could attempt to proceed as above. Consider a term b in the ς_{db}-calculus such that $1 \notin FV(b)$. Then we have,

$$[l \doteq \varsigma(b)].l \to_{\varsigma_{db}} b\{\!\{1 \leftarrow [l \doteq \varsigma(b)]\}\!\} = b^-$$

where b^- represents b with free indices decremented in one unit. The result obtained is not the same as the body of the method $l \doteq \varsigma(b)$.

Thus we may simulate fields in ς_{db}-calculus by representing them as methods $l \doteq \varsigma(b^+)$ where b^+ represents b where all free indices are incremented in one unit ($b^+ \stackrel{\text{def}}{=} \mathcal{U}^2_0(b)$). Nevertheless, we shall introduce fields as primitive constructs in the language. The reason for doing so is that when explicit substitutions are introduced into the calculus and the translation of (an explicit substitution version of) the λ-calculus into this extension studied, field simulation is no longer for free (Section 5.4).

Therefore in our de Bruijn setting we incorporate, as a primitive notion, that of a field. The terms of the ς-calculus à la de Bruijn with fields (hereafter the ς^f_{db}-calculus), denoted $\mathcal{T}_{\varsigma^f_{db}}$, are called *pure terms* and are characterized by the following grammar:

$$\begin{array}{lcl} a & ::= & n \mid a.l \mid a \lhd < m > \mid [m_i^{\,i\in 1..n}] \\ m & ::= & l \doteq g \mid l := a \\ g & ::= & \varsigma(a) \end{array}$$

where n is a natural number greater than zero. Note that we have chosen the above presentation for the sort of methods over the more natural $m \ ::= \ l \doteq \varsigma(a) \mid l := a$. This is done in preparation for the following section where we shall extend the sort of method bodies (g above) with explicit substitutions.

An object is constructed by a list of methods and fields. A method is denoted '$l \doteq g$' where l is its label and g its body. A field is denoted '$l := a$' where l is its label and a its body. Note that we may override a method with a field and viceversa; so there is only one sort of labels (i.e. labels for fields and labels for methods are not distinguished).

The set of free indices of a term $a \in \mathcal{T}_{\varsigma^f_{db}}$, denoted $FI(a)$ is defined similarly as done in Section 5.2. Note that for fields we add $FI(l := a) \stackrel{\text{def}}{=} FI(a)$. The same applies for ordinary substitution and the updating functions.

For fields and method bodies we add $(l := a)\{\!\{n \leftarrow b\}\!\} \overset{\text{def}}{=} l := a\{\!\{n \leftarrow b\}\!\}$ and $\varsigma(a)\{\!\{n \leftarrow b\}\!\} \overset{\text{def}}{=} \varsigma(a\{\!\{n{+}1 \leftarrow b\}\!\})$, and $\mathcal{U}_i^n(l := b) \overset{\text{def}}{=} l := \mathcal{U}_i^n(b)$ and $\mathcal{U}_i^n(\varsigma(a)) \overset{\text{def}}{=} \varsigma(\mathcal{U}_{i+1}^n(a))$.

We now define the appropriate reduction rules using the notion of substitution defined above.

Definition 5.14 (Reduction in the ς_{db}^f-calculus) Reduction in the ς_{db}^f-calculus is defined by the following rewrite rules:

$$\begin{array}{lll}
[l_j \doteq \varsigma(a), m_i{}^{i\in 1..n, i\neq j}].l_j & \rightarrow_{\varsigma_{db}^f} & a\{\!\{1 \leftarrow [l_j \doteq \varsigma(a), m_i{}^{i\in 1..n, i\neq j}]\}\!\} \\
[l_j := a, m_i{}^{i\in 1..n, i\neq j}].l_j & \rightarrow_{\varsigma_{db}^f} & a \\
[m_i{}^{i\in 1..n}] \lhd < l_j \doteq \varsigma(a) > & \rightarrow_{\varsigma_{db}^f} & [l_j \doteq \varsigma(a),\ m_i{}^{i\in 1..n, i\neq j}] \\
[m_i{}^{i\in 1..n}] \lhd < l_j := a > & \rightarrow_{\varsigma_{db}^f} & [l_j := a,\ m_i{}^{i\in 1..n, i\neq j}]
\end{array}$$

The second rule indicates that the bodies of fields should be projected without undergoing any index adjustment. Notice that substitution is still a metaoperation in this calculus, completely external to the rewrite rules of the formalism.

As for the ς_{db}-calculus, basic properties concerning substitution (such as the Substitution Lemma), and preservation of reduction via substitution and the updating functions, also hold for the ς_{db}^f-calculus. These results are used in the appendix to prove that ς_{db}^f is confluent.

Lemma 5.15 (Confluence of the ς_{db}^f-calculus) The ς_{db}^f-calculus is confluent.

We finish this section with a word on the relation between the ς_{db}^f-calculus and the ς_{db}-calculus. Firstly, note that since $\mathcal{T}_{\varsigma_{db}} \subseteq \mathcal{T}_{\varsigma_{db}^f}$ and by the definition of reduction in the ς_{db}-calculus and in the ς_{db}^f-calculus we have that for any $a, b \in \mathcal{T}_{\varsigma_{db}}$ if $a \rightarrow_{\varsigma_{db}} b$ then $a \rightarrow_{\varsigma_{db}^f} b$. In order to show that the ς_{db}^f-calculus can be simulated in the ς_{db}-calculus we define the following translation function.

Definition 5.16 The translation $h(\bullet)$: $\mathcal{T}_{\varsigma_{db}^f} \rightarrow \mathcal{T}_{\varsigma_{db}}$ is defined as,

$$\begin{array}{lll@{\qquad}lll}
h(n) & \overset{\text{def}}{=} & n & h(l \doteq g) & \overset{\text{def}}{=} & l \doteq h(g) \\
h(b.l) & \overset{\text{def}}{=} & h(b).l & h(l := a) & \overset{\text{def}}{=} & l \doteq \varsigma(\mathcal{U}_0^2(h(a))) \\
h(b \lhd < m >) & \overset{\text{def}}{=} & h(b) \lhd < h(m) > & h([m_i{}^{i\in 1..n}]) & \overset{\text{def}}{=} & [h(m_i)\ {}^{i\in 1..n}] \\
h(\varsigma(b)) & \overset{\text{def}}{=} & \varsigma(h(b)) & & &
\end{array}$$

Note that if t is a term in $\mathcal{T}_{\varsigma_{db}}$ then $h(t) = t$.

We may now verify that if a and b are terms in $\mathcal{T}_{\varsigma_{db}^f}$ and $a \rightarrow_{\varsigma_{db}^f} b$ then $h(a) \rightarrow_{\varsigma_{db}} h(b)$. This requires first proving the following two items.

1. Let $a \in \mathcal{T}_{\varsigma_{db}^f}$, $i \geq 0$ and $k > 0$. Then $h(\mathcal{U}_i^k(a)) = \mathcal{U}_i^k(h(a))$.

2. Let $a, b \in \mathcal{T}_{\varsigma_{db}^f}$ and $k > 0$. Then $h(a\{\!\{k \leftarrow b\}\!\}) = h(a)\{\!\{k \leftarrow h(b)\}\!\}$.

The proofs are straightforward but tedious and hence omitted.

5.4 Introducing Explicit Substitutions

The ς-calculus with explicit substitutions and de Bruijn indices, which we shall hereafter refer to as the ς_{dbes}-calculus, is presented in this section. This calculus introduces two forms of substitution into the object-language: ordinary substitution and invoke substitution. Also, since the ς_{dbes}-calculus builds on ς_{db}^f we have (explicit) fields in the object-language at our disposal. We sum up the object calculi we have already seen in Figure 5.1. In this section we shall explain why we have incorporated fields into the object-language.

Let us begin by describing the set of terms of our new calculus. The set of terms of the ς_{dbes}-calculus, denoted $\mathcal{T}_{\varsigma_{dbes}}$, consists of terms of sort `Term` and terms of sort `Subst`. These are defined by the following grammar (sort `Term` to the left and sort `Subst` to the right):

Calculus	Notation of variables	Fields	Substitution
ς	Names	Simulated	Implicit
ς_{db}	Indices	Simulated	Implicit
ς^{f}_{db}	Indices	Primitive	Implicit
ς_{dbes}	Indices	Primitive	Explicit

Figure 5.1: Variants of ς

$$\begin{array}{lcl} a & ::= & n \mid a.l \mid a \lhd < m > \mid [{m_i}^{i \in\ 1..n}] \mid a[s] \\ m & ::= & l \doteq g \mid l := a \\ g & ::= & \varsigma(a) \mid g[s] \end{array} \qquad s \ ::= \ a/ \mid @l \mid \Uparrow(s) \mid \uparrow$$

where n is a natural number greater than zero.

Unless otherwise stated when we say that "a is a term in $\mathcal{T}_{\varsigma_{dbes}}$" we mean "$a$ is a term in $\mathcal{T}_{\varsigma_{dbes}}$ of sort Term". A *closure* is a term of the form $a[s]$. A term that does not contain occurrences of closures as subterms is called a *pure term.* A term $a[s]$ may be regarded as the term a with pending substitution s. The substitution operator $\bullet[\bullet]$ is part of the calculus (i.e. it is at the object-level). A substitution s with an occurrence of $a/$ is called an *ordinary substitution* whereas a substitution s with an occurrence of $@l$ is called an *invoke substitution.* Properties of invoke substitutions shall be studied later. Note that if we erase the grammar rules generating closures then we obtain the set $\mathcal{T}_{\varsigma^{f}_{db}}$.

The substitution grammar (and substitution calculus) for ordinary substitution is based on the calculus of explicit substitution for the lambda calculus, λv [Les94]. Although there are many calculi of explicit substitutions in the literature we are inclined to using λv due to its simplicity.

We shall frequently use the notation $\Uparrow^i(s)$ and $a[s]^i$ defined inductively as

$$\begin{array}{lcl} \Uparrow^0(s) & \stackrel{\text{def}}{=} & s \\ \Uparrow^{i+1}(s) & \stackrel{\text{def}}{=} & \Uparrow(\Uparrow^i(s)) \end{array} \qquad \begin{array}{lcl} a[s]^0 & \stackrel{\text{def}}{=} & a \\ a[s]^{i+1} & \stackrel{\text{def}}{=} & a[s]^i[s] \end{array}$$

The semantics of the ς_{dbes}-calculus is defined by the set of rewrite rules given in Figure 5.2.

The rule MI activates a method invocation. The rule FI activates a field invocation. The rules MO, FO activate method override and field override respectively. Rules SM, SO, SF, SB, SI, SU allow the propagation of the substitution operator through method body, object, field, method, invocation and override constructors. Rules $FVar$, $RVar$, $Finv$, $RInv$, $FVarLift$, $RVarLift$, $VarShift$ allow the computation of substitutions on indices. Finally, the rule CO expresses a form of interaction of substitutions, and SW expresses a (weak) form of commutation or switching of substitutions. These two rules will be used in simulating λv in the ς_{dbes}-calculus.

It is interesting to compare rules $RVar$ and $RInv$. The creation of a substitution of the form $b/$ is accompanied by the elimination of a binder (see rule MI). Hence all 'free' indices should be decremented in one unit. Whereas in the case of the invoke substitution operator '$@\bullet$' no such adjustment is made. This is because the invoke substitution is only applied to bound indices, as we shall see below. This may be illustrated by the following observation, which may be verified by induction:

Proposition 5.17 For every term a in $\mathcal{T}_{\varsigma_{dbes}}$and $n \geq 1$ and every $i \geq 0$,

$$n[\Uparrow^i(@l)] \twoheadrightarrow_{R_1} \begin{cases} n.l & n = i+1 \\ n & n \neq i+1 \end{cases} \qquad n[\Uparrow^i(a/)] \twoheadrightarrow_{R_2} \begin{cases} n-1 & n > i+1 \\ a[\uparrow]^i & n = i+1 \\ n & n < i+1 \end{cases}$$

where $R_1 = \{FInv, RInv, FVarLift, RVarLift, VarShift\}$ and $R_2 = \{FVar, RVar, FVarLift, RVarLift, VarShift\}$

The exact relationship between the explicit substitution operators and their metalevel counterparts shall be made precise in Section 5.5.

The ς_{dbes}-calculus without the rules MI, MO, FI and FO is referred to as the $ESDB$ rewrite system. Note that $ESDB$ is not locally confluent since for example the term $1[@l_1][[l_1 := b]/]$ reduces to two different terms by the rules $FInv$ and CO respectively, and requires FI to close the diagram. The rewrite system obtained by eliminating rules CO and SW is called the BES (Basic Explicit Substitution)-rewrite system.

$$
\begin{array}{llll}
[l_j \doteq \varsigma(a), {m_i}^{i\in 1..n, i\neq j}].l_j & \rightarrow_{MI} & a[[l_j \doteq \varsigma(a), {m_i}^{i\in 1..n, i\neq j}]/] & \\
[l_j := a, {m_i}^{i\in 1..n, i\neq j}].l_j & \rightarrow_{FI} & a & \\
[{m_i}^{i\,\in\,1..n}] \lhd < l_j \doteq g > & \rightarrow_{MO} & [l_j \doteq g,\ m_i^{i\in 1..n, i\neq j}] & j \in 1..n \\
[{m_i}^{i\,\in\,1..n}] \lhd < l_j := a > & \rightarrow_{FO} & [l_j := a,\ m_i^{i\in 1..n, i\neq j}] & j \in 1..n \\
 & & & \\
(\varsigma(c))[s] & \rightarrow_{SM} & \varsigma(c[\Uparrow(s)]) & \\
[{m_i}^{i\in\ 1..n}][s] & \rightarrow_{SO} & [m_i[s]\ ^{i\in\ 1..n}] & \\
(l := a)[s] & \rightarrow_{SF} & l := a[s] & \\
(l \doteq g)[s] & \rightarrow_{SB} & l \doteq g[s] & \\
a.l[s] & \rightarrow_{SI} & a[s].l & \\
a \lhd < m > [s] & \rightarrow_{SU} & a[s] \lhd < m[s] > & \\
1[a/] & \rightarrow_{FVar} & a & \\
p+1[a/] & \rightarrow_{RVar} & p & \\
1[@l] & \rightarrow_{FInv} & 1.l & \\
p+1[@l] & \rightarrow_{RInv} & p+1 & \\
1[\Uparrow(s)] & \rightarrow_{FVarLift} & 1 & \\
p+1[\Uparrow(s)] & \rightarrow_{RVarLift} & p[s][\uparrow] & \\
p[\uparrow] & \rightarrow_{VarShift} & p+1 & \\
a[\Uparrow^i(@l_j)][\Uparrow^i([l_j := b, {m_i}^{i\in 1..n, i\neq j}]/)] & \rightarrow_{CO} & a[\Uparrow^i(b/)] & \\
a[\Uparrow^i(@l)][\Uparrow^k(s)] & \rightarrow_{SW} & a[\Uparrow^k(s)][\Uparrow^i(@l)] & k > i
\end{array}
$$

Figure 5.2: The ς_{dbes}-calculus

5.4.1 The Need for Explicit Fields

In Section 5.3 we saw that although the ς^f_{db}-calculus incorporated fields as primitive constructs this is not strictly necessary as fields may be simulated in the ς_{db}-calculus in a rather natural way (Definition 5.16). This situation no longer holds when explicit substitutions are introduced and when we attempt to encode the λv-calculus in the ς'_{dbes}-calculus using the function-object translation (Definition 5.5). Let us delve deeper into this issue.

Let us ignore fields as a primitive construct in the language for the moment and return to our simulation of fields as discussed in Section 5.3. A field b is represented as the method $l \doteq \varsigma(b^+)$. The ς_{dbes}-calculus is then reduced to, say, ς'_{dbes}, where rules FI, FO, SF and CO have been eliminated.

Now when we attempt to translate the λv-calculus into the ς'_{dbes}-calculus in the style of $\prec\!\prec \bullet \succ\!\succ$ (Definition 5.5) we arrive naturally to the following translation function k:

$$
\begin{array}{lcl@{\qquad}lcl}
k(a/) & \stackrel{\text{def}}{=} & k(a)/ & k(n) & \stackrel{\text{def}}{=} & n \\
k(\Uparrow(s)) & \stackrel{\text{def}}{=} & \Uparrow(k(s)) & k(\lambda a) & \stackrel{\text{def}}{=} & [arg \doteq \varsigma(1.arg), val \doteq \varsigma(k(a)[@arg])] \\
k(a[s]) & \stackrel{\text{def}}{=} & k(a)[k(s)] & k(\uparrow) & \stackrel{\text{def}}{=} & \uparrow \\
k(ab) & \stackrel{\text{def}}{=} & (k(a) \lhd < arg \doteq \varsigma(k(b)^+) >).val & & &
\end{array}
$$

But the meaning of $k(b)^+$ is no longer clear since $k(b)$ may have occurrences of the explicit substitution operator (it is no longer a pure term). To remedy this situation the next logical step would be to introduce an 'explicit substitution version' of the $\bullet^+$ operator which in fact we already have: the $\uparrow$ operator. The final clause of the definition of k is now replaced by $k(ab) \stackrel{\text{def}}{=} (k(a) \lhd < arg \doteq \varsigma(k(b)[\uparrow]) >).val$

So now we proceed to verify that the translation is correct (preserves λv-reduction). Consider for example the λv-reduction rule $(\lambda a)b \rightarrow_{Beta} a[b/]$. Then we must have $k((\lambda a)b) \twoheadrightarrow_{\varsigma'_{dbes}} k(a[b/])$. We can go as far as:

$$
\begin{array}{ll}
k((\lambda a)b) & \stackrel{\text{def}}{=} \\
([arg \doteq \varsigma(1.arg), val \doteq \varsigma(k(a)[@arg])] \lhd < arg \doteq \varsigma(k(b)[\uparrow]) >).val & \rightarrow_{MO} \\
[arg \doteq \varsigma(k(b)[\uparrow]), val \doteq \varsigma(k(a)[@arg])].val & \rightarrow_{MI} \\
k(a)[@arg][[arg \doteq \varsigma(k(b)[\uparrow]), val \doteq \varsigma(k(a)[@arg])]/] &
\end{array}
$$

Thus in order to arrive at $k(a)[k(b)/]$ we are in need of adding to the ς'_{dbes}-calculus a commutation rule of the form: $a[\Uparrow^i (@l_j)][\Uparrow^i ([l_j \doteq \varsigma(b[\uparrow]), {m_i}^{i\in 1..n, i\neq j}]/)] \rightarrow_{Com} a[\Uparrow^i (b/)]$ (taking $i = 0$ suffices for our example). But adding a rule like *Com* clearly introduces confluence problems.

A variant could be *Com′* defined as: $a[\Uparrow^i (@l_j)][\Uparrow^i ([l_j \doteq \varsigma(b), {m_i}^{i\in 1..n, i\neq j}]/)] \rightarrow_{Com'} a[\Uparrow^i (c/)]$ where $b =_{BES} c[\uparrow]$. The major drawbacks are then the fact that the rule is conditional and (computationally) expensive checking on the equational substitution theory is required (this resembles problems studied when dealing with η-contraction in explicit substitution calculi [Bri95, Río93, Kes00]).

These problems stem from the fact that the formulation of rules which are subject to restrictions on the free variables in a de Bruijn indices setting and in the presence of explicit substitutions is non trivial. Here, we have solved these issues by a minor change in the syntax so as to represent fields as primitive operators. In fact, the rewrite rule *CO* of the named ς_{ES} presented in [KML98] is conditional, whereas the *CO*-rule presented in this work, in a de Bruijn index setting, is actually simpler since no condition is present.

5.4.2 Encoding λv-terms in the ς_{dbes}-calculus

Let us now consider how to encode the lambda calculus with explicit substitutions λv in the ς_{dbes}-calculus. We start by augmenting the grammar productions for the terms of the ς_{dbes}-calculus in order to allow abstractions and applications as legal terms. We then define a translation from terms in the λv-calculus into this augmented set of terms which preserves reduction. We recall the main definitions of the λv-calculus, see Section 2.3 for further details. Terms are defined by the following grammars $t ::= n \mid tt \mid \lambda t \mid t[s]$ with n a natural number greater than zero, and $s ::= \uparrow \mid t/ \mid \Uparrow (s)$. We recall the rules below.

$$\begin{array}{lll@{\qquad\qquad}lll}
(\lambda a)b & \rightarrow_{Beta} & a[b/] & n+1[a/] & \rightarrow_{RVar} & n \\
(a\ b)[s] & \rightarrow_{App} & a[s]b[s] & 1[\Uparrow (s)] & \rightarrow_{FVarLift} & 1 \\
\lambda a[s] & \rightarrow_{Lam} & \lambda(a[\Uparrow (s)]) & n+1[\Uparrow (s)] & \rightarrow_{RVarLift} & n[s][\uparrow] \\
1[a/] & \rightarrow_{FVar} & a & n[\uparrow] & \rightarrow_{VarShift} & n+1
\end{array}$$

The mixed set of terms, which we shall call $\mathcal{T}_{\lambda\varsigma_{dbes}}$, consists of terms of sort Term and terms of sort Subst (which remain unaltered). The terms of sort Term are defined by the following grammar:

$$\begin{array}{lcl}
a & ::= & n \mid a.l \mid a \lhd < m > \mid [{m_i}^{i\in\ 1..n}] \mid a[s] \mid \lambda a \mid (a\ a) \\
m & ::= & l \doteq g \mid l := a \\
g & ::= & \varsigma(a) \mid g[s]
\end{array}$$

where n is any natural number greater than zero.

The rewrite rules of the $\lambda\varsigma_{dbes}$-calculus consists of the rewrite rules of the ς_{dbes}-calculus together with the rules *Beta*, *Lam* and *App* of the λv-calculus (note that the remaining rules of λv already belong to the ς_{dbes}-calculus). The resulting system may be proved confluent using the interpretation technique [Har87] and the fact that the corresponding system with metalevel substitutions is an orthogonal rewrite system.

The encoding of λv-terms into $\lambda\varsigma_{dbes}$-terms makes use of the invoke explicit substitution operator '@•' and fields.

Definition 5.18 (Translation of $\lambda\varsigma_{dbes}$-terms into ς_{dbes}-terms) The translation $\prec \bullet \succ$ from $\lambda\varsigma_{dbes}$-terms into terms in $\mathcal{T}_{\varsigma_{dbes}}$ is defined as follows:

$$\begin{array}{lcl@{\qquad\qquad}lcl}
\prec n \succ & \overset{\text{def}}{=} & n & \prec \varsigma(a) \succ & \overset{\text{def}}{=} & \varsigma(\prec a \succ) \\
\prec a.l \succ & \overset{\text{def}}{=} & \prec a \succ .l & \prec a/ \succ & \overset{\text{def}}{=} & \prec a \succ / \\
\prec a \lhd < m > \succ & \overset{\text{def}}{=} & \prec a \succ \ \lhd < \prec m \succ > & \prec \Uparrow (s) \succ & \overset{\text{def}}{=} & \Uparrow (\prec s \succ) \\
\prec [{m_i}^{i\ \in\ 1..n}] \succ & \overset{\text{def}}{=} & [\prec m_i \succ^{\ i\ \in\ 1..n}] & \prec \uparrow \succ & \overset{\text{def}}{=} & \uparrow \\
\prec l \doteq g \succ & \overset{\text{def}}{=} & l \doteq \prec g \succ & \prec @l \succ & \overset{\text{def}}{=} & @l \\
\prec l := a \succ & \overset{\text{def}}{=} & l := \prec a \succ & \prec \lambda a \succ & \overset{\text{def}}{=} & [arg \doteq \varsigma(1.arg), val \doteq \varsigma(\prec a \succ [@arg])] \\
\prec a[s] \succ & \overset{\text{def}}{=} & \prec a \succ [\prec s \succ] & \prec ab \succ & \overset{\text{def}}{=} & \prec a \succ \circledast \prec b \succ \\
 & & & & & \text{where } p \circledast q \overset{\text{def}}{=} (p \lhd < arg := q >).val
\end{array}$$

The translation interprets the lambda expressions abstraction and application into objects leaving the rest of the constructions without modifications. The translation of an abstraction introduces the invoke substitution. Note that the index level 1 (to which the invoke substitution applies) is bound. This reveals a difference as regards the behaviour of ordinary and invoke substitutions, as discussed above. Ordinary substitution is of no use since its index adjusting mechanism does not exhibit the desired behaviour.

We illustrate the translation with an example. Consider the K combinator defined as $K = \lambda\lambda 2$

$$\prec K \succ = [arg \doteq \varsigma(1.arg), val \doteq \varsigma([arg \doteq \varsigma(1.arg), val \doteq \varsigma(2[@arg])])[@arg]]$$

The principal motivation behind the introduction of the rules describing the interaction of ordinary substitution and invoke substitution lies in the following proposition.

Proposition 5.19 (ς_{dbes} **simulates** $\lambda\upsilon$) Let a, b be $\lambda\varsigma_{dbes}$-terms. If $a \to_{\lambda\upsilon} b$ then $\prec a \succ \twoheadrightarrow_{\varsigma_{dbes}} \prec b \succ$.

Proof. The proof is by structural induction on the $\lambda\upsilon$-term. We just consider the cases $\prec (\lambda a)b \succ \twoheadrightarrow_{\varsigma_{dbes}} \prec a[b/] \succ$ (Case 1) and $\prec \lambda a[s] \succ \twoheadrightarrow_{\varsigma_{dbes}} \prec \lambda(a[\Uparrow (s)]) \succ$ (Case 2) as examples.

Case 1.

$$\begin{array}{ll}
\prec (\lambda a)b \succ & \overset{\text{def}}{=} \\
([arg \doteq \varsigma(1.arg), val \doteq \varsigma(\prec a \succ [@arg])] \lhd < arg := \prec b \succ >).val & \to_{MO} \\
[arg := \prec b \succ, val \doteq \varsigma(\prec a \succ [@arg])].val & \to_{MI} \\
\prec a \succ [@arg][[arg := \prec b \succ, val \doteq \varsigma(\prec a \succ [@arg])]/] & \to_{CO} \\
\prec a \succ [\prec b \succ /] & \overset{\text{def}}{=} \\
\prec a[b/] \succ &
\end{array}$$

Case 2.

$$\begin{array}{ll}
\prec (\lambda a)[s] \succ & \overset{\text{def}}{=} \\
[arg \doteq \varsigma(1.arg), val \doteq \varsigma(\prec a \succ [@arg])][\prec s \succ] & \to_{SO} \\
[arg \doteq (\varsigma(1.arg))[\prec s \succ], val \doteq (\varsigma(\prec a \succ [@arg]))[\prec s \succ]] & \twoheadrightarrow_{BES} \\
[arg \doteq \varsigma(1.arg[\Uparrow (\prec s \succ)]), val \doteq \varsigma(\prec a \succ [@arg][\Uparrow (\prec s \succ)])] & \twoheadrightarrow_{BES} \\
[arg \doteq \varsigma(1.arg), val \doteq \varsigma(\prec a \succ [@arg][\Uparrow (\prec s \succ)])] & \to_{SW} \\
[arg \doteq \varsigma(1.arg), val \doteq \varsigma(\prec a \succ [\Uparrow (\prec s \succ)][@arg])] & \overset{\text{def}}{=} \\
\prec \lambda(a[\Uparrow (s)]) \succ &
\end{array}$$

The cases where the reduction is internal are similar and may be dealt with by applying the induction hypothesis.

■

We may therefore conclude that $\lambda\upsilon$-derivations may be translated into ς_{dbes}-derivations, thereby implementing objects and functions at the same time.

5.5 Confluence and PSN of the ς_{dbes}-calculus

In this section we shall prove some essential properties required for any calculus of explicit substitutions implementing a calculus where substitution operates at the metalevel. Firstly, we study some properties of the substitution calculus such as strong normalization. Then the relation between the ς^{f}_{db}-calculus and the ς_{dbes}-calculus is stated (Propositions 5.30 and 5.31). This allows us to prove confluence of the full calculus with explicit substitutions. Finally, we shall prove the property of preservation of strong normalization, that is, that every strongly normalizing term in ς^{f}_{db}-calculus must also be strongly normalizing in the ς_{dbes}-calculus. Since we allow some interaction between substitutions this property is essential in our current setting.

For the proof of confluence we shall use the interpretation method; the proof of preservation of strong normalization is based on the technique introduced by R.Bloo and H.Geuvers in [BH98, Blo97].

5.5.1 Confluence

Confluence shall be the first of the properties we shall look at. We shall use the interpretion method which requires that we study how ς^f_{db} may be simulated in ς_{dbes}, and viceversa (via some appropriate interpretation function). For this we shall use the BES-calculus to interpret terms in $\mathcal{T}_{\varsigma_{dbes}}$ into terms in $\mathcal{T}_{\varsigma^f_{db}}$. Figure 5.3 pictures the diagram we shall complete in this section. Diagrams 1 and 2 of this figure shall be closed by Proposition 5.31, namely that each ς_{dbes}-rewrite step may be projected via BES interpretation into a ς^f_{db}-derivation. Diagram 3 follows from confluence of ς^f_{db} (Lemma 5.15). Finally, the fact that $BES(b_1) \twoheadrightarrow_{\varsigma_{dbes}} c$ and $BES(b_2) \twoheadrightarrow_{\varsigma_{dbes}} c$ follows from $BES(b_1) \twoheadrightarrow_{\varsigma^f_{db}} c$ and $BES(b_2) \twoheadrightarrow_{\varsigma^f_{db}} c$ and Proposition 5.30, since after all ς_{dbes} is a calculus of explicit substitutions for ς^f_{db}.

$$\begin{tikzcd}
 & & a \arrow[dll, "\varsigma_{dbes}"'] \arrow[drr, "\varsigma_{dbes}"] \arrow[dd, "BES"'] & & \\
b_1 \arrow[dd, "BES"'] & 1 & & 2 & b_2 \arrow[dd, "BES"] \\
 & & BES(a) \arrow[dll, "\varsigma^f_{db}"] \arrow[drr, "\varsigma^f_{db}"'] & & \\
BES(b_1) \arrow[drr, dotted, "\varsigma^f_{db}"'] & & 3 & & BES(b_2) \arrow[dll, dotted, "\varsigma^f_{db}"] \\
 & & c & &
\end{tikzcd}$$

Figure 5.3: The Interpretation Method

Strong normalization of BES may be obtained from strong normalization of $ESDB$. The latter result is rather tedious but standard techniques suffice. The details may be found in the appendix (Section A.2.2). Confluence of BES then follows from local confluence (there is no overlapping) by applying Newman's lemma. This entails the following result.

Corollary 5.20 (Uniqueness of BES-normal forms) The BES-normal forms are unique.

Thus we shall use BES-normal forms to interpret terms of $\mathcal{T}_{\varsigma_{dbes}}$ into terms in $\mathcal{T}_{\varsigma^f_{db}}$. We shall now show that BES-normal forms are exactly $\mathcal{T}_{\varsigma^f_{db}}$.

Proposition 5.21 (BES-normal forms are pure terms) The BES-normal forms (of terms of sort Term) are pure terms (of sort Term). Thus if c is a term (of sort Term) then we use $BES(c)$ to denote the BES-normal form of c.

Proof. Pure terms are clearly in BES-normal form. So we must show that every term in BES-normal form is a pure term. We proceed in the style of [Río93]. Suppose $c \in \mathcal{T}_{\varsigma_{dbes}}$ is a BES-normal form, we use induction on c.

- $c = n$. It is clear that for any n, n is a pure term.
- $c = a.l, \varsigma(a), (l \doteq g), (l := a), a \lhd < m >$ or $[m_i^{\,i \in 1..n}]$. Then since c is a BES-normal form then the subterms must be BES-normal forms. Thus, by the induction hypothesis they are pure terms and therefore also c is.
- $c = g[s]$. Suppose $g[s]$ is a BES-normal form. Then g is a BES-normal form, thus, by the induction hypothesis, g is a pure method. But then $g = \varsigma(a)$ for some pure term a, in which case, $\varsigma(a)[s]$ cannot be a BES-normal form due to the presence of the rule SM in BES. Therefore $g[s]$ is not a BES-normal form.

- $c = a[s]$. Since $a[s]$ is a BES-normal form then a must be in BES-normal form. Thus by the induction hypothesis we may assume a is a pure term and therefore cannot be a closure. By SO, SF, SB, SI, SU the term a cannot be $[m_i{}^{i\,\in\,1..n}]$, $a.l$, $(l := a)$, $(l \doteq g)$ *or* $a \lhd < m >$. So a must be an index p and we analyse s.
  - $s \neq\Uparrow (s')$ since $FVarlift, RVarlift$ are in BES.
  - $s \neq @l$ since $FInv, RInv$ are in BES.
  - $s \neq b/$ since $FVar, RVar$ are in BES.
  - $s \neq\uparrow$ since $VarShift$ is in BES.

  Thus $a[s]$ may not be a BES-normal form and therefore this case does not arise.

•

The next step in our studies is to consider how ς^f_{db} may be simulated in ς_{dbes} and viceversa (via BES-interpretation). In order to accomplish such a task we must take a closer look at the relation between explicit substitutions in the ς_{dbes}-calculus and their implicit (or metalevel) counterparts. This concerns not only usual substitution but also invoke substitutions. We shall thus continue with some technical results on invoke substitutions and then resume (with Lemma 5.28) our analysis between explicit and implicit substitutions.

We shall use $\bullet \ll \bullet \leftarrow \bullet \gg$ to denote invoke substitution at the metalevel. Intuitively, if a is a term in $\mathcal{T}_{\varsigma^f_{db}}$, i is an index and l is a label, then $a \ll i \leftarrow l \gg$ denotes the term that results by replacing every occurrence of i in a with the method invocation $i.l$. The difference with a term such as $a\{\!\{i \leftarrow i.l\}\!\}$ is that in ordinary substitution all indices greater than i are decremented in one unit. This owes to the fact that an ordinary substitution arises when a binder (such as ς) symbol has been eliminated and thus adjustment of indices is needed. In contrast, since invoke substitution is used in an encoding process and is not generated by a rule eliminating a binder, no such adjustment is necessary. We consider this operation to be a substitution in the sense that constructors such as binders and override operators are traversed until indices are reached, at which point the replacement takes place.

Definition 5.22 (Invoke Substitution) Let $a \in \mathcal{T}_{\varsigma^f_{db}}$, l a label, and $n > 0$. Then the invoke substitution of a with l at level n, noted $a \ll n \leftarrow l \gg$, is defined as follows:

$$
\begin{array}{lcl}
(\varsigma(c)) \ll n \leftarrow l \gg & \stackrel{\text{def}}{=} & \varsigma(c \ll n+1 \leftarrow l \gg) \\
[m_i{}^{i\in 1..m}] \ll n \leftarrow l \gg & \stackrel{\text{def}}{=} & [m_i \ll n \leftarrow l \gg {}^{i\in 1..m}] \\
(a.l') \ll n \leftarrow l \gg & \stackrel{\text{def}}{=} & a \ll n \leftarrow l \gg .l' \\
a \lhd < m > \ll n \leftarrow l \gg & \stackrel{\text{def}}{=} & a \ll n \leftarrow l \gg \ \lhd < m \ll n \leftarrow l \gg > \\
(l \doteq \varsigma(c)) \ll n \leftarrow l \gg & \stackrel{\text{def}}{=} & (l \doteq \varsigma(c) \ll n \leftarrow l \gg) \\
(l := b) \ll n \leftarrow l \gg & \stackrel{\text{def}}{=} & (l := b \ll n \leftarrow l \gg) \\
n' \ll n \leftarrow l \gg & \stackrel{\text{def}}{=} & \begin{cases} n'.l & if\ n' = n \\ n' & if\ n' \neq n \end{cases}
\end{array}
$$

Remark 5.23 Note that if $k > i$ then $\mathcal{U}^k_0(b) \ll i \leftarrow l \gg = \mathcal{U}^k_0(b)$ since $\mathcal{U}^k_0(b)$ increases all free indices in b by k units.

The following lemmas shed some light on the interaction between ordinary substitutions and invoke substitutions and the updating functions. All items are proved by induction on a.

Lemma 5.24 For any terms $a, b, c \in \mathcal{T}_{\varsigma^f_{db}}$, labels l, l' and indices i, j, k, n

1. If $i, k > 0$ and $k > i$ then $a \ll i \leftarrow l \gg \{\!\{k \leftarrow b\}\!\} = a\{\!\{k \leftarrow b\}\!\} \ll i \leftarrow l \gg$
2. If $i, k > 0$ and $k > i$ then $a \ll i \leftarrow l \gg \ll k \leftarrow l' \gg = a \ll k \leftarrow l' \gg \ll i \leftarrow l \gg$
3. If $i > 0$ then $a \ll i \leftarrow l \gg \{\!\{i \leftarrow b\}\!\} = a\{\!\{i \leftarrow b.l\}\!\}$
4. If $i, n > 0$, $k \geq 0$ and $i \leq n - k$ then $\mathcal{U}^i_k(a) \ll n \leftarrow b \gg = \mathcal{U}^i_k(a \ll n - i + 1 \leftarrow b \gg)$

5. If $i > 0$ and $1 \le i \le n$ then $a\{\!\{i \leftarrow b\}\!\} \ll n \leftarrow l \gg = a \ll n+1 \leftarrow l \gg \{\!\{i \leftarrow b \ll n-i+1 \leftarrow l \gg\}\!\}$
6. If $i, j > 0$ and $k \ge 0$ and $k \ge i$ then $\mathcal{U}^j_k(a \ll i \leftarrow l \gg) = \mathcal{U}^j_k(a) \ll i \leftarrow l \gg$.

Invoke substitution preserves reduction in the ς^f_{db}-calculus as the following lemma illustrates. It may be proved for $k = 1$ by induction on a (using Lemma 5.24(5)) and then extended to derivations of length $k > 1$.

Lemma 5.25 Let $a, a' \in \mathcal{T}_{\varsigma^f_{db}}$ and $i > 0$ and $k \ge 0$. Then if $a \xrightarrow{k}_{\varsigma^f_{db}} a'$ then $a \ll i \leftarrow l \gg \xrightarrow{k}_{\varsigma^f_{db}} a' \ll i \leftarrow l \gg$.

We may no resume the plan we set out to follow. Our first result relating explicit and implicit substitution considers the updating functions. Ordinary and invoke substitution shall be the subject of the second result, namely Lemma 5.29

Lemma 5.26 (Relation between explicit and implicit substitutions I) For any $c \in \mathcal{T}_{\varsigma^f_{db}}$ and $i \ge 0$ we have, $BES(c[\Uparrow^i(\uparrow)]) = \mathcal{U}^2_i(c)$.

Proof. We use induction on c. And for the base case (c is a de Bruijn index n) we use induction on n.

Lemma 5.27 Let $a \in \mathcal{T}_{\varsigma^f_{db}}$ and $k, i \ge 0$. Then we have $BES(a[\Uparrow^k(\uparrow)]^i) = \mathcal{U}^{i+1}_k(a)$.

Proof. By induction on i using lemmas 5.26 and A.3(3).

Corollary 5.28 Let $a \in \mathcal{T}_{\varsigma_{dbes}}$ and $k, i \ge 0$ we have $BES(a[\Uparrow^k(\uparrow)]^i) = \mathcal{U}^{i+1}_k(BES(a))$.

Proof. By the previous lemma we have $BES(a[\Uparrow^k(\uparrow)]^i) = BES(BES(a)[\Uparrow^k(\uparrow)]^i) = \mathcal{U}^{i+1}_k(BES(a))$.

•

Lemma 5.29 (Relation between explicit and implicit substitutions II) For any $a, b \in \mathcal{T}_{\varsigma_{dbes}}$ and $i \ge 0$ we have:

1. $BES(a[\Uparrow^i(b/)]) = BES(a)\{\!\{i+1 \leftarrow BES(b)\}\!\}$
2. $BES(a[\Uparrow^i(@l)]) = BES(a) \ll i+1 \leftarrow l \gg$

Proof. We prove the first and second item using structural induction on a and considering firstly the case where a is a pure term. Then, in order to complete the proof of these two items, we consider the case where a is not a pure term.
As for ordinary substitution we have

- $a = n$. By Proposition 5.17(2) and the uniqueness of BES-normal forms we have

  $$BES(n[\Uparrow^i(b/)]) = \begin{cases} n-1 & n > i+1 \\ BES(b[\uparrow]^i) & n = i+1 \\ n & n < i+1 \end{cases}$$

  And by the definition of substitution (Def. 5.6) and Corollary 5.28 one may verify that in each case the term is exactly $n\{\!\{i+1 \leftarrow BES(b)\}\!\}$.

- $a = c.l$. Then we have

  $$BES(c.l[\Uparrow^i(b/)]) = BES(c[\Uparrow^i(b/)].l) = BES(c[\Uparrow^i(b/)]).l = c\{\!\{i+1 \leftarrow BES(b)\}\!\}.l = c.l\{\!\{i+1 \leftarrow BES(b)\}\!\}$$

The other cases hold by the induction hypothesis, just as the second case considered above.

Now for the invoke substitution we proceed analogously, considering a pure term a and using structural induction on a.

- $a = n$. By Proposition 5.17(1) and the uniqueness of BES-normal forms we have

  $$BES(n[\Uparrow^i(@l)]) = \begin{cases} n.l & n = i+1 \\ n & n \ne i+1 \end{cases}$$

  And by the definition of invoke substitution (Def. 5.22) we may easily verify that in each case the term is exactly $n \ll i+1 \leftarrow l \gg$.

- $a = c.l'$. Then we have $BES(c.l'[\Uparrow^i (@l)]) = BES(c[\Uparrow^i (@l)].l') = c \ll i+1 \leftarrow l \gg .l' = c.l' \ll i+1 \leftarrow l \gg$

The other cases hold by the induction hypothesis, just as the second case considered above.
Now suppose a is not a pure term, then we have:

- $BES(a[\Uparrow^i (b/)]) = BES(BES(a)[\Uparrow^i (b/)])$ by uniqueness of BES-normal forms. Now since $BES(a)$ is pure (Proposition 5.21) the previous case for pure terms applies and we have by uniqueness of BES-normal forms $BES(BES(a))\{\!\{i+1 \leftarrow BES(b)\}\!\} = BES(a)\{\!\{i+1 \leftarrow BES(b)\}\!\}$.
- $BES(a[\Uparrow^i (@l)]) = BES(BES(a)[\Uparrow^i (@l)])$ by uniqueness of BES-normal forms. Now since $BES(a)$ is pure (Proposition 5.21) the previous case for pure terms applies and we have by uniqueness of BES-normal forms $BES(BES(a)) \ll i+1 \leftarrow l \gg = BES(a) \ll i+1 \leftarrow l \gg$.

The following lemma states that reduction in ς^f_{db} is preserved by the ς_{dbes}-calculus.

Proposition 5.30 (The ς_{dbes}-calculus simulates the ς^f_{db}-calculus) Let a, b be pure terms. If $a \rightarrow_{\varsigma^f_{db}} b$ then $a \twoheadrightarrow_{\varsigma_{dbes}} b$.

Proof. By structural induction on a. For each case we consider the cases where reduction takes place at the root or is internal.

- $a = c.l_j$.
  - The reduction is not at the root: then $c \rightarrow_{\varsigma_{DB}} c'$ and $b = c'.l$ and we may apply the induction hypothesis.
  - The reduction is at the root and $c = [l_j \doteq \varsigma(d), m_i{}^{i \in 1..n, i \neq j}]$ with $1 \leq j \leq n$. Also, $b = d\{\!\{1 \leftarrow c\}\!\}$. Therefore we have, $c.l_j \rightarrow_{MI} d[c/] \twoheadrightarrow_{\varsigma_{dbes}} BES(d[c/])$. By Lemma 5.29(1) and the fact that d and c are pure terms we have $BES(d[c/]) = d\{\!\{1 \leftarrow c\}\!\}$.
  - The reduction is at the root and $c = [l_j := d, m_i{}^{i \in 1..n, i \neq j}]$ with $1 \leq j \leq n$. Also, $b = d$. Therefore we have, $c.l_j \rightarrow_{FI} d$
- $a = \varsigma(c)$. Then the reduction must be internal and we may apply the induction hypothesis.
- $a = c \lhd < m >$.
  - The reduction is not at the root: then $c \rightarrow_{\varsigma^f_{db}} c'$ or $m \rightarrow_{\varsigma^f_{db}} m'$ and we may apply the induction hypothesis.
  - The reduction is at the root and $c = [m_i{}^{i \in 1..n}]$ and $m = (l_j \doteq \varsigma(d))$ with $1 \leq j \leq n$. Then $b = [l_j \doteq \varsigma(d), m_i{}^{i \in 1..n, i \neq j}]$. Therefore we have, $a \rightarrow_{MO} b$.
  - The reduction is at the root and $c = [m_i{}^{i \in 1..n}]$ and $m = (l_j := d)$ with $1 \leq j \leq n$. Then $b = [l_j := d, m_i{}^{i \in 1..n, i \neq j}]$. Therefore we have, $a \rightarrow_{FO} b$.
- $a = [m_i{}^{i \in 1..n}]$. In this case the reduction is internal and we may apply the induction hypothesis.

The remaining cases are similar and may be handled accordingly by making use of the induction hypothesis.

■

Proposition 5.31 (The ς^f_{db}-calculus simulates the ς_{dbes}-calculus) Let $a, b \in \mathcal{T}_{\varsigma_{dbes}}$. If $a \rightarrow_{\varsigma_{dbes}} b$ then $BES(a) \twoheadrightarrow_{\varsigma^f_{db}} BES(b)$. Moreover, if $a \rightarrow_R b$ with $R = \{MI, FI, MO, FO\}$ and the reduction takes place at the root then $BES(a) \rightarrow_{\varsigma^f_{db}} BES(b)$.

Proof. By structural induction on a. For each case we consider the case where reduction takes place at the root, the other cases follow by applying the induction hypothesis.

Suppose that $a \twoheadrightarrow_r b$ with $r \in BES$ then $BES(a) = BES(b)$. Therefore the only interesting cases are those where $r \in \{MI, FI, MO, FO, CO, SW\}$.

- Case $r = MI$. $a = [l_j \doteq \varsigma(b_j), {m_i}^{i \in 1..n, i \neq j}].l_j$ and $b = b_j[[l_j \doteq \varsigma(b_j), {m_i}^{i \in 1..n, i\neq j}]/]$. Then

$$\begin{array}{lll} BES(a) & = & [l_j \doteq \varsigma(BES(b_j)), BES(m_i)^{\ i \in 1..n, i\neq j}].l_j \\ & \rightarrow_{\varsigma^{f}_{db}} & BES(b_j)\{\!\{1 \leftarrow BES([l_j \doteq \varsigma(b_j), {m_i}^{i \in 1..n, i\neq j}]\}\!\} \\ & =_{L\ 5.29(1)} & BES(b_j[([l_j \doteq \varsigma(b_j), {m_i}^{i \in 1..n, i\neq j}]/] \\ & = & BES(b) \end{array}$$

- Case $r = FI$. $a = [l_j := d, {m_i}^{i \in 1..n, i\neq j}].l_j$ and $b = d$. Then

$$\begin{array}{lll} BES(a) & = & [l_j := BES(d), BES(m_i)^{\ i \in 1..n, i\neq j}].l_j \\ & \rightarrow_{\varsigma^{f}_{db}} & BES(d) \\ & = & BES(b) \end{array}$$

- Case $r = MO$. $a = c \lhd < l_j \doteq f >$ where $c = [{m_i}^{i \in 1..n}]$, and $b = [l_j \doteq f, {m_i}^{i \in 1..n, i\neq j}]$. Then we have

$$\begin{array}{lll} BES(c \lhd < l_j \doteq f >) & = & BES(c) \lhd < l_j \doteq BES(f) > \\ & = & [BES(m_i)^{\ i\in 1..n}] \lhd < l_j \doteq BES(f) > \\ & \rightarrow_{\varsigma^{f}_{db}} & [l_j \doteq BES(f), BES(m_i)^{\ i\in 1..n, i\neq j}] \\ & = & BES(b) \end{array}$$

- Case $r = FO$. $a = c \lhd < l_j := d >$ where $c = [{m_i}^{i \in 1..n}]$, and $b = [l_j := d, {m_i}^{i\in 1..n, i\neq j}]$. Then we have

$$\begin{array}{lll} BES(c \lhd < l_j := d >) & = & BES(c) \lhd < l_j := BES(d) > \\ & = & [BES(m_i)^{\ i\in 1..n}] \lhd < l_j := BES(d) > \\ & \rightarrow_{\varsigma^{f}_{db}} & [l_j := BES(d), BES(m_i)^{\ i \in 1..n, i\neq j}] \\ & = & BES(b) \end{array}$$

- Case $r = \dot{CO}$. $a = c[\Uparrow^i (@l_j)][\Uparrow^i ([l_j := d, {m_i}^{i\in 1..n, i\neq j}]/)]$ and $b = c[\Uparrow^i (d/)]$.
  Now by Lemma 5.29 we have:

$$\begin{array}{lll} BES(a) & = & BES(c) \ll i+1 \leftarrow l_j \gg \{\!\{i+1 \leftarrow [l_j := BES(d), BES(m_i)^{\ i\in 1..n, i\neq j}]\}\!\} \\ & =_{L\ 5.24(3)} & BES(c)\{\!\{i+1 \leftarrow [l_j := BES(d), BES(m_i)^{\ i\in 1..n, i\neq j}].l_j\}\!\} \end{array}$$

  Note that $[l_j := BES(d), BES(m_i)^{\ i\in 1..n, i\neq j}].l_j \rightarrow_{\varsigma^{f}_{db}} BES(d)$. Thus by Lemma A.5(2) we have

$$\begin{array}{lll} BES(c)\{\!\{i+1 \leftarrow [l_j := BES(d), BES(m_i)^{\ i\in 1..n, i\neq j}].l_j\}\!\} & \twoheadrightarrow_{\varsigma^{f}_{db}} & BES(c)\{\!\{i+1 \leftarrow BES(d)\}\!\} \\ & =_{L\ 5.29(1)} & BES(c[\Uparrow^i (d/)]) \\ & = & BES(b) \end{array}$$

- Case $r = SW$. $a = c[\Uparrow^i (@l)][\Uparrow^k (s)]$ and $b = c[\Uparrow^k (s)][\Uparrow^i (@l)]$ where $k > i$. We shall analyse each possible form of s.
  - $s = \Uparrow^n (b/)$. Then $BES(a) = BES(c) \ll i+1 \leftarrow l \gg \{\!\{k+n+1 \leftarrow BES(b)\}\!\}$ and also $BES(b) = BES(c)\{\!\{k+n+1 \leftarrow BES(b)\}\!\} \ll i+1 \leftarrow l \gg$. Both terms are equal by Lemma 5.24(1).
  - $s = \Uparrow^n (\uparrow)$. On one hand we have

$$\begin{array}{lll} BES(a) & = & BES(BES(c[\Uparrow^i (@l)])[\Uparrow^{k+n} (\uparrow)]) \\ & =_{L\ 5.28} & \mathcal{U}^2_{k+n}(BES(c[\Uparrow^i (@l)])) \\ & =_{unique\ BES-nf} & \mathcal{U}^2_{k+n}(BES(BES(c)[\Uparrow^i (@l)])) \\ & =_{L\ 5.29(2)} & \mathcal{U}^2_{k+n}(BES(c) \ll i+1 \leftarrow l \gg) \end{array}$$

  On the other hand we have

$$\begin{array}{lll} BES(b) & = & BES(BES(c[\Uparrow^{k+n}(\uparrow)])[\Uparrow^{i}(@l)]) \\ & =_{L\ 5.29(2)} & BES(c[\Uparrow^{k+n}(\uparrow)]) \ll i+1 \leftarrow l \gg \\ & =_{unique\ BES-nf} & BES(BES(c)[\Uparrow^{k+n}(\uparrow)]) \ll i+1 \leftarrow l \gg \\ & =_{L\ 5.28} & \mathcal{U}^2_{k+n}(BES(c)) \ll i+1 \leftarrow l \gg \end{array}$$

Due to the restriction $k > i$, it follows that $k + n \geq i + 1$. Then by applying Lemma 5.24(6) both terms are equal.

- $s = \Uparrow^n (@l')$. On one hand we have

$$\begin{array}{lll} BES(a) & = & BES(BES(c[\Uparrow^{i}(@l)])[\Uparrow^{k+n}(@l')]) \\ & =_{L\ 5.29(2)} & BES(c[\Uparrow^{i}(@l)]) \ll k+n+1 \leftarrow l' \gg \\ & =_{L\ 5.29(2)} & BES(c) \ll i+1 \leftarrow l \gg \ll k+n+1 \leftarrow l' \gg \end{array}$$

On the other hand we have $BES(b) = BES(c) \ll k+n+1 \leftarrow l' \gg \ll i+1 \leftarrow l \gg$. Both terms are equal by Lemma 5.24(2).

Theorem 5.32 (Confluence of ς_{dbes}-calculus) The ς_{dbes}-calculus is confluent.

Proof. Let a, b, c be terms of sort Term in $\mathcal{T}_{\varsigma_{dbes}}$ such that $a \twoheadrightarrow_{\varsigma_{dbes}} b$ and $a \twoheadrightarrow_{\varsigma_{dbes}} c$. Then by Proposition 5.31 we have $BES(a) \twoheadrightarrow_{\varsigma^f_{db}} BES(b)$ and $BES(a) \twoheadrightarrow_{\varsigma^f_{db}} BES(c)$. Since by Lemma 5.15 the ς^f_{db}-calculus is confluent we may obtain a pure term d such that $BES(b) \twoheadrightarrow_{\varsigma^f_{db}} d$ and $BES(c) \twoheadrightarrow_{\varsigma^f_{db}} d$, and by Proposition 5.30 we may close the diagram by $b \twoheadrightarrow_{BES} BES(b) \twoheadrightarrow_{\varsigma_{dbes}} d$ and $c \twoheadrightarrow_{BES} BES(c) \twoheadrightarrow_{\varsigma_{dbes}} d$. See Figure 5.3

5.5.2 Preservation of Strong Normalization

Preservation of strong normalization for the ς_{dbes}-calculus is the last property we shall look at. We shall use a technique due to Bloo and Geuvers [BH98]. As remarked before, this property is an essential ingredient in any explicit substitution implementation of a calculus, more so if there is some form of interaction between substitutions as is our case.

The idea is to define a subset $\mathcal{F}$ of terms in $\mathcal{T}_{\varsigma_{dbes}}$ which is closed under ς_{dbes}-reduction and which contains all the pure terms which admit no finite ς^f_{db}-derivations. Then, one defines a translation $\mathcal{S}(\bullet)$ from terms in $\mathcal{F}$ to terms in a set $\mathcal{T}_l$, the latter of which are equipped with a well-founded order $\succ_{\mathcal{T}_l}$. Finally, it is shown that if $a \rightarrow_{\varsigma_{dbes}} b$ for $a \in \mathcal{F}$, then $\mathcal{S}(a)$ is strictly greater than $\mathcal{S}(b)$ in this order $\succ_{\mathcal{T}_l}$. In full rigour we shall see that ς_{dbes} may be partitioned into two subsystems, say R_1 and R_2, with R_2 strongly normalizing. Then it is shown that if $a \rightarrow_{R_1} b$ for $a \in \mathcal{F}$, then $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(b)$ and if $a \rightarrow_{R_2} b$ for $a \in \mathcal{F}$, then $\mathcal{S}(a) \succeq_{\mathcal{T}_l} \mathcal{S}(b)$. By Lemma 2.6 this suffices for our purpose.

We recall the definition of $maxred_\bullet(\bullet)$ from Chapter 2, more precisely of $maxred_{\varsigma^f_{db}}(\bullet)$.

Def. 2.5.
We define the function $maxred_{\varsigma^f_{db}}(\bullet) : \mathcal{T}_{\varsigma^f_{db}} \rightarrow \mathbb{N} \cup \{\infty\}$ as:

$$maxred_{\varsigma^f_{db}}(a) \overset{\text{def}}{=} \begin{cases} n & \text{if there is a derivation } a \rightarrow_{\varsigma^f_{db}} a_1 \rightarrow_{\varsigma^f_{db}} a_2 \ldots \rightarrow_{\varsigma^f_{db}} a_n \\ & \text{such that for any derivation } a \rightarrow_{\varsigma^f_{db}} a'_1 \rightarrow_{\varsigma^f_{db}} a'_2 \ldots \rightarrow_{\varsigma^f_{db}} a'_m \text{ we have } m \leq n \\ \infty & \text{otherwise} \end{cases}$$

Thus if a is a term in $\mathcal{T}_{\varsigma^f_{db}}$, then if a is strongly normalizing, $maxred_{\varsigma^f_{db}}(a)$ returns the length of the longest ς^f_{db}-reduction sequence from a otherwise it returns the special symbol ∞. Below we state some properties satisfled by this function.

Lemma 5.33 (Properties of $maxred_{\varsigma^f_{db}}(\bullet)$) Let $a \in \mathcal{T}_{\varsigma^f_{db}}$, $i > 0$ and $k \geq 0$. And suppose $maxred_{\varsigma^f_{db}}(a) < \infty$ and $maxred_{\varsigma^f_{db}}([m_i{}^{i \in 1..n}]) < \infty$. Then we have:

1. $maxred_{\varsigma^f_{db}}(a) = maxred_{\varsigma^f_{db}}(\varsigma(a))$

2. $\mathit{maxred}_{\varsigma^f_{db}}(m_j) \leq \mathit{maxred}_{\varsigma^f_{db}}([m_i{}^{i\in 1..n}])$ for each $j \in 1..n$
3. $\mathit{maxred}_{\varsigma^f_{db}}(a)=\mathit{maxred}_{\varsigma^f_{db}}(l := a)$, $\mathit{maxred}_{\varsigma^f_{db}}(g)=\mathit{maxred}_{\varsigma^f_{db}}(l \doteq g)$
4. $\mathit{maxred}_{\varsigma^f_{db}}(a) \leq \mathit{maxred}_{\varsigma^f_{db}}(a.l)$
5. $\mathit{maxred}_{\varsigma^f_{db}}(a)=\mathit{maxred}_{\varsigma^f_{db}}(\mathcal{U}^i_k(a))$
6. $\mathit{maxred}_{\varsigma^f_{db}}(a) \leq \mathit{maxred}_{\varsigma^f_{db}}(a \lhd < m >)$ and $\mathit{maxred}_{\varsigma^f_{db}}(m) \leq \mathit{maxred}_{\varsigma^f_{db}}(a \lhd < m >)$
7. $\mathit{maxred}_{\varsigma^f_{db}}(a) = \mathit{maxred}_{\varsigma^f_{db}}(a \ll i \leftarrow l \gg)$

Note that all items but for (5) and (7) are direct. One observes that the argument of left-hand side (for example, a in the first item) is included in the argument of the right-hand side ($\varsigma(a)$), and moreover, no reduction rules apply at the root of the argument (in our example $\varsigma(a)$). The intuition behind the proofs of items (5) and (7) is that the updating functions as well as the invoke substitution operator do not introduce new redexes. The former merely adjusts indices and the latter modifies a term by substituting occurrences of an indice, say n, with occurrences of $n.l$. Thus informally, a and $\mathcal{U}^i_k(a)$ have the same set of redexes, likewise for a and $a \ll i \leftarrow l \gg$. These proofs require the development of additional lemmas that we shall tackle below, namely Lemma 5.34, 5.35 and 5.36.

Lemma 5.34 Let $a \in \mathcal{T}_{\varsigma^f_{db}}$, $j > 0$ and $k \geq 0$. Then we have the following:

- if $\mathcal{U}^j_k(a) = [m_i{}^{i\in 1..n}]$. Then $a = [m'_i{}^{i\in 1..n}] \in \mathcal{T}_{\varsigma^f_{db}}$ where $\mathcal{U}^j_k(m'_i) = m_i$.
- if $\mathcal{U}^j_k(a) = (l \doteq \varsigma(b))$. Then $a = (l \doteq \varsigma(b')) \in \mathcal{T}_{\varsigma^f_{db}}$ where $\mathcal{U}^j_{k+1}(b') = b$.
- if $\mathcal{U}^j_k(a) = (l := b)$. Then $a = (l := b') \in \mathcal{T}_{\varsigma^f_{db}}$ where $\mathcal{U}^j_k(b') = b$.
- if $\mathcal{U}^j_k(a) = \varsigma(b)$. Then $a = \varsigma(b') \in \mathcal{T}_{\varsigma^f_{db}}$ where $\mathcal{U}^j_{k+1}(b') = b$.

Proof. By a close inspection of the clauses defining the updating functions.

Lemma 5.35 Let $a \in \mathcal{T}_{\varsigma^f_{db}}$ and $j > 0$. Then we have the following:

- if $a \ll j \leftarrow l \gg = [m_i{}^{i\in 1..n}]$. Then $a = [m'_i{}^{i\in 1..n}] \in \mathcal{T}_{\varsigma^f_{db}}$ where $m'_i \ll j \leftarrow l \gg = m_i$.
- if $a \ll j \leftarrow l \gg = (l' \doteq \varsigma(b))$. Then $a = (l' \doteq \varsigma(b')) \in \mathcal{T}_{\varsigma^f_{db}}$ where $b' \ll j+1 \leftarrow l \gg = b$.
- if $a \ll j \leftarrow l \gg = (l' := b)$. Then $a = (l' := b') \in \mathcal{T}_{\varsigma^f_{db}}$ where $b' \ll j \leftarrow l \gg = b$.
- if $a \ll j \leftarrow l \gg = \varsigma(b)$. Then $a = \varsigma(b') \in \mathcal{T}_{\varsigma^f_{db}}$ where $b' \ll j \leftarrow l \gg = b$.

Proof. By a close inspection of the clauses defining the invoke substitution.

Lemma 5.36 Let $a, b \in \mathcal{T}_{\varsigma^f_{db}}$, $j > 0$ and $k \geq 0$.

1. If $\mathcal{U}^j_k(a) \to_{\varsigma^f_{db}} b$ then there exists $c \in \mathcal{T}_{\varsigma^f_{db}}$ such that $b = \mathcal{U}^j_k(c)$ and $a \to_{\varsigma^f_{db}} c$.
2. If $a \ll j \leftarrow l \gg \to_{\varsigma^f_{db}} b$ then there exists $c \in \mathcal{T}_{\varsigma^f_{db}}$ such that $b = c \ll j \leftarrow l \gg$ and $a \to_{\varsigma^f_{db}} c$.

Proof. By induction on a using lemmas 5.34 and A.3(4) for the first item, and lemmas 5.34 and 5.24(4) for the second item.

- $a = n$. Trivial since there is no redex.
- $a = d.l_j$. Then $\mathcal{U}^j_k(d.l_j) = \mathcal{U}^j_k(d).l_j \to_{\varsigma^f_{db}} b$. Thus we have three cases to consider:

- The reduction is internal. Thus $\mathcal{U}^j_k(d) \to_{\varsigma'_{db}} d'$ and $b = d'.l_j$. Then by the induction hypothesis there is a $c' \in \mathcal{T}_{\varsigma'_{db}}$ such that $d' = \mathcal{U}^j_k(c')$ and $d \to_{\varsigma'_{db}} c'$. Then we take $c = c'.l_j$. Note that since $d \to_{\varsigma'_{db}} c'$ we have $d.l_j \to_{\varsigma'_{db}} c'.l_j$, and also $\mathcal{U}^j_k(c'.l_j) = \mathcal{U}^j_k(c').l_j = d'.l_j = b$.
- The reduction takes place at the root and $\mathcal{U}^j_k(d) = [l_j \doteq \varsigma(e), {m_i}^{i\in 1..n, i\neq j}]$ and $b = e\{\!\{1 \leftarrow \mathcal{U}^j_k(d)\}\!\}$. Now by Lemma 5.34 there is an e' and m'_i with $i \in 1..n, i \neq j$ such that $d = [l_j \doteq \varsigma(e'), {m'_i}^{i\in 1..n, i\neq j}]$ with $e = \mathcal{U}^j_{k+1}(e')$ and $m_i = \mathcal{U}^j_k(m'_i)$. Then we have,

$$\begin{array}{lcl} b & = & \mathcal{U}^j_{k+1}(e')\{\!\{1 \leftarrow \mathcal{U}^j_k(d)\}\!\} \\ & =_{L\ A.3(4)(n=1)} & \mathcal{U}^j_k(e'\{\!\{1 \leftarrow d\}\!\}) \end{array}$$

  Thus we take $c = e'\{\!\{1 \leftarrow d\}\!\}$.
- The reduction is at the root and $\mathcal{U}^j_k(d) = [l_j := e, {m_i}^{i\in 1..n, i\neq j}]$ and $b = e$. Now by Lemma 5.34 there is an e' and m'_i with $i \in 1..n, i \neq j$ such that $d = [l_j := e', {m'_i}^{i\in 1..n, i\neq j}]$ with $e = \mathcal{U}^j_k(e')$ and $m_i = \mathcal{U}^j_k(m'_i)$. Then since $d.l_j \to_{\varsigma'_{db}} e'$ we take $c = e'$.

- $a = [{m_i}^{i\in 1..n}]$. Then $a = \mathcal{U}^j_k([{m_i}^{i\in 1..n}]) = [\mathcal{U}^j_k(m_i)\ ^{i\in 1..n}]$. So the reduction must be internal. Thus $b = [b', \mathcal{U}^j_k(m_i)\ ^{i\in 1..n, i\neq h}]$ with $\mathcal{U}^j_k(m_h) \to_{\varsigma'_{db}} b'$. Then by induction hypothesis there is a c' such that $b' = \mathcal{U}^j_k(c')$ and $m_h \to_{\varsigma'_{db}} c'$. So we take $c = [c', {m_i}^{i\in 1..n, i\neq h}]$.

- $a = d \lhd < m >$. Then $\mathcal{U}^j_k(a) = \mathcal{U}^j_k(d) \lhd < \mathcal{U}^j_k(m) >$. Thus we have three further cases to consider,

  - The reduction is internal. Then $\mathcal{U}^j_k(d) \to_{\varsigma'_{db}} e$ (the case where $\mathcal{U}^j_k(m) \to_{\varsigma'_{db}} e$ is treated similarly). Then by induction hypothesis there is a c' such that $e = \mathcal{U}^j_k(c')$ and $d \to_{\varsigma'_{db}} c'$. In which case we take $c = c' \lhd < m >$.
  - The reduction is at the root and $\mathcal{U}^j_k(d) = [{m_i}^{i\in 1..n}]$ and $\mathcal{U}^j_k(m) = (l_h \doteq \varsigma(e))$ and $b = [l_h \doteq \varsigma(e), {m_i}^{i\in 1..n, i\neq h}]$. Now by Lemma 5.34 we have $d = [{m'_i}^{i\in 1..n}]$ where $m_i = \mathcal{U}^j_k(m'_i)$, and also, $m = (l_h \doteq \varsigma(e'))$ where $e = \mathcal{U}^j_{k+1}(e')$. Then we take $c = [l_h \doteq \varsigma(e'), {m'_i}^{i\in 1..n, i\neq h}]$.
  - The reduction is at the root and $\mathcal{U}^j_k(d) = [{m_i}^{i\in 1..n}]$ and $\mathcal{U}^j_k(m) = (l_h := e)$ and $b = [l_h := e, {m_i}^{i\in 1..n, i\neq h}]$. We proceed as in the previous case.

In the remaining cases the reduction must be internal and we proceed as above.

For the second item the proof is in the line of the proof of Lemma 5.36. The only difference is that in the case where $a = d.l_j$ instead of using the Lemma A.3(4) we use Lemma 5.24(4).

- $a = d.l_j$. Then $(d.l_j) \ll j \leftarrow l \gg = d \ll j \leftarrow l \gg .l_j \to_{\varsigma'_{db}} b$. Thus we have three cases to consider:

  - The reduction is internal. Thus $d \ll j \leftarrow l \gg \to_{\varsigma'_{db}} d'$ and $b = d'.l_j$. Then by the induction hypothesis there is a $c' \in \mathcal{T}_{\varsigma'_{db}}$ such that $d' = c' \ll j \leftarrow l \gg$ and $d \to_{\varsigma'_{db}} c'$. Then we take $c = c'.l_j$. Note that since $d \to_{\varsigma'_{db}} c'$ we have $d.l_j \to_{\varsigma'_{db}} c'.l_j$, and also $(c'.l_j) \ll j \leftarrow l \gg = c' \ll j \leftarrow l \gg .l_j = d'.l_j = b$.
  - The reduction is at the root and $d \ll j \leftarrow l \gg = [l_j \doteq \varsigma(e), {m_i}^{i\in 1..n, i\neq j}]$ and $b = e\{\!\{1 \leftarrow d \ll j \leftarrow l \gg\}\!\}$. Now by Lemma 5.35 there is an e' and m'_i with $i \in 1..n, i \neq j$ such that $d = [l_j \doteq \varsigma(e'), {m'_i}^{i\in 1..n, i\neq j}]$ with $e = e' \ll j+1 \leftarrow l \gg$ and $m_i = m'_i \ll j \leftarrow l \gg$. Then we have,

$$\begin{array}{lcl} b & = & e' \ll j+1 \leftarrow l \gg \{\!\{1 \leftarrow d \ll j \leftarrow l \gg\}\!\} \\ & =_{L\ 5.24(4)(n=1)} & e'\{\!\{1 \leftarrow d\}\!\} \ll j \leftarrow l \gg \end{array}$$

  Thus we take $c = e'\{\!\{1 \leftarrow d\}\!\}$.
  - The reduction is at the root and $d \ll j \leftarrow l \gg = [l_j := e, {m_i}^{i\in 1..n, i\neq j}]$ and $b = e$. Now by Lemma 5.35 there is an e' and m'_i with $i \in 1..n, i \neq j$ such that $d = [l_j := e', {m'_i}^{i\in 1..n, i\neq j}]$ with $e = e' \ll j \leftarrow l \gg$ and $m_i = m'_i \ll j \leftarrow l \gg$. Then since $d.l_j \to_{\varsigma'_{db}} e'$ we take $c = e'$.

We are now in conditions of proving items (5) and (7) of Lemma 5.33.

Proof. The proof of both items is similar. We concentrate on the first item. Recall that Lemma A.5(3) indicates that if $a \to_{\varsigma_{db}^{f}} a'$ then $\mathcal{U}_k^i(a) \to_{\varsigma_{db}^{f}} \mathcal{U}_k^i(a')$. Thus we have $maxred_{\varsigma_{db}^{f}}(a) \leq maxred_{\varsigma_{db}^{f}}(\mathcal{U}_k^i(a))$. And, by Lemma 5.36(1), we have $maxred_{\varsigma_{db}^{f}}(a) \geq maxred_{\varsigma_{db}^{f}}(\mathcal{U}_k^i(a))$. Hence $maxred_{\varsigma_{db}^{f}}(a) = maxred_{\varsigma_{db}^{f}}(\mathcal{U}_k^i(a))$.

•

Definition 5.37 (SN pure terms of $\mathcal{T}_{\varsigma_{dbes}}$) Let $SN_{\varsigma_{db}^{f}}$ denote the set of all the ς_{db}^{f}-strongly normalizing pure terms of $\mathcal{T}_{\varsigma_{dbes}}$.

Then we may define $\mathcal{F}$ as $\mathcal{F} = \{a \in \mathcal{T}_{\varsigma_{dbes}} |$ for all $b \subseteq a$ of sort $\texttt{Term}, BES(b) \in SN_{\varsigma_{db}^{f}}\}$.

Next we show that $\mathcal{F}$ is closed with respect to reduction in the ς_{dbes}-calculus. We recall the reader that we write $b \subseteq a$ to indicate that b is a subterm of a.

Lemma 5.38 ($\mathcal{F}$ is closed under ς_{dbes}-rewriting) Let $a, b \in \mathcal{T}_{\varsigma_{db}^{f}}$. If $a \in \mathcal{F}$ and $a \to_{\varsigma_{dbes}} b$ then $b \in \mathcal{F}$.

Proof. We show that for every $e \subseteq b$ we have $BES(e) \in SN_{\varsigma_{db}^{f}}$. The proof is by induction on a.

- $a = n$. Trivial since there is no redex.
- $a = c.l$. Then there are three subcases to consider
  - The reduction is internal. Thus $c \to_{\varsigma_{dbes}} c'$ and since $c \in \mathcal{F}$ (since $a \in \mathcal{F}$) by the induction hypothesis we obtain that $c' \in \mathcal{F}$. It remains to see that $BES(c'.l) \in SN_{\varsigma_{db}^{f}}$. Thus suppose that $BES(c'.l) \notin SN_{\varsigma_{db}^{f}}$, then since $a \to_{\varsigma_{dbes}} b$ we have by Proposition 5.31 that $BES(a) \twoheadrightarrow_{\varsigma_{db}^{f}} BES(b)$. But then $BES(a) \notin SN_{\varsigma_{db}^{f}}$, contradicting the hypothesis that $a \in \mathcal{F}$.
  - The reduction takes place at the root, $c = [l_j \doteq \varsigma(d), m_i{}^{i \in 1..n, i \neq j}]$ and $l = l_j$. Then $b = d[[l_j \doteq \varsigma(d), m_i{}^{i \in 1..n, i \neq j}]/]$. If $e \subseteq d$ or $e \subseteq c$ then since $a \in \mathcal{F}$ we are done. Suppose then that $e = b$. Then by Proposition 5.31 we have $BES(a) \twoheadrightarrow_{\varsigma_{db}^{f}} BES(b)$. Therefore since $a \in \mathcal{F}$ it must be that $BES(b) \in SN_{\varsigma_{db}^{f}}$.
  - The reduction is at the root and $c = [l_j := d, m_i{}^{i \in 1..n, i \neq j}]$. Then $b = d$ and since $b \subseteq a$ we are done.
- $a = c \lhd < m >$. Then there are three subcases we should consider
  - The reduction is internal. Here we have either $c \to_{\varsigma_{dbes}} c'$ or $m \to_{\varsigma_{dbes}} m'$. In both cases we use the induction hypothesis.
  - The reduction is at the root, $c = [m_i{}^{i \in 1..n}]$ and $m = (l_j \doteq \varsigma(d))$ and $b = [l_j \doteq \varsigma(d), m_i\ i \in 1..n, i \neq j]$. Now if $e \subseteq (l_j \doteq \varsigma(d))$ or $e \subseteq m_i$ with $i \in 1..n, i \neq j$ then since $a \in \mathcal{F}$ we are done. If $e = b$ we proceed as in the previous cases.
  - The reduction is at the root, $c = [m_i{}^{i \in 1..n}]$ and $m = (l_j := d)$. Similar to the previous case.
- $a = c[s]$. Then we must consider the following subcases
  - The reduction is internal. Here it may occur that $c \to_{\varsigma_{dbes}} c'$ or that $s = \Uparrow^i (d/)$ and $d \to_{\varsigma_{dbes}} d'$. In both cases we conclude as above.
  - The reduction is at the root. Here we consider each possible rule applied,
    - * SM. Thus $c = \varsigma(d)$. We must consider,
      - · $e \subseteq d$. We use the induction hypothesis.
      - · $e \subseteq s$. We use the induction hypothesis.
      - · $e = d[\Uparrow (s)]$. Note that $BES(a) = BES(\varsigma(d)[s]) = BES(\varsigma(e)) \doteq \varsigma(BES(e))$. Then since $BES(a) \in SN_{\varsigma_{db}^{f}}$ it must be that $BES(e) \in SN_{\varsigma_{db}^{f}}$.
      - · $e = b$. Using Proposition 5.31 and hypothesis $a \in \mathcal{F}$ as before.

    The rules SO, SF, SB, SI, SU are treated similarly.

  - $FVar$. Direct since $b \subset a$.
  - $LVar$. Direct since indices are strongly normalizing. The same applies to $FInv, RInv, FVarLift$ and $VarShift$.
  - $RVarLift$. The interesting case is $e = n[s]$. Then we have $BES(a) = BES(b) =_{L\ 5.26} \mathcal{U}_0^2(BES(n[s]))$. Now $BES(n[s])$ must be strongly normalizing for otherwise using Lemma A.5(3) we would contradict the strong normalization of $BES(a)$.
  - CO. This rule presents no problems.
  - SW. Then $c = a'[\Uparrow^i (@l)]$ and $s = \Uparrow^k (s')$. If $e \subseteq a'$ or $e \subseteq s'$ then we use the hypothesis. If $e = b$ then since $BES(a) = BES(b)$ (see case SW in the proof of Proposition 5.31) we may use the hypothesis. As for the case $e = a'[\Uparrow^k (s')]$, we have $BES(b) =_{L\ 5.29(2)} BES(a'[\Uparrow^k (s')]) \ll i+1 \leftarrow l \gg$. Then $BES(a'[\Uparrow^k (s')])$ must be strongly normalizing for otherwise by Lemma 5.25 we would contradict the strong normalization of $BES(a) = BES(b)$.

- $a = \varsigma(c)$.Then the reduction must be internal, ie. $c \rightarrow_{\varsigma_{dbes}} c'$. As before, since $a \in \mathcal{F}$ we have $c' \in \mathcal{F}$. For the interesting case $e = b$ we proceed as above: since no rule may be applied at the root of b by Proposition 5.31 and the hypothesis $a \in \mathcal{F}$ we may conclude.

- $a = [m_i{}^{i \in 1..n}]$. Then the reduction must be internal, ie. $m_j \rightarrow_{\varsigma^f_{db}} m'_j$ for some $j \in 1..n$. We use the induction hypothesis.

The remaining cases may be dealt with similarly.

∎

Lemma 5.39 Let $a, b \in \mathcal{T}_{\varsigma^f_{db}}$. Then $a \rightarrow_{\varsigma^f_{db}} b$ implies $\mathit{maxred}_{\varsigma^f_{db}}(a) \geq \mathit{maxred}_{\varsigma^f_{db}}(b)$. If $\mathit{maxred}_{\varsigma^f_{db}}(a) < \infty$ then $\mathit{maxred}_{\varsigma^f_{db}}(a) > \mathit{maxred}_{\varsigma^f_{db}}(b)$.

Proof. The interesting case is the second statement. Suppose that $\mathit{maxred}_{\varsigma^f_{db}}(a) < \infty$, say $\mathit{maxred}_{\varsigma^f_{db}}(a) = n$. Then $\mathit{maxred}_{\varsigma^f_{db}}(b) \neq \infty$, thus we may assume that $\mathit{maxred}_{\varsigma^f_{db}}(b) = k$ and therefore there exists a derivation $b \rightarrow_{\varsigma^f_{db}} b_1 \rightarrow_{\varsigma^f_{db}} b_2 \ldots \rightarrow_{\varsigma^f_{db}} b_k$ which is maximum. Suppose $k \geq n$, then we would obtain the derivation $a \rightarrow_{\varsigma^f_{db}} b \rightarrow_{\varsigma^f_{db}} b_1 \rightarrow_{\varsigma^f_{db}} b_2 \ldots \rightarrow_{\varsigma^f_{db}} b_k$ of length $k+1$ greater than n.

∎

Lemma 5.39 generalizes to one or more ς^f_{db}-rewrite steps as follows.

Corollary 5.40 Let $a, b \in \mathcal{T}_{\varsigma^f_{db}}$ and $\mathit{maxred}_{\varsigma^f_{db}}(a) < \infty$ then $a \xrightarrow{+}_{\varsigma^f_{db}} b$ implies $\mathit{maxred}_{\varsigma^f_{db}}(a) > \mathit{maxred}_{\varsigma^f_{db}}(b)$.

We now move on to labelled terms. Recall that the aim is to define a set of labelled terms $\mathcal{T}_l$ equipped with a well-founded ordering $\succ_{\mathcal{T}_l}$, and a translation $\mathcal{S}(\bullet)$ from terms in $\mathcal{F}$ to terms in $\mathcal{T}_l$. We shall then show that

1. if $a \in \mathcal{F}$ then $a \rightarrow_R a'$ implies $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(a')$, where $R = \{MI, FI, MO, FO\}$, and
2. if $a \in \mathcal{F}$ then $a \rightarrow_R a'$ implies $\mathcal{S}(a) \succeq_{\mathcal{T}_l} \mathcal{S}(a')$, where $R = \varsigma_{dbes} - \{MI, FI, MO, FO\}$.

From this we shall obtain PSN reasoning by contradiction (Proposition 5.46).

Definition 5.41 (Labelled terms) We define the set *labelled terms*, denoted $\mathcal{T}_l$, over the alphabet $\mathcal{A} = \{\star, \circ, \bullet._n\bullet, \lhd(\bullet,\bullet), \bullet\langle\bullet\rangle_n, \bullet[\![\bullet]\!]_n, \varsigma(\bullet), [\bullet], \bullet \doteq \bullet, \bullet := \bullet\}$ by the following grammar:

$$\begin{array}{lcl} t & ::= & \star \mid t._n\circ \mid t\langle t\rangle_n \mid t[\![\circ]\!]_n \mid \lhd(t,u) \mid [u_i{}^{i\in 1..n}] \\ u & ::= & \circ \doteq f \mid \circ := t \\ f & ::= & \varsigma(t) \mid f\langle t\rangle_n \end{array}$$

where n is a natural number greater or equal to zero.

Definition 5.42 (Translation from $\mathcal{F}$ to $\mathcal{T}_l$) The translation $\mathcal{S}(\bullet) : \mathcal{F} \rightarrow \mathcal{T}_l$ is defined as follows:

$$\begin{array}{lcll}
\mathcal{S}(n) & \stackrel{\text{def}}{=} & \star & \\
\mathcal{S}([m_i^{i\,\in 1..n}]) & \stackrel{\text{def}}{=} & [\mathcal{S}(m_i)^{i\,\in 1..n}] & \\
\mathcal{S}(l \doteq g) & \stackrel{\text{def}}{=} & \mathcal{S}(l) \doteq \mathcal{S}(g) & \\
\mathcal{S}(l := a) & \stackrel{\text{def}}{=} & \mathcal{S}(l) := \mathcal{S}(a) & \\
\mathcal{S}(\varsigma(a)) & \stackrel{\text{def}}{=} & \varsigma(\mathcal{S}(a)) & \\
\mathcal{S}(a.l) & \stackrel{\text{def}}{=} & \mathcal{S}(a)._n\mathcal{S}(l) & \text{where } n = maxred_{\varsigma^f_{db}}(BES(a.l)) \\
\mathcal{S}(a \lhd < m >) & \stackrel{\text{def}}{=} & \lhd(\mathcal{S}(a), \mathcal{S}(m)) & \\
\mathcal{S}(a[\Uparrow^i(b/)]) & \stackrel{\text{def}}{=} & \mathcal{S}(a)\langle\mathcal{S}(b)\rangle_n & \text{where } n = maxred_{\varsigma^f_{db}}(BES(a[\Uparrow^i(b/)])) \\
\mathcal{S}(a[\Uparrow^i(\uparrow)]) & \stackrel{\text{def}}{=} & \mathcal{S}(a) & \\
\mathcal{S}(a[\Uparrow^i(@l)]) & \stackrel{\text{def}}{=} & \mathcal{S}(a)\llbracket\mathcal{S}(l)\rrbracket_n & \text{where } n = maxred_{\varsigma^f_{db}}(BES(a[\Uparrow^i(@l)]))
\end{array}$$

where $\mathcal{S}(l) = \circ$.

We define a precedence (partial ordering) on the set of operators of $\mathcal{A}$ as follows: $\bullet ._{n+1} \bullet \gg \bullet\langle\bullet\rangle_n \gg \bullet\llbracket\bullet\rrbracket_n \gg \bullet ._n \bullet \gg \lhd(\bullet, \bullet) \gg \varsigma(\bullet), \bullet \doteq \bullet, \bullet := \bullet, [\bullet], \star, \circ$. Then since $\gg$ is well-founded the induced Recursive Path Ordering '$\succ_{\mathcal{T}_l}$' defined below is well-founded on $\mathcal{T}_l$ [Der82].

Note that since RPOs are Simplification Orderings [Der82] the subterm property holds, that is, if $s, t \in \mathcal{T}_l$ and s is a proper subterm of t then $t \succ_{\mathcal{T}_l} s$.

Lemma 5.43 Let $a \in \mathcal{F}$. Then $a \to_R a'$ implies $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(a')$ where $R = \{MI, FI, MO, FO\}$.

Proof. The proof is by structural induction on a.

- $a = n$. Trivial since there is no redex.
- $a = b.l$. Then we have three cases to consider.
  - The reduction is internal. Thus $b \to_R b'$ and by induction hypothesis we have $\mathcal{S}(b) \succ_{\mathcal{T}_l} \mathcal{S}(b')$. Note that $\mathcal{S}(b.l) = \mathcal{S}(b)._m\circ$ and $\mathcal{S}(b'.l) = \mathcal{S}(b')._n\circ$. Now by Proposition 5.31 we have that $BES(b) \twoheadrightarrow_{\varsigma^f_{db}} BES(b')$. Therefore we consider two cases:[1]
    - $BES(b) = BES(b')$. Then $m = n$ and therefore we must use the 'equal heads' case for comparing the terms. And in effect, we have $\langle\mathcal{S}(b), \circ\rangle \succ'_{\mathcal{T}_l} \langle\mathcal{S}(b'), \circ\rangle$.
    - $BES(b) \xrightarrow{+}_{\varsigma^f_{db}} BES(b')$. Then by Corollary 5.40 $maxred_{\varsigma^f_{db}}(BES(b)) > maxred_{\varsigma^f_{db}}(BES(b'))$. But then since $._m \gg ._n$ and $\mathcal{S}(b)._m\circ \succ_{\mathcal{T}_l} \mathcal{S}(b')$ and $\mathcal{S}(b)._m\circ \succ_{\mathcal{T}_l} \circ$, we are done.
  - The reduction is at the root and $a = [l_j \doteq \varsigma(c), m_i{}^{i\in 1..n, i\neq j}].l_j$. Then $a' = c[[l_j \doteq \varsigma(c), m_i{}^{i\in 1..n, i\neq j}]/]$. Now $\mathcal{S}(a) = \mathcal{S}([l_j \doteq \varsigma(c), m_i{}^{i\in 1..n, i\neq j}])._m\circ$ and $\mathcal{S}(a') = \mathcal{S}(c)\langle\mathcal{S}([l_j \doteq \varsigma(c), m_i{}^{i\in 1..n, i\neq j}])\rangle_n$ and by Proposition 5.31 and Corollary 5.40 we have $m > n$. Then since $._m \gg <>_n$ and $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(c)$ and $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}([l_j \doteq \varsigma(c), m_i{}^{i\in 1..n, i\neq j}])$ we may conclude $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(a')$.
  - The reduction is at the root and $a = [l_j := c, m_i{}^{i\in 1..n, i\neq j}].l_j$. Then $a' = c$. Now $\mathcal{S}(a) = \mathcal{S}([l_j \doteq \varsigma(c), m_i{}^{i\in 1..n, i\neq j}])._m\circ$ and $\mathcal{S}(a') = \mathcal{S}(c)$ and therefore $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(a')$.
- $a = b \lhd < m >$. Then we have three cases to consider:
  - The reduction is internal in which case either $b \to_R b'$ or $m \to_R m'$. Both are handled as in the previous case but making use of the 'equal heads' case.
  - The reduction is at the root and $m = (l_j \doteq \varsigma(c))$. Then $b = [m_i{}^{i\in 1..n}]$ and $a' = [l_j \doteq \varsigma(c), m_i{}^{i\in 1..n, i\neq j}]$. Therefore $\mathcal{S}(a) = \lhd(\mathcal{S}(b), \mathcal{S}(m))$ and $\mathcal{S}(a') = [\mathcal{S}(l_j \doteq \varsigma(c)), \mathcal{S}(m_i)\ {}^{i\in 1..n, i\neq j}]$. And, since $\lhd(\bullet, \bullet) \gg [.]$ we must verify that $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(l_j \doteq \varsigma(c))$ and $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(m_i)$ with $i \in 1..n, i \neq j$ all of which are valid.

[1] Note that strictly speaking it suffices to consider the case $BES(b) = BES(b')$ since if $t = f(t_1, .., t_n)$ and $s = g(s_1, .., s_m)$ then $\langle t_1, .., t_n\rangle \succ'_{\mathcal{T}_l} \langle s_1, .., s_m\rangle$ implies $t \succ_{\mathcal{T}_l} s_1, .., t \succ_{\mathcal{T}_l} s_m$. We chose to consider both for the sake of clarity.

- The reduction is at the root and $m = (l_j := c)$. Then $b = [m_i{}^{i \in 1..n}]$ and $a' = [l_j := c, m_i{}^{i \in 1..n, i \neq j}]$. Analogous to the previous case.

- $a = [m_i{}^{i \in 1..n}]$. Then the reduction must be internal and we make use of the induction hypothesis.

- $a = b[s]$. Then the reduction must be internal and we have two cases to consider,
  - case $b \rightarrow_R b'$. Then by induction hypothesis $\mathcal{S}(b) \succ_{\mathcal{T}_l} \mathcal{S}(b')$. Now we analyse by cases on s.
    - $s = \Uparrow^i (c/)$. Then $\mathcal{S}(a) = \mathcal{S}(b)\langle\mathcal{S}(c)\rangle_n$ where $n = maxred_{\varsigma^f_{db}}(BES(a))$ and $\mathcal{S}(a') = \mathcal{S}(b')\langle\mathcal{S}(c)\rangle_m$ where $m = maxred_{\varsigma^f_{db}}(BES(a'))$. Now by Proposition 5.31 we have $BES(b) \twoheadrightarrow_{\varsigma^f_{db}} BES(b')$. Therefore we consider,
      - $BES(b) = BES(b')$.Then $m = n$ since $BES(a) =_{L\ 5.29} BES(b)\{\!\{i+1 \leftarrow BES(c)\}\!\} = BES(b')\{\!\{i+1 \leftarrow BES(c)\}\!\} =_{L\ 5.29} BES(a')$. And the fact that $\langle\mathcal{S}(b), \mathcal{S}(c)\rangle \succ'_{\mathcal{T}_l} \langle\mathcal{S}(b'), \mathcal{S}(c)\rangle$ concludes this case.
      - $BES(b) \xrightarrow{+}_{\varsigma^f_{db}} BES(b')$. Then by Lemma A.6(1) we obtain $BES(a) =_{L\ 5.29} BES(b)\{\!\{i+1 \leftarrow BES(c)\}\!\} \xrightarrow{+}_{\varsigma^f_{db}} BES(b')\{\!\{i+1 \leftarrow BES(c)\}\!\} =_{L\ 5.29} BES(a')$. Thus by Corollary 5.40 we have $n > m$. Now since $<>_n \gg <>_m$ and $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(b')$ and $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(c)$ we may conclude $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(a')$.
    - $s = \Uparrow^i (\uparrow)$. Then $\mathcal{S}(a) = \mathcal{S}(b) \succ_{\mathcal{T}_l} \mathcal{S}(b') = \mathcal{S}(a')$.
    - $s = \Uparrow^i (@l)$. Then $\mathcal{S}(a) = \mathcal{S}(b)[\![\circ]\!]_n$ where $n = maxred_{\varsigma^f_{db}}(BES(a))$ and $\mathcal{S}(a') = \mathcal{S}(b')[\![\circ]\!]_m$ where $m = maxred_{\varsigma^f_{db}}(BES(a'))$ and we reason as in the first subcase but making use of Lemma 5.25.
  - case $s = \Uparrow^i (c/)$ and $c \rightarrow_R c'$. Thus by induction hypothesis $\mathcal{S}(c) \succ_{\mathcal{T}_l} \mathcal{S}(c')$. Also, $\mathcal{S}(a) = \mathcal{S}(b)\langle\mathcal{S}(c)\rangle_n$ where $n = maxred_{\varsigma^f_{db}}(BES(a))$ and $\mathcal{S}(a') = \mathcal{S}(b)\langle\mathcal{S}(c')\rangle_m$ where $m = maxred_{\varsigma^f_{db}}(BES(a'))$. Now by Proposition 5.31 we have $BES(c) \twoheadrightarrow_{\varsigma^f_{db}} BES(c')$. Therefore we consider,
    - $BES(c) = BES(c')$. Then $m = n$ since $BES(a) =_{L\ 5.29} BES(b)\{\!\{i+1 \leftarrow BES(c)\}\!\} = BES(b)\{\!\{i+1 \leftarrow BES(c')\}\!\} =_{L\ 5.29} BES(a')$. And as $\langle\mathcal{S}(b), \mathcal{S}(c)\rangle \succ'_{\mathcal{T}_l} \langle\mathcal{S}(b), \mathcal{S}(c')\rangle$, we are done.
    - $BES(c) \xrightarrow{+}_{\varsigma^f_{db}} BES(c')$. Then by Lemma A.6(2) we have $BES(a) =_{L\ 5.29} BES(b)\{\!\{i+1 \leftarrow BES(c)\}\!\} \twoheadrightarrow_{\varsigma^f_{db}} BES(b)\{\!\{i+1 \leftarrow BES(c')\}\!\} =_{L\ 5.29} BES(a')$. Then we must consider two subcases:
      1. $BES(a) \xrightarrow{+}_{\varsigma^f_{db}} BES(a')$. Then by Corollary 5.40 we have $n > m$. And in order to conclude $\mathcal{S}(a) \succ_{\mathcal{T}_l} \mathcal{S}(a')$ we may verify that $\mathcal{S}(b)\langle\mathcal{S}(c)\rangle_n \succ_{\mathcal{T}_l} \mathcal{S}(b)$ and $\mathcal{S}(b)\langle\mathcal{S}(c)\rangle_n \succ_{\mathcal{T}_l} \mathcal{S}(c')$.
      2. $BES(a) = BES(a')$. Then $n = m$ and we may verify that $\langle\mathcal{S}(b), \mathcal{S}(c)\rangle \succ'_{\mathcal{T}_l} \langle\mathcal{S}(b), \mathcal{S}(c')\rangle$.

The remaining cases ($a = (l \doteq g)$, $a = (l := c)$ and $a = \varsigma(c)$) are handled by making use of the induction hypothesis.

■

Lemma 5.44 Let $a \in \mathcal{F}$. Then $a \rightarrow_R a'$ implies $\mathcal{S}(a) \succeq_{\mathcal{T}_l} \mathcal{S}(a')$ where $R = \varsigma_{dbes} - \{MI, FI, MO, FO\}$.

Proof. The proof is by structural induction on a.

- $a = n$. Trivial since there is no redex.

- $a = b.l$. Then the reduction must be internal and therefore $b \rightarrow_R b'$ and by induction hypothesis we have $\mathcal{S}(b) \succeq_{\mathcal{T}_l} \mathcal{S}(b')$. Now by Proposition 5.31 we have $BES(b) \twoheadrightarrow_{\varsigma^f_{db}} BES(b')$. Thus we must consider two cases:
  - $BES(b) = BES(b')$. Then $BES(b.l) = BES(b'.l)$ and $maxred_{\varsigma^f_{db}}(BES(b.l)) = maxred_{\varsigma^f_{db}}(BES(b'.l))$, which allows us to conclude that $\mathcal{S}(b)._n\circ \succeq_{\mathcal{T}_l} \mathcal{S}(b')._n\circ$.
  - $BES(b) \xrightarrow{+}_{\varsigma^f_{db}} BES(b')$. Here we reason as in the corresponding case of the previous lemma.

- $a = b \lhd < m >$. Then the reduction must be internal and have two cases to consider,

  - case $b \rightarrow_R b'$. Thus by the induction hypothesis $\mathcal{S}(b) \succeq_{\mathcal{T}_l} \mathcal{S}(b')$ and therefore one may verify that $\lhd(\mathcal{S}(b), \mathcal{S}(m)) \succeq_{\mathcal{T}_l} \lhd(\mathcal{S}(b'), \mathcal{S}(m))$.
  - case $m \rightarrow_R m'$. Then we use the induction hypothesis as above.

- $a = [m_i{}^{i \in 1..n}]$. Then the reduction must be internal and we make use of the induction hypothesis.
- $a = b[s]$. Then there are three cases to consider,
  - The reduction is internal and $b \rightarrow_R b'$. Then we proceed as in the corresponding case of the previous lemma, considering additionaly the case $BES(b) = BES(b')$ from which we may infer the desired result using the 'equal heads' case.
  - The reduction is internal and $s = \Uparrow^i(c/)$ and $c \rightarrow_R c'$. Then we proceed as in the corresponding case of the previous lemma considering additionaly the case $BES(c) = BES(c')$.
  - The reduction takes place at the root. Then there are several cases we must consider depending on the rule applied,
    - SM. Then $a = \varsigma(c)[s]$ and $a' = \varsigma(c[\Uparrow(s)])$. Now depending on s we have,
      - $s = \Uparrow^i(d/)$. Then we have $\mathcal{S}(a) = (\varsigma(\mathcal{S}(c)))\langle\mathcal{S}(d)\rangle_n$ and $\mathcal{S}(a') = \varsigma(\mathcal{S}(c)\langle\mathcal{S}(d)\rangle_m)$ where $n = maxred_{\varsigma^{f}_{db}}(BES(a))$ and $m = maxred_{\varsigma^{f}_{db}}(BES(c[\Uparrow(s)]))$. Note that,

$$\begin{array}{rcl} n & = & maxred_{\varsigma^{f}_{db}}(BES(\varsigma(c)[s])) \\ & = & maxred_{\varsigma^{f}_{db}}(BES(\varsigma(c[\Uparrow(s)]))) \\ & = & maxred_{\varsigma^{f}_{db}}(\varsigma(BES(c[\Uparrow(s)]))) \\ & =_{L\ 5.33(1)} & maxred_{\varsigma^{f}_{db}}(BES(c[\Uparrow(s)])) \\ & = & m \end{array}$$

        Now since $\langle\rangle_n \gg \varsigma()$ we must verify that $\mathcal{S}(a) = (\varsigma(\mathcal{S}(c)))\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \mathcal{S}(c)\langle\mathcal{S}(d)\rangle_m$ and therefore that $\langle\varsigma(\mathcal{S}(c)), \mathcal{S}(d)\rangle \succeq'_{\mathcal{T}_l} \langle\mathcal{S}(c), \mathcal{S}(d)\rangle$ which is valid.
      - $s = \Uparrow^i(\uparrow)$. Then $\mathcal{S}(a) \doteq \dot{\varsigma}(\mathcal{S}(c)) = \mathcal{S}(a')$.
      - $s = \Uparrow^i(@l)$. This is similar to case when $s = \Uparrow^i(d/)$. Note that $\mathcal{S}(a) = (\varsigma(\mathcal{S}(c)))[\![\circ]\!]_m$ and $\mathcal{S}(a') \doteq \varsigma(\mathcal{S}(c)[\![\circ]\!]_n)$. By a similar argument we have $m = n$ and $[\![]\!]_n \gg \varsigma()$ and we proceed similarly.
  - SO. Then $a = [m_i{}^{i \in 1..n}][s]$ and $a' = [m_i[s]\ {}^{i \in 1..n}]$. Then depending on s we have,
    - $s = \Uparrow^i(d/)$. Then $\mathcal{S}(a) = [\mathcal{S}(m_i)\ {}^{i \in 1..n}]\langle\mathcal{S}(d)\rangle_n$ and $\mathcal{S}(a') = [\mathcal{S}(m_i)\langle\mathcal{S}(d)\rangle_{n_i}^{i \in 1..n}]$ where $n = maxred_{\varsigma^{f}_{db}}(BES(a))$ and $n_i = maxred_{\varsigma^{f}_{db}}(BES(m_i[s]))$. Now since $\langle\rangle_n \gg [.]$ we must verify that $\mathcal{S}(a) = [\mathcal{S}(m_i)\ {}^{i \in 1..n}]\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \mathcal{S}(m_i)\langle\mathcal{S}(d)\rangle_{n_i}$ for each $i \in 1..n$. By Lemma 5.33(2) we have $n \geq n_i$ so we have two further cases to consider,
      1. $n > n_i$. Then since $\langle\rangle_n \gg \langle\rangle_{n_i}$ we verify that $\mathcal{S}(a) = [\mathcal{S}(m_i)\ {}^{i \in 1..n}]\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \mathcal{S}(m_i)$ and $\mathcal{S}(a) = [\mathcal{S}(m_i)\ {}^{i \in 1..n}]\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \mathcal{S}(d)$.
      2. $n = n_i$. We verify that $\langle[\mathcal{S}(m_i)\ {}^{i \in 1..n}], \mathcal{S}(d)\rangle \succeq'_{\mathcal{T}_l} \langle\mathcal{S}(m_i), \mathcal{S}(d)\rangle$.
    - $s = \Uparrow^i(\uparrow)$. Then $\mathcal{S}(a) = [\mathcal{S}(m_i)\ {}^{i \in 1..n}] = \mathcal{S}(a')$.
    - $s = \Uparrow^i(@l)$. This is similar to case when $s = \Uparrow^i(d/)$.
  - SF. Then $a = (l := c)[s]$ and $a' = (l := c[s])$. Then depending on s we have,
    - $s = \Uparrow^i(d/)$. Then $\mathcal{S}(a) = (\circ := \mathcal{S}(c))\langle\mathcal{S}(d)\rangle_n$ and $\mathcal{S}(a') = \circ := (\mathcal{S}(c)\langle\mathcal{S}(d)\rangle_m)$ where $n = maxred_{\varsigma^{f}_{db}}(BES((l := c)[s]))$ and $m = maxred_{\varsigma^{f}_{db}}(BES(c[s]))$. Now since $\langle\rangle_n \gg :=$ we must verify that $\mathcal{S}(a) = (\circ := \mathcal{S}(c))\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \circ$ and that $\mathcal{S}(a) = (\circ := \mathcal{S}(c))\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \mathcal{S}(c)\langle\mathcal{S}(d)\rangle_m$. The first is valid trivially. As for the second item note that,

$$\begin{array}{rcl} n & = & maxred_{\varsigma^{f}_{db}}(BES((l := c)[s])). \\ & = & maxred_{\varsigma^{f}_{db}}(l := BES(c[s])) \\ & =_{L\ 5.33(3)} & maxred_{\varsigma^{f}_{db}}(BES(c[s])) \\ & = & m \end{array}$$

      so we must verify that $\langle\circ := \mathcal{S}(c), \mathcal{S}(d)\rangle \succeq'_{\mathcal{T}_l} \langle\mathcal{S}(c), \mathcal{S}(d)\rangle$ which is valid.
    - $s = \Uparrow^i(\uparrow)$. Then $\mathcal{S}(a) = (\circ := \mathcal{S}(c)) = \mathcal{S}(a')$.

  - $*$ $s =\Uparrow^i(@l)$. This is similar to case when $s =\Uparrow^i(d/)$.

- SB. Then $a = (l = f)[s]$ and $a' = (l = f[s])$ and we proceed as in the previous case.
- SI. Then $a = b.l[s]$ and $a' = b[s].l$. Then depending on s we have,
  - $*$ $s =\Uparrow^i(d/)$. Then we have $\mathcal{S}(a) = (\mathcal{S}(b)._p\circ)\langle\mathcal{S}(d)\rangle_n$ and $\mathcal{S}(a') = \mathcal{S}(b)\langle\mathcal{S}(d)\rangle_q._m\circ$ where $n = \mathit{maxred}_{\varsigma^f_{db}}(BES(a))$ and $m = \mathit{maxred}_{\varsigma^f_{db}}(BES(a'))$. Note that $m = n$. Now since $\langle\rangle_n \gg ._n$ we must verify that $\mathcal{S}(a) = (\mathcal{S}(b)._p\circ)\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \mathcal{S}(b)\langle\mathcal{S}(d)\rangle_q$ and that $\mathcal{S}(a) = (\mathcal{S}(b)._p\circ)\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \circ$. The second is valid trivially. As for the first we have,

$$\begin{array}{rcl} n & = & \mathit{maxred}_{\varsigma^f_{db}}(BES((b.l)[s])) \\ & = & \mathit{maxred}_{\varsigma^f_{db}}(BES(b[s].l)) \\ & = & \mathit{maxred}_{\varsigma^f_{db}}(BES(b[s]).l) \\ & \geq_{L\ 5.33(4)} & \mathit{maxred}_{\varsigma^f_{db}}(BES(b[s])) \\ & = & q \end{array}$$

    So we have two further cases to consider,

    1. $n > q$. Then since $\langle\rangle_n \gg \langle\rangle_q$ we simply verify that $(\mathcal{S}(b)._p\circ)\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \mathcal{S}(b)$ and that $(\mathcal{S}(b)._p\circ)\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \mathcal{S}(d)$.
    2. $n = q$. We verify that $\langle\mathcal{S}(b)._p\circ, \mathcal{S}(d)\rangle \succeq'_{\mathcal{T}_l} \langle\mathcal{S}(b), \mathcal{S}(d)\rangle$.

  - $*$ $s =\Uparrow^i(\uparrow)$. Then $\mathcal{S}(a) = \mathcal{S}(b)._n\circ$ and $\mathcal{S}(a') = \mathcal{S}(b)._m\circ$ where $n = \mathit{maxred}_{\varsigma^f_{db}}(BES(b.l))$ and $m = \mathit{maxred}_{\varsigma^f_{db}}(BES(b[s].l))$. Then we reason,

$$\begin{array}{rcl} m & = & \mathit{maxred}_{\varsigma^f_{db}}(BES(b[s].l)) \\ & = & \mathit{maxred}_{\varsigma^f_{db}}(BES(b[s]).l) \\ & =_{L\ 5.26} & \mathit{maxred}_{\varsigma^f_{db}}(\mathcal{U}_i^2(BES(b)).l) \\ & = & \mathit{maxred}_{\varsigma^f_{db}}(\mathcal{U}_i^2(BES(b.l))) \\ & =_{L\ 5.33(5)} & \mathit{maxred}_{\varsigma^f_{db}}(BES(b.l)) \\ & = & n \end{array}$$

    And we may verify that $\langle\mathcal{S}(b), \circ\rangle \succeq'_{\mathcal{T}_l} \langle\mathcal{S}(b), \circ\rangle$.

  - $*$ $s =\Uparrow^i(@l')$. Then $\mathcal{S}(a) = \mathcal{S}(b)._m \circ \llbracket\circ\rrbracket_n$ and $\mathcal{S}(a') = \mathcal{S}(b)\llbracket\circ\rrbracket_q._p\circ$. Then since $n = p$ and $\llbracket\rrbracket_n \gg ._n$ we proceed analogously to the case $s =\Uparrow^i(d/)$.

- SU. Then $a = b \lhd < m > [s]$ and $a' = b[s] \lhd < m[s] >$. Then depending on s we have,
  - $*$ $s =\Uparrow^i(d/)$. Then $\mathcal{S}(a) = (\lhd(\mathcal{S}(b), \mathcal{S}(m)))\langle\mathcal{S}(d)\rangle_n$ and $\mathcal{S}(a') = \lhd(\mathcal{S}(b)\langle\mathcal{S}(d)\rangle_p, \mathcal{S}(m)\langle\mathcal{S}(d)\rangle_q)$. Now since $\langle\rangle_n \gg \lhd$ we must verify that $\mathcal{S}(a) = (\lhd(\mathcal{S}(b), \mathcal{S}(m)))\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \mathcal{S}(b)\langle\mathcal{S}(d)\rangle_p$ and that $\mathcal{S}(a) = (\lhd(\mathcal{S}(b), \mathcal{S}(m)))\langle\mathcal{S}(d)\rangle_n \succeq_{\mathcal{T}_l} \mathcal{S}(m)\langle\mathcal{S}(d)\rangle_q$.
    Recall that $n = \mathit{maxred}_{\varsigma^f_{db}}(BES(b \lhd < m > [s])) = \mathit{maxred}_{\varsigma^f_{db}}(BES(b[s]) \lhd < BES(m[s]) >)$, $p = \mathit{maxred}_{\varsigma^f_{db}}(BES(b[s]))$ and $q = \mathit{maxred}_{\varsigma^f_{db}}(BES(m[s]))$. Then by Lemma 5.33(6) $n \geq p$ and $n \geq q$. We consider the first item (the second is similar). If $n > p$ then since $\mathcal{S}(b)$ and $\mathcal{S}(d)$ are subterms of $\mathcal{S}(a)$ we may conclude $\mathcal{S}(a) \succeq_{\mathcal{T}_l} \mathcal{S}(a')$. The case $n = p$ is similar.
  - $*$ $s =\Uparrow^i(\uparrow)$. Then $\mathcal{S}(a) = \lhd(\mathcal{S}(b), \mathcal{S}(m)) = \mathcal{S}(a')$.
  - $*$ $s =\Uparrow^i(@l)$. Then $\mathcal{S}(a) = (\lhd(\mathcal{S}(b), \mathcal{S}(m)))\llbracket\circ\rrbracket_n$ and $\mathcal{S}(a') = \lhd(\mathcal{S}(b)\llbracket\circ\rrbracket_p, \mathcal{S}(m)\llbracket\circ\rrbracket_q)$. Then we proceed analogously to the case $s =\Uparrow^i(d/)$.

- $FVar$. Then $a = 1[b/]$ and $a' = b$. Then $\mathcal{S}(a) = \star\langle\mathcal{S}(b)\rangle_n$ and $\mathcal{S}(a') = \mathcal{S}(b)$ where $n = \mathit{maxred}_{\varsigma^f_{db}}(BES(1[b/]))$. Since $\mathcal{S}(a')$ is a subterm of $\mathcal{S}(a)$ we are done.

- $RVar$. Then $a = m + 1[b/]$ and $a' = m$. Then $\mathcal{S}(a) = \star\langle\mathcal{S}(b)\rangle_n$ and $\mathcal{S}(a') = \star$ where $n = \mathit{maxred}_{\varsigma^f_{db}}(BES(m + 1[b/]))$. Since $\mathcal{S}(a')$ is a subterm of $\mathcal{S}(a)$ we are done.

- $FInv$. Then $a = 1[@l]$ and $a' = 1.l$. Then $\mathcal{S}(a) = \star\llbracket\circ\rrbracket_0$ and $\mathcal{S}(a') = \star._0\circ$. Since $\llbracket\rrbracket_0 \gg ._0$ we may verify that indeed $\mathcal{S}(a) \succeq_{\mathcal{T}_l} \mathcal{S}(a')$.

- $RInv$. Then $a = 1[\Uparrow(s)]$ and $a' = 1$. Then $\mathcal{S}(a') = \star$ and as for $\mathcal{S}(a)$ we must consider the following cases depending on s,
  - $*$ $s =\Uparrow^i(d/)$. Then $\mathcal{S}(a) = \star\langle\mathcal{S}(d)\rangle_0$ and we are done.

  - $s =\Uparrow^i(\uparrow)$. Then $\mathcal{S}(a) = \star$ and we are done.
  - $s =\Uparrow^i(@l)$. Then $\mathcal{S}(a) = \star\llbracket\circ\rrbracket_0$ and we are done.

- *RVarLift*. Then $a = p + 1[\Uparrow(s)]$ and $a' = p[s][\uparrow]$. We must consider the following cases depending on s,
  - $s =\Uparrow^i(d/)$. Then $\mathcal{S}(a) = \star\langle\mathcal{S}(d)\rangle_m$ and $\mathcal{S}(a') = \star\langle\mathcal{S}(d)\rangle_n$ where $n = \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(a))$ and $m = \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(p[s]))$. Then we reason as follows,

$$\begin{array}{rcl} m & = & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(a)) \\ & = & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(a')) \\ & =_{L\ 5.26} & \mathit{maxred}_{\varsigma^f_{db}}(\mathcal{U}^2_0(\mathit{BES}(p[s]))) \\ & =_{L\ 5.33(5)} & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(p[s])) \\ & = & n \end{array}$$

    Thus we use the 'equal heads' case and we are done.
  - $s =\Uparrow^i(\uparrow)$. Then $\mathcal{S}(a) = \star = \mathcal{S}(a')$ and we are done.
  - $s =\Uparrow^i(@l)$. Then $\mathcal{S}(a) = \star\llbracket\circ\rrbracket_0 = \mathcal{S}(a')$ and we are done.

- *VarShift*. Then $a = n[\uparrow]$ and $a' = n + 1$. Then $\mathcal{S}(a) = \star = \mathcal{S}(a')$.
- *CO*. Then $a = b[\Uparrow^i(@l_j)][\Uparrow^i([l_j := c, {m_i}^{i\in 1..n, i\neq j}]/)]$ and $a' = b[\Uparrow^i(c/)]$. Then we have $\mathcal{S}(a) = \mathcal{S}(b)\llbracket\circ\rrbracket_p\langle\mathcal{S}([l_j := c, {m_i}^{i\in 1..n, i\neq j}])\rangle_n$ and $\mathcal{S}(a') = \mathcal{S}(b)\langle\mathcal{S}(c)\rangle_q$. Then by Proposition 5.31 we have $\mathit{BES}(a) \twoheadrightarrow_{\varsigma^f_{db}} \mathit{BES}(a')$, thus by Lemma 5.39 $n \geq q$. As remarked before it suffices to consider the case $n = q$. We must verify that $\langle\mathcal{S}(b)\llbracket\circ\rrbracket_p, \mathcal{S}([l_j := c, {m_i}^{i\in 1..n, i\neq j}])\rangle \succeq'_{T_l} \langle\mathcal{S}(b), \mathcal{S}(c)\rangle$ which is valid.
- *SW*. Then $a = b[\Uparrow^i(@l)][\Uparrow^k(s)]$ and $a' = b[\Uparrow^k(s)][\Uparrow^i(@l)]$ with $k > i$. We must consider the following cases depending on s,
  - $s =\Uparrow^j(d/)$. Then $\mathcal{S}(a) = \mathcal{S}(b)\llbracket\circ\rrbracket_m\langle\mathcal{S}(d)\rangle_n$ and $\mathcal{S}(a') = \mathcal{S}(b)\langle\mathcal{S}(d)\rangle_q\llbracket\circ\rrbracket_p$. Since $n = p$ and $<>_n \gg \llbracket\rrbracket_n$ we must verify that $\mathcal{S}(a) = \mathcal{S}(b)\llbracket\circ\rrbracket_m\langle\mathcal{S}(d)\rangle_n \succeq_{T_l} \mathcal{S}(b)\langle\mathcal{S}(d)\rangle_q$ and that $\mathcal{S}(a) = \mathcal{S}(b)\llbracket\circ\rrbracket_m\langle\mathcal{S}(d)\rangle_n \succeq_{T_l} \circ$. The second item is straightforward. As for the first item we reason as follows,

$$\begin{array}{rcl} n & = & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(a)) \\ & = & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(a')) \\ & =_{L\ 5.29(2)} & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b[\Uparrow^k(s)]) \ll i+1 \leftarrow l \gg) \\ & =_{L\ 5.33(7)} & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b[\Uparrow^k(s)])) \\ & = & q \end{array}$$

    And since $\langle\mathcal{S}(b)\llbracket\circ\rrbracket_m, \mathcal{S}(d)\rangle \succeq'_{T_l} \langle\mathcal{S}(b), \mathcal{S}(d)\rangle$ we are done.
  - $s =\Uparrow^j(\uparrow)$. Then $\mathcal{S}(a) = \mathcal{S}(b)\llbracket\circ\rrbracket_n$ and $\mathcal{S}(a') = \mathcal{S}(b)\llbracket\circ\rrbracket_p$ where $n = \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b[\Uparrow^i(@l)]))$ and $p = \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(a'))$. Then we reason as follows,

$$\begin{array}{rcl} n & = & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b[\Uparrow^i(@l)])) \\ & =_{L\ 5.29(2)} & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b) \ll i+1 \leftarrow l \gg) \\ & =_{L\ 5.33(5)} & \mathit{maxred}_{\varsigma^f_{db}}(\mathcal{U}^2_{k+j}(\mathit{BES}(b) \ll i+1 \leftarrow l \gg)) \\ & =_{L\ 5.24(6)} & \mathit{maxred}_{\varsigma^f_{db}}(\mathcal{U}^2_{k+j}(\mathit{BES}(b)) \ll i+1 \leftarrow l \gg) \\ & =_{L\ 5.28} & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b[\Uparrow^{k+j}(\uparrow)]) \ll i+1 \leftarrow l \gg) \\ & =_{L\ 5.29(2)} & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(a')) \\ & = & p \end{array}$$

  - $s =\Uparrow^j(@l')$. Then $\mathcal{S}(a) = \mathcal{S}(b)\llbracket\circ\rrbracket_m\llbracket\circ\rrbracket_n$ and $\mathcal{S}(a') = \mathcal{S}(b)\llbracket\circ\rrbracket_p\llbracket\circ\rrbracket_q$. Since $n = q$ we must verify that $\langle\mathcal{S}(b)\llbracket\circ\rrbracket_m, \circ\rangle \succeq'_{T_l} \langle\mathcal{S}(b)\llbracket\circ\rrbracket_p, \circ\rangle$. Recall that $m = \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b[\Uparrow^i(@l)]))$ and $p = \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b[\Uparrow^{k+j}(@l')]))$. We show below that $m = p$,

$$\begin{array}{rcl} m & = & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b[\Uparrow^i(@l)])) \\ & =_{L\ 5.29(2)} & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b) \ll i+1 \leftarrow l \gg) \\ & =_{L\ 5.33(7)} & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b)) \\ & =_{L\ 5.33(7)} & \mathit{maxred}_{\varsigma^f_{db}}(\mathit{BES}(b) \ll k+j+1 \leftarrow l' \gg) \\ & = & p \end{array}$$

The remaining cases ($a = (l \doteq g)$, $a = (l := c)$ and $a = \varsigma(c)$) are handled by making use of the induction hypothesis.

∎

Remark 5.45 Lemma 5.43 still holds if the rules *RVar*, *FVar*, *FInv*, *CO* are added to R.

We may now prove the main proposition of this subsection, namely, the proposition of preservation of strong normalization for the ς_{dbes}-calculus.

Proposition 5.46 (PSN of the ς_{dbes}-calculus) The ς_{dbes}-calculus preserves strong normalization.

Proof. Suppose that the ς_{dbes}-calculus does not preserve strong normalization. Thus there is a pure term a which is strongly ς^{f}_{db}-normalizing but which possesses an infinite derivation in the ς_{dbes}-calculus. Since the rewrite system $S = ESDB \cup \{MO, FO, FI\}$ is strongly normalizing this derivation must have the form

$$a = a_1 \twoheadrightarrow_S a_2 \rightarrow_{MI} a_3 \twoheadrightarrow_S a_4 \rightarrow_{MI} a_5 \dots$$

where the reductions $a_{2k} \rightarrow_{MI} a_{2k+1}$ for $k \geq 1$ occur infinitely many times. Now since a is in $\mathcal{F}$, and since by Lemma 5.38 the set $\mathcal{F}$ is closed under reduction in ς^{f}_{db} we obtain an infinite sequence

$$\mathcal{S}(a) = \mathcal{S}(a_1) \succeq_{\mathcal{T}_l} \mathcal{S}(a_2) \succ_{\mathcal{T}_l} \mathcal{S}(a_3) \succeq_{\mathcal{T}_l} \mathcal{S}(a_4) \succ_{\mathcal{T}_l} \mathcal{S}(a_5) \dots$$

This contradicts well-foundedness of the recursive path ordering $\succ_{\mathcal{T}_l}$.

Part III

From Higher-Order to First-Order Rewriting

Chapter 6

A de Bruijn Notation for Higher-Order Rewriting

Higher-order (term) rewriting concerns the transformation of terms in the presence of binding mechanisms for variables and substitution, its theory may be seen to start with the pioneering work of J.W.Klop in his 1980 PhD thesis [Klo80]. The paradigmatic example of a higher-order rewrite system is the λ-calculus:

$$(\lambda x.M)N \rightarrow_{\beta} M\{x \leftarrow N\}$$

The right-hand side of this rule makes use of *substitution*: $M\{x \leftarrow N\}$ denotes the term which results from substituting N for all free occurrences of x in M. Substitution is a metalevel notion (it lives in the world of our language of discourse) that may be seen as a consequence of the existence of special symbols called *binder symbols* that have the power to bind variables in terms. This entails that substitution may not be confined to usual first-order replacement, but rather has to be careful to respect the status (free or bound) of variables when doing its work. In this sense, it is fair to say that substitution is 'respectful replacement'. However, it is a mistake to dismiss substitution as a trivial concept: the theory of higher-order rewriting is considerably more involved than that of first-order rewriting.

Many higher-order rewrite systems (HORS) exist and work in the area is currently very active. In the seminal work of J.W.Klop [Klo80] *Combinatory Reduction Systems* (*CRS*) were introduced. Several formalisms introduced later, of which we mention some, are: Z.Khasidashvili's *Expression Reduction Systems* (*ERS*) of which an early reference is [Kha90], T.Nipkow introduces *Higher-Order Rewrite Systems* (*HRS*) in [Nip91], D.A.Wolfram defines *Higher-Order Term Rewrite Systems* [Wol93], V. van Oostrom and F. van Raamsdonk introduce *Higher-Order term Rewriting Systems* [OR94] as a general higher-order rewriting formalism encompassing many known formalisms [Oos94, Raa96] and B.Pagano defines *Explicit Reduction Systems* (*XRS*) [Pag98] using de Bruijn indices notation. F. van Raamsdonk's PhD thesis provides a survey [Raa96].

This chapter aims at getting rid of α-conversion in the substitution process. Although from the metalevel the execution of a substitution is atomic, the cost of computing it highly depends on the form of the terms, especially if unwanted variable capture conflicts must be avoided by renaming bound variables. So this aim has a practical interest since any implementation of higher-order rewriting must include instructions for computing this notion of substitution. As illustrated in Section 2.2.2, there is a standard technique introduced by de Bruijn to get rid of α-conversion. De Bruijn indices take care of renaming because the representation of variables by indices completely eliminates the capture of variables. However, de Bruijn formalisms have only been studied for particular systems (and only on the term level) and no general framework of higher-order rewriting with indices has been proposed. We address this problem here by focusing not only on de Bruijn terms (as usually done in the literature for λ-calculus [KR98]) but also on de Bruijn metaterms, which are the syntactical objects used to express any general higher-order rewrite system formulated in a de Bruijn context. More precisely, we shall introduce a de Bruijn notation for Expression Reduction Systems, obtaining $SERS_{db}$. In fact, we shall formulate a slightly simplified version of *ERS* that we shall call Simplified *ERS* (*SERS*), better suited for our purposes, and then consider a de Bruijn notation for this formalism. The reason for choosing the *ERS* formalism is that its syntax is close to the 'usual' presentation of the λ-calculus. For example, the β-rewrite rule is written $app((\lambda x.M), N) \rightarrow M[x \leftarrow N]$ where M and N can be instantiated by any terms.

The *SERS* formalism may be viewed as an *interface* of a programming language based on higher-order rewriting. Since the use of variable name based formalisms are necessary for humans to interact with computers in a user-friendly way, technical resources like de Bruijn indices (and, later on, explicit substitutions) should live behind the scene, in other words, should be implementation concerns. Moreover, it is required of whatever is behind the scene to be as faithful as possible as regards the formalism it is implementing. So a key issue shall be the detailed study of the relationship between *SERS* and $SERS_{db}$. The definitions developed in this chapter give formal translations from higher-order syntax with names to higher-order syntax with indices and vice-versa. These translators are extensions to the higher-order setting of the translations presented in [Cur93], also studied in [KR98].

As regards existing higher-order rewrite formalisms based on de Bruijn index notation and/or explicit substitutions to the best of the author's knowledge there are three: *Explicit CRS* [BR96], *Explicit Reduction Systems* (*XRS*) [Pag98], and the *Calculus of Indexed Names and Named Indices* (*CINNI*) [Ste00]. In [BR96] explicit substitutions à la λx [Ros92, Blo97] are added to the *CRS* formalism as a first step towards using higher-order rewriting with explicit substitutions for modeling the evaluation of functional programs in a faithful way. Since this is done in a variable name setting α-conversion must be dealt with as in *CRS*. Pagano's *XRS* constitutes the first HORS which fuses de Bruijn indices notation and explicit substitutions. It is presented as a generalization of the $\lambda\sigma_{\Uparrow}$-calculus [CHL96] but no connection has been established between *XRS* and well-known systems such as *CRS*, *ERS* and *HRS*. Indeed, it is not clear at all how some seemingly natural rules expressible, say, in the *ERS* formalism, may be written in an *XRS*. As an example, consider a rewrite system for logical expressions such that if $imply(e_1, e_2)$ reduces to the constant *true* then e_1 logically implies e_2 in classical first-order predicate logic. A possible rewrite rule could be:

$$imply(\exists x \forall y M, \forall y \exists x M) \rightarrow_{imp} true$$

A naïve attempt might consider the rewrite rule:

$$imply(\exists \forall M, \forall \exists M) \rightarrow_{imp_{db}} true$$

as a possible representation of this rule in the *XRS* formalism, but it does not have the desired effect. Indeed, for example the term $imply(\exists x \forall y.x, \forall y \exists x.x)$ is an instance of the *imp*-rule, whereas its naïve de Bruijn representation $imply(\exists \forall 2, \forall \exists 1)$ is not an instance of imp_{db}. Note that regardless of the fact that *XRS* incorporate explicit substitutions, this problem arises already at the level of de Bruijn notation. Another example of interest is the extensional rule for functional types η:

$$\lambda x.(app(M, x)) \rightarrow M \text{ if } x \text{ is not free in } M$$

which is usually expressed in a de Bruijn based system with explicit substitutions as η_{db}

$$\lambda(app(M, 1)) \rightarrow N \text{ if } M =_{\mathcal{E}} N[\uparrow]$$

where $M =_{\mathcal{E}} N$ means that M and N are equivalent modulo the theory of explicit substitutions $\mathcal{E}$ (for example $\mathcal{E}$ might be υ). Neither the *imp*-rule nor η_{db} is possible in the *XRS* formalism so that they do not, in principle, have the same expressive power as *ERS*. Recently, the author has learned of an alternative representation for terms introduced by K.J.Berkling (see [Ste00] for references). This notation is used by M-O.Stehr [Ste00] to eliminate α-conversion from higher-order rewrite systems. As in B.Pagano's *XRS*, no relation to established HORS in the literature is presented. In fact, the definition of the higher-order rewriting setting is not provided. We shall show that $SERS_{db}$ allows rules such as those previously mentioned to be faithfully represented, and at the same time shall establish precise links with *ERS*.

Structure of the chapter

We begin by introducing our work and study scenario, the *SERS* formalism. After defining notions such as pre-metaterms, metaterms and terms and their corresponding notions of substitution, we consider rewrite rules. Valuations are then introduced in order to put rewrite rules to work. Metaterms are used to specify rewrite rules, and rewrite rules are used to rewrite terms. The de Bruijn based formalism $SERS_{db}$ is defined in Section 6.2, and analogous concepts are considered in that setting. Next we undertake the task of comparing these two formalisms: Section 6.3 studies an encoding of *SERS* in $SERS_{db}$ and Section 6.4 considers the opposite encoding. In each case, this requires that we deal with a static phase by showing how terms and rewrite rules may be encoded, and a rewrite-preservation phase or dynamical phase in which we must show that valuations, and hence the induced rewrite relation, may also be encoded appropriately. The $SERS_{db}$-to-*SERS* direction shall prove to be technically more demanding than the other. The reason is that we have a choice for selecting appropriate names for variables *and* metavariables, and we must rest assured that the results are not biased by our selection. Also, the valuations obtained by this process of translation must yield 'good' valuations in the sense that they are permitted to be used in order to use the rewrite rules for rewriting terms. Finally we consider preservation of confluence:

- if $\mathcal{R}$ is a confluent *SERS* then its translation to the de Bruijn indices setting, $U(\mathcal{R})$, is a confluent $SERS_{db}$.
- if $\mathcal{R}$ is a confluent $SERS_{db}$ then is translation to the named setting, $T(\mathcal{R})$, is a confluent *SERS*.

6.1 Simplified Expression Reduction Systems

We introduce the name based higher-order rewrite formalism *SERS*. The latter is an appropriate simplification of Khasidashvili's *ERS* [Kha90] which consists in restricting binders to those which bind one variable and restricting substitution to simple substitution (in contrast to simultaneous or parallel substitution).

Definition 6.1 (Signature) A *SERS*-signature Σ consists of the denumerable (and possibly infinite) disjoint sets:

- $\Sigma_v = \{x_1, x_2, x_3, \ldots\}$ a set of *variables*, arbitrary variables are denoted $x, y, \ldots$
- $\Sigma_{bmv} = \{\alpha_1, \alpha_2, \alpha_3, \ldots\}$ a set of *pre-bound o-metavariables* (o for object), denoted $\alpha, \beta, \ldots$
- $\Sigma_{fmv} = \{\widehat{\alpha_1}, \widehat{\alpha_2}, \widehat{\alpha_3}, \ldots\}$ a set of *pre-free o-metavariables*, denoted $\widehat{\alpha}, \widehat{\beta}, \ldots$
- $\Sigma_{tmv} = \{X_1, X_2, X_3, \ldots\}$ a set of *t-metavariables* (t for term), denoted $X, Y, Z, \ldots$
- $\Sigma_f = \{f_1, f_2, f_3, \ldots\}$ a set of *function symbols* equipped with a fixed (possibly zero) arity, denoted $f, g, h, \ldots$
- $\Sigma_b = \{\lambda_1, \lambda_2, \lambda_3, \ldots\}$ a set of *binder symbols* equipped with a fixed (non-zero) arity, denoted $\lambda, \mu, \nu, \xi, \ldots$

The union of Σ_{bmv} and Σ_{fmv} is the set of o-metavariables of the signature. When speaking of metavariables without further qualifiers we refer to o- and t-metavariables. Since all these alphabets are ordered, given any symbol s we shall denote $\mathcal{O}(s)$ its position in the corresponding alphabet.

Definition 6.2 (Labels) A *label* is a finite sequence of symbols of an alphabet. We shall use $k, l, l_i, \ldots$ to denote arbitrary labels and ϵ for the empty label. If s is a symbol and l is a label then the notation $s \in l$ means that the symbol s *appears in* the label l, and also, we use sl to denote the new label whose head is s and whose tail is l. Other notations are $|l|$ for the *length* of l (number of symbols in l) and $\mathtt{at}(l, n)$ for the n-th element of l assuming $n \leq |l|$. Also, if s occurs (at least once) in l then $\mathtt{pos}(s, l)$ denotes the *position of the first occurrence* of s in l. If θ is a function defined on the alphabet of a label $l = s_1 \ldots s_n$, then $\theta(l)$ denotes the label $\theta(s_1) \ldots \theta(s_n)$. In the sequel, we may use a label as a set (e.g. if S is a set then $S \cap l$ denotes the intersection of S with the underlying set determined by l) if no confusion arises. A *simple label* is a label without repeated symbols.

Definition 6.3 (Pre-metaterms) The set of *SERS pre-metaterms* over Σ, denoted $\mathcal{PMT}$, is defined by:

$$M \quad ::= \quad \alpha \mid \widehat{\alpha} \mid X \mid f(M, \ldots, M) \mid \xi\alpha.(M, \ldots, M) \mid M[\alpha \leftarrow M]$$

Arities are supposed to be respected, i.e. a pre-metaterm like $f(M_1, \ldots, M_n)$ (resp. $\xi\alpha.(M_1, \ldots, M_n)$) is generated by the grammar only if f (resp. ξ) has arity $n \geq 0$ (resp. $n > 0$).

We shall use $M, N, M_i, \ldots$ to denote pre-metaterms. The symbol $\bullet[\bullet \leftarrow \bullet]$ in the pre-metaterm $M_1[\alpha \leftarrow M_2]$ is called *metasubstitution operator*. The o-metavariable α in a pre-metaterm of the form $\xi\alpha.(M_1, \ldots, M_n)$ or $M_1[\alpha \leftarrow M_2]$ is referred to as the *formal parameter*. The set of binder symbols together with the metasubstitution operator are called *binder operators*, thus the metasubstitution operator is a binder operator (since it has binding power) but is *not* a binder symbol since it is not an element of Σ_b.

The main difference between *SERS* and *ERS* is that in the latter binders and metasubstitutions are defined on *multiple* o-metavariables. Indeed, pre-metaterms like $\xi\alpha_1 \ldots \alpha_k.(M_1, \ldots, M_m)$ and $M[\alpha_1 \ldots \alpha_k \leftarrow M_1, \ldots, M_k]$ are possible in *ERS*, with the underlying hypothesis that $\alpha_1 \ldots \alpha_k$ are all distinct and with the underlying semantics that $M[\alpha_1 \ldots \alpha_k \leftarrow M_1, \ldots, M_k]$ denotes usual (parallel) substitution. It is well known that multiple substitution can be encoded by simple substitution. Indeed, $M[\alpha_1 \ldots \alpha_k \leftarrow M_1, \ldots, M_k]$ can be encoded as the pre-metaterm $M[\alpha_1 \leftarrow \beta_1][\alpha_2 \leftarrow \beta_2] \ldots [\alpha_k \leftarrow \beta_k][\beta_1 \leftarrow M_1][\beta_2 \leftarrow M_2] \ldots [\beta_k \leftarrow M_k]$, where $\beta_1, \ldots, \beta_k$ are *fresh* pre-bound o-metavariables. As for $\xi\alpha_1 \ldots \alpha_k.(M_1, \ldots, M_m)$ it may be encoded with the help of two binder symbols ξ and ξ' as the pre-metaterm $\xi\alpha_1.(\xi\alpha_2.(\ldots \xi'\alpha_k.(M_1, \ldots, M_m)))$. There is also a notion of scope indicator in *ERS*, used to express in which arguments of the quantifier variables are bound. Scope indicators shall not be considered in *SERS* since they do not seem to contribute to the expressive power of *ERS*.

A pre-metaterm M has an associated *tree*, denoted *tree*(M), in the following way:

- the tree of a metavariable α, $\widehat{\alpha}$ or X is the tree with the single node α, $\widehat{\alpha}$ and X, respectively.
- if $T_1, \ldots, T_n$ are the trees of $M_1, \ldots, M_n$, then the tree of $f(M_1, \ldots, M_n)$ is that of Figure 6.1(a).
- if $T_1, \ldots, T_n$ are the trees of $M_1, \ldots, M_n$, then the tree of $\xi\alpha.(M_1, \ldots, M_n)$ is that of Figure 6.1(b).
- if T_1, T_2 are the trees of M_1, M_2, then the tree of $M_1[\alpha \leftarrow M_2]$ is that of Figure 6.1(c).

The tree of $f(M_1, \ldots, M_n)$ has the expected form, however the tree of $M_1[\alpha \leftarrow M_2]$ may seem somewhat odd since there are two nodes above the tree of M_1. The reason is that the metasubstitution operator is asymmetric in that its left argument M_1 is considered to be under a binding effect whereas M_2 is not. We would like this to be reflected in the structure of the tree, enabling us to look "above" a position (Figure 6.1(c)) in a tree to know under which binders it occurs.

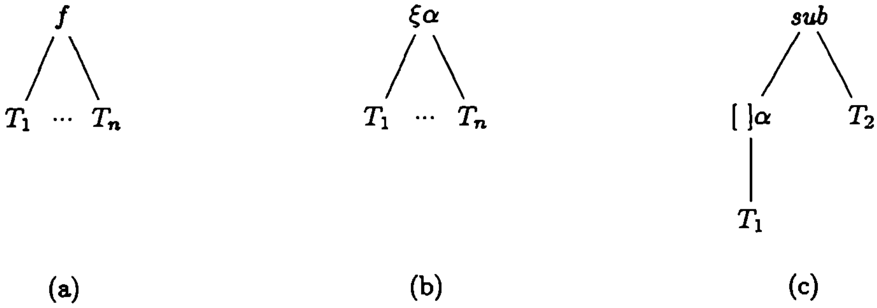


Figure 6.1: Pre-Metaterms as Trees

A position is a label over the alphabet $\mathbb{N}$. We use ϵ to denote the empty word in $\mathbb{N}^*$. Given a pre-metaterm N appearing in M, the set of *occurrences* of N in M is the set of positions of *tree*(M) where *tree*(N) occurs (positions in trees are defined as usual). The *parameter path* of a position p in *tree*(M) is the list containing all the (pre-bound) o-metavariables occuring in the path from p to the root of *tree*(M). Likewise, we may define the parameter path of an occurrence of N in M.

Example 6.4 Consider the pre-metaterm $M = f((\xi\alpha.(X)), Y)$. Then X occurs at position 1.1 and Y at position 2. The parameter path of 1.1 (or just X) is α and the parameter path of 2 (or just Y) is ϵ. Consider the pre-metaterm $M = \mu\beta.(X[\alpha \leftarrow \lambda\gamma.(g(\beta, g(\gamma, Z)))])$. Then the submetaterm $\lambda\gamma.(g(\beta, g(\gamma, Z)))$ occurs at position 1.2 and $g(\gamma, Z)$ occurs at position 1.2.1.2; the parameter path of 1.2 (or just $\lambda\gamma.(g(\beta, g(\gamma, Z))))$ is β, the parameter path of 1.2.1.2 is $\gamma\beta$.

The following definition introduces the set of *metaterms*, which are pre-metaterms that are *well-formed* in the sense that they prevent the use of the same name for two different occurrences of a formal parameter appearing in the parameter path of a given pre-metaterm, i.e. all the formal parameters appearing in the same path of a pre-metaterm must be different. Also, it guarantees that metavariables in Σ_{bmv} only occur bound.

Definition 6.5 (Metaterms) A pre-metaterm $M \in \mathcal{PMT}$ over Σ is said to be a *metaterm* over Σ iff the predicate $\mathcal{WF}(M)$ holds, where $\mathcal{WF}(M)$ iff $\mathcal{WF}_{\epsilon}(M)$ holds, and $\mathcal{WF}_{l}(M)$ is defined by induction on the structure of the pre-metaterm M for any label l as follows:

- $\mathcal{WF}_l(\alpha)$ iff $\alpha \in l$
- $\mathcal{WF}_l(\widehat{\alpha})$ and $\mathcal{WF}_l(X)$ hold iff l is a simple label
- $\mathcal{WF}_l(f(M_1, \ldots, M_n))$ iff for all $1 \leq i \leq n$ we have $\mathcal{WF}_l(M_i)$
- $\mathcal{WF}_l(\xi\alpha.(M_1, \ldots, M_n))$ iff $\alpha \notin l$ and for all $1 \leq i \leq n$ we have $\mathcal{WF}_{\alpha l}(M_i)$
- $\mathcal{WF}_l(M_1[\alpha \leftarrow M_2])$ iff $\alpha \notin l$ and $\mathcal{WF}_l(M_2)$ and $\mathcal{WF}_{\alpha l}(M_1)$.

Example 6.6 The pre-metaterms $f(\xi\alpha.(X), \lambda\alpha.(Y))$, $f(\widehat{\beta}, \lambda\alpha.(Y))$ and $g(\lambda\alpha.(\xi\beta.(h)))$ are metaterms, while the pre-metaterms $f(\alpha, \xi\alpha.(X))$ and $f(\widehat{\beta}, \lambda\alpha.(\xi\alpha.(X)))$ are not.

In the sequel, pre-bound (resp. pre-free) o-metavariables occurring in metaterms shall simply be referred to as bound (resp. free) o-metavariables. Also, we shall assume, whenever possible, some fixed signature Σ and hence speak of pre-metaterms or metaterms instead of pre-metaterms over Σ or metaterms over Σ. As we shall see, metaterms are used to specify rewrite rules.

Definition 6.7 (Free metavariables of pre-metaterms) Let M be a pre-metaterm, then $FMVar(M)$ denotes the *set of free metavariables* of M, which is defined as follows:

$$\begin{array}{lll}
FMVar(X) \stackrel{\text{def}}{=} \{X\} & FMVar(\alpha) \stackrel{\text{def}}{=} \{\alpha\} & FMVar(\widehat{\alpha}) \stackrel{\text{def}}{=} \{\widehat{\alpha}\} \\
FMVar(f(M_1, \ldots, M_n)) & \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} FMVar(M_i) & \\
FMVar(\xi\alpha.(M_1, \ldots, M_n)) & \stackrel{\text{def}}{=} (\bigcup_{i=1}^{n} FMVar(M_i)) \setminus \{\alpha\} & \\
FMVar(M_1[\alpha \leftarrow M_2]) & \stackrel{\text{def}}{=} (FMVar(M_1) \setminus \{\alpha\}) \cup FMVar(M_2) &
\end{array}$$

All metavariables in a pre-metaterm M which are not free are bound. We use $BMVar(M)$ to denote the bound metavariables of a pre-metaterm M. Note that only o-metavariables may occur bound in a metaterm, metavariables of the form $\widehat{\alpha}$ or X_i shall always occur free (if they occur at all) in a metaterm. We denote the set of *all* the metavariables of a metaterm or a pre-metaterm M by $MVar(M)$. So we have $MVar(M) = FMVar(M) \cup BMVar(M)$.

Let M be the metaterm $f(\widehat{\beta}, \lambda\alpha.Y)$. Then $FMVar(M) = \{\widehat{\beta}, Y\}$, $MVar(M) = \{\widehat{\beta}, Y, \alpha\}$, and $BMVar(M) = \{\alpha\}$. If M is the metaterm $f(\widehat{\beta}, \lambda\alpha.\alpha)$ then $FMVar(M) = \{\widehat{\beta}\}$, $BMVar(M) = \{\alpha\}$ and $MVar(M) = \{\alpha, \widehat{\beta}\}$.

Definition 6.8 (Terms and contexts) The set of *SERS terms* over Σ, denoted $\mathcal{T}$, and *contexts* are defined by:

$$\begin{array}{lcl}
\text{Terms} & t & ::= \; x \mid f(t, \ldots, t) \mid \xi x.(t, \ldots, t) \\
\text{Contexts} & C & ::= \; \Box \mid f(t, \ldots, C, \ldots, t) \mid \xi x.(t, \ldots, C, \ldots, t)
\end{array}$$

where $\Box$ denotes a 'hole'. We shall use $s, t, t_i, \ldots$ for terms and C, D for contexts. Contexts are just terms with exactly one occurrence of a hole. The x in ξx is called a *binding variable*. We remark that in contrast to other formalisms dealing with higher-order rewriting such as *CRS*, the set of terms is not contained in the set of pre-metaterms since the set of variables and the set of o-metavariables are disjoint. Terms shall be obtained from metaterms by suitable instantiation of o-metavariables and t-metavariables.

With $C[t]$ we denote the term obtained by replacing t for the hole $\Box$ in the context C. Note that this operation may introduce variable capture. We define the *label of a context* as a sequence of variables as follows:

$$\begin{array}{ll}
\mathtt{label}(\Box) & \stackrel{\text{def}}{=} \epsilon \\
\mathtt{label}(f(t_1, \ldots, C, \ldots, t_n)) & \stackrel{\text{def}}{=} \mathtt{label}(C) \\
\mathtt{label}(\xi x.(t_1, \ldots, C, \ldots, t_n)) & \stackrel{\text{def}}{=} \mathtt{label}(C)x
\end{array}$$

For example, the label of the context $f(\lambda x.(z, \xi y.(h(y, \Box))))$ is the sequence yx. The label of a context is a notion analogous to that of a parameter path of an occurrence, but defined for terms instead of pre-metaterms and where the only occurrence considered is that of the hole. The reason for not using the name 'parameter path' is just because the latter notion is defined on pre-metaterms, and contexts are not pre-metaterms.

Definition 6.9 (Free variables of terms) Let $t \in \mathcal{T}$, the set $FV(t)$ of *free variables* in t is defined by:

$$\begin{array}{ll} FV(x) & \stackrel{\text{def}}{=} \{x\} \\ FV(f(t_1, \ldots, t_n)) & \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} FV(t_i) \\ FV(\xi x.(t_1, \ldots, t_n)) & \stackrel{\text{def}}{=} (\bigcup_{i=1}^{n} FV(t_i)) \setminus \{x\} \end{array}$$

This definition can be extended to contexts by adding the clause $FV(\Box) \stackrel{\text{def}}{=} \emptyset$.

Substitution on terms can be defined as follows:

Definition 6.10 ((Restricted) substitution of terms) The *(restricted) substitution* of a term t for a variable x in a term s, denoted $s\{x \leftarrow t\}$, is defined:

$$\begin{array}{lll} x\{x \leftarrow t\} & \stackrel{\text{def}}{=} t & \\ y\{x \leftarrow t\} & \stackrel{\text{def}}{=} y & \text{if } x \neq y \\ f(s_1, \ldots, s_n)\{x \leftarrow t\} & \stackrel{\text{def}}{=} f(s_1\{x \leftarrow t\}, \ldots, s_n\{x \leftarrow t\}) & \\ \xi x.(s_1, \ldots, s_n)\{x \leftarrow t\} & \stackrel{\text{def}}{=} \xi x.(s_1, \ldots, s_n) & \\ \xi y.(s_1, \ldots, s_n)\{x \leftarrow t\} & \stackrel{\text{def}}{=} \xi y.(s_1\{x \leftarrow t\}, \ldots, s_n\{x \leftarrow t\}) & \\ & \quad \text{if } x \neq y, \text{ and } (y \notin FV(t) \text{ or } x \notin FV(s)) & \end{array}$$

Thus $\bullet\{\bullet \leftarrow \bullet\}$ denotes the substitution operator on terms, but it may *not* apply α-conversion (renaming of bound variables) in order to avoid variable captures. Therefore this notion of substitution is not defined for all terms (hence its name). For example $(\xi y.x)\{x \leftarrow y\}$ is not defined. When defining the notion of rewrite relation on terms induced by rewrite rules we shall take α-conversion into consideration in order to guarantee that any substitution to be performed may be completed with restricted substitution. This shall allow us to 'localize' α-conversion when applying rewrite rules.

The fourth clause of Def. 6.10 could be avoided. However this complicates the definition of α-conversion, and also of v-equivalence (Def. 6.11) if the notion of restricted substitution for pre-metaterms is modified accordingly. So we shall stick to Def. 6.10 as it stands.

We may define α-conversion on terms as the smallest reflexive, symmetric and transitive relation closed by contexts verifying the following equality:

$$(\alpha) \quad \xi x.(s_1, \ldots, s_n) \; =_\alpha \; \xi y.(s_1\{x \leftarrow y\}, \ldots, s_n\{x \leftarrow y\}) \qquad y \text{ does not occur in } s_1, \ldots, s_n$$

Note that since y does not occur in $s_1, \ldots, s_n$ substitution is defined. We shall use $t =_\alpha s$ to denote that the terms t and s are α-convertible. This conversion is sound in the sense that $t =_\alpha s$ implies $FV(t) = FV(s)$. In fact, the latter identity holds at the occurrence level: if p is an occurrence of a free variable x in t then we find x at position p in s too (and vice-versa).

The notion of α-conversion for terms has a corresponding one for pre-metaterms which we call *v-equivalence* (v for variant). The intuitive meaning of two v-equivalent pre-metaterms is that they are able to receive the *same* set of potential 'valuations' (Def. 6.19). Thus for example, as one would expect, $\lambda\alpha.X \neq_v \lambda\beta.X$ because when α and X are replaced by x, and β is replaced by y, one obtains $\lambda x.x$ and $\lambda y.x$, which are not α-convertible. However, since pre-metaterms contain t-metavariables, the notion of v-equivalence is not straightforward as the notion of α-conversion in the case of terms. More on the intuitive idea of v-equivalence shall be said below.

Definition 6.11 (v-equivalence for pre-metaterms) Given pre-metaterms M and N, we say that M is *v-equivalent* to N, iff $M =_v N$ where $=_v$ is the smallest reflexive, symmetric and transitive relation closed by metacontexts[1] verifying:

$$\begin{array}{llll} (v1) & \xi\alpha.(P_1, \ldots, P_n) & =_v & \xi\beta.(P_1 \ll\alpha \leftarrow \beta\gg \ldots P_n \ll\alpha \leftarrow \beta\gg) \\ (v2) & P_1[\alpha \leftarrow P_0] & =_v & P_1 \ll\alpha \leftarrow \beta\gg [\beta \leftarrow P_0] \end{array}$$

[1] Metacontexts are defined analogously to contexts. The notion of 'label of a context' is extended to metacontexts as expected.

where P_i does not contain t-metavariables for $1 \leq i \leq n$, and

(v1) β is a pre-bound o-metavariable which does not occur in $P_1, \ldots, P_n$, and
(v2) β is a pre-bound o-metavariable which does not occur in P_1

$P \ll \alpha \leftarrow Q \gg$ is the restricted substitution for pre-metaterms:

$$\begin{array}{lll}
\alpha \ll \alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} Q & \\
\alpha' \ll \alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} \alpha' & \alpha \neq \alpha' \\
\widehat{\alpha'} \ll \alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} \widehat{\alpha'} & \\
X \ll \alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} X & \\
f(M_1, \ldots, M_n) \ll \alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} f(M_1 \ll \alpha \leftarrow Q \gg, \ldots, M_n \ll \alpha \leftarrow Q \gg) & \\
(\xi\alpha.(M_1, \ldots, M_n)) \ll \alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} \xi\alpha.(M_1, \ldots, M_n) & \\
(\xi\alpha'.(M_1, \ldots, M_n)) \ll \alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} \xi\alpha'.(M_1 \ll \alpha \leftarrow Q \gg, \ldots, M_n \ll \alpha \leftarrow Q \gg) & \\
 & \quad \text{if } \alpha \neq \alpha' \text{ and } (\alpha' \notin \mathit{FMVar}(Q) \text{ or } \alpha \notin \mathit{FMVar}(M_i)) & \\
(M_1[\alpha \leftarrow M_2]) \ll \alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} M_1[\alpha \leftarrow M_2 \ll \alpha \leftarrow Q \gg] & \\
(M_1[\alpha' \leftarrow M_2]) \ll \alpha \leftarrow Q \gg & \stackrel{\text{def}}{=} (M_1 \ll \alpha \leftarrow Q \gg)[\alpha' \leftarrow M_2 \ll \alpha \leftarrow Q \gg] & \\
 & \quad \text{if } \alpha \neq \alpha' \text{ and } (\alpha' \notin \mathit{FMVar}(Q) \text{ or } \alpha \notin \mathit{FMVar}(M_1)) &
\end{array}$$

Example 6.12 $\lambda\alpha.\alpha =_v \lambda\beta.\beta$, $\lambda\alpha.f =_v \lambda\beta.f$, but $\lambda\alpha.X \neq_v \lambda\beta.X$, $\lambda\beta.\lambda\alpha.X \neq_v \lambda\alpha.\lambda\beta.X$.

Note that pre-metaterms may be seen as contexts where the holes of a context are represented by t-metavariables. However, metaterms are not treated as first class citizens as in [BdV99].

We shall now consider the rewrite rules of a *SERS*. The rewrite rules are specified by using metaterms, whereas the rewrite relation is defined on terms.

Definition 6.13 (*SERS*-**rewrite rule**) A *SERS-rewrite rule* over Σ is a pair of metaterms $(G, D)^2$ over Σ (also written $G \rightarrow D$) such that:

- the first symbol (called head symbol) in G is a function symbol or a binder symbol,
- $\mathit{FMVar}(D) \subseteq \mathit{FMVar}(G)$, and
- G contains no occurrence of the metasubstitution operator.

Definition 6.14 (*SERS*) A *SERS* is a pair $(\Sigma, \mathcal{R})$ where Σ is a *SERS*-signature and $\mathcal{R}$ is a set of *SERS*-rewrite rules over Σ.

We shall often omit Σ and write $\mathcal{R}$ instead of $(\Sigma, \mathcal{R})$, if no confusion arises.

Example 6.15 The λ-calculus is defined by considering the signature containing the function symbols $\Sigma_f = \{app\}$ and binder symbols $\Sigma_b = \{\lambda\}$, together with the *SERS*-rewrite rule: $app(\lambda\alpha.X, Z) \rightarrow_\beta X[\alpha \leftarrow Z]$. The $\lambda\eta$-calculus is obtained by adding the following *SERS*-rewrite rule: $\lambda\alpha.(app(X, \alpha)) \rightarrow_\eta X$.

Example 6.16 The λx-calculus [BR96, Ros92] is defined by considering the signature containing the function symbols $\Sigma_f = \{app, subs\}$ and binder symbols $\Sigma_b = \{\lambda, \sigma\}$, together with the following *SERS*-rewrite rules:

$$\begin{array}{lll}
app(\lambda\alpha.X, Z) & \rightarrow_{Beta} & subs(\sigma\alpha.X, Z) \\
subs(\sigma\alpha.(app(X, Y)), Z) & \rightarrow_{App} & app(subs(\sigma\alpha.X, Z), subs(\sigma\alpha.Y, Z)) \\
subs(\sigma\alpha.\lambda\beta.(X), Z) & \rightarrow_{Lam} & \lambda\beta.(subs(\sigma\alpha.X, Z)) \\
subs(\sigma\alpha.\alpha, Z) & \rightarrow_{Var} & Z \\
subs(\sigma\alpha.\widehat{\beta}, Z) & \rightarrow_{rGc} & \widehat{\beta}
\end{array}$$

[2] We shall reserve letters L and R for the de Bruijn formalism $SERS_{db}$.

Example 6.17 The $\lambda\Delta$-calculus [RS94] is defined by considering the signature containing the function symbols $\Sigma_f = \{app\}$ and binder symbols $\Sigma_b = \{\lambda, \Delta\}$, together with the following *SERS*-rewrite rules:

$$\begin{array}{lll}
app(\lambda\alpha.X, Z) & \rightarrow_{Beta} & X[\alpha \leftarrow Z] \\
app(\Delta\alpha.X, Z) & \rightarrow_{\Delta 1} & \Delta\beta.(X[\alpha \leftarrow \lambda\gamma.(app(\beta, app(\gamma, Z)))]) \\
\Delta\alpha.(app(\alpha, X)) & \rightarrow_{\Delta 2} & X \\
\Delta\alpha.(app(\alpha, (\Delta\beta.(app(\alpha, X))))) & \rightarrow_{\Delta 3} & X
\end{array}$$

Example 6.18 A further example is the foldl recursion scheme over lists.

$$\begin{array}{lll}
foldl(\xi\alpha.(\xi\beta.X), Y, nil) & \rightarrow_{f1} & Y \\
foldl(\xi\alpha.(\xi\beta.X), Y, cons(Z, W)) & \rightarrow_{f2} & foldl(\xi\alpha.(\xi\beta.X), X[\alpha \leftarrow Y][\beta \leftarrow Z], W)
\end{array}$$

We shall now proceed to define the way in which rewrite rules are instantiated in order to obtain the induced rewrite relation on terms. This implies defining how the 'holes' in the metaterms of the rule, represented by t-metavariables and o-metavariables, are replaced by terms and variables, respectively. Thus the notion of valuation shall be introduced, followed by some additional conditions imposed on these valuations in order to single out the 'good' valuations (referred to as *admissible valuations*) from the 'bad' ones. A word on notation: if θ is a (partial) function from a set S_1 to a set S_2 then we use $Dom(\theta)$ to denote its domain, i.e. the subset of S_1 for which it is defined.

Definition 6.19 (Valuation) A *variable assignment* over Σ is a (partial) function θ_v from o-metavariables to variables with finite domain, such that for every pair of o-metavariables $\alpha, \widehat{\beta}$ we have $\theta_v\alpha \neq \theta_v\widehat{\beta}$ (pre-bound and pre-free o-metavariables are assigned different values).

A *valuation* θ over Σ is a pair of (partial) functions (θ_v, θ_t) where θ_v is a variable assignment over Σ and θ_t maps t-metavariables in Σ to terms over Σ. It is defined as:

$$\begin{array}{lll}
\theta\alpha & \overset{\text{def}}{=} & \theta_v\alpha \\
\theta\widehat{\alpha} & \overset{\text{def}}{=} & \theta_v\widehat{\alpha} \\
\theta X & \overset{\text{def}}{=} & \theta_t X
\end{array}$$

A valuation θ may be extended in a unique way to the set of pre-metaterms M such that $MVar(M) \subseteq Dom(\theta)$, where $Dom(\theta)$ denotes the domain of θ, as follows:

$$\begin{array}{lll}
\overline{\theta}(f(M_1, \ldots, M_n)) & \overset{\text{def}}{=} & f(\overline{\theta}M_1, \ldots, \overline{\theta}M_n) \\
\overline{\theta}(\xi\alpha.(M_1, \ldots, M_n)) & \overset{\text{def}}{=} & \xi\theta_v\alpha.(\overline{\theta}M_1, \ldots, \overline{\theta}M_n) \\
\overline{\theta}(M_1[\alpha \leftarrow M_2]) & \overset{\text{def}}{=} & \overline{\theta}(M_1)\{\theta_v\alpha \leftarrow \overline{\theta}M_2\}
\end{array}$$

We shall not distinguish between θ and $\overline{\theta}$ if no ambiguities arise. Also, we sometimes write $\theta(M)$ thereby implicitly assuming that $MVar(M) \subseteq Dom(\theta)$.

Returning to the intuition behind *v-equivalence* the idea is that it can be translated into α-conversion in the sense that $M =_v N$ implies $\theta M =_\alpha \theta N$ for any valuation θ such that θM and θN are defined. Indeed, coming back to Example 6.12 and taking $\theta = \{\alpha/x, \beta/y, X/x\}$, we have $\theta(\lambda\alpha.\alpha) = \lambda x.x =_\alpha \lambda y.y = \theta(\lambda\beta.\beta)$ and $\theta(\lambda\alpha.f) = \lambda x.f =_\alpha \lambda y.f = \theta(\lambda\beta.f)$. However $\theta(\lambda\alpha.X) = \lambda x.x \neq_\alpha \lambda y.x = \theta(\lambda\beta.X)$ and $\theta(\lambda\beta.\lambda\alpha.X) = \lambda y.\lambda x.x \neq_\alpha \lambda x.\lambda y.x = \theta(\lambda\alpha.\lambda\beta.X)$.

As the reader may have observed, a valuation computes a metasubstitution operator by executing metalevel substitution. However, since metalevel substitution is restricted in that no α-conversion is allowed to take place, we must require the valuation to be capable of executing all metasubstitution operators in a given pre-metaterm.

Definition 6.20 (Safe valuations) Let M be a pre-metaterm over Σ and θ a valuation over Σ. We say that *θ is safe for M* if $MVar(M) \subseteq Dom(\theta)$ and $\overline{\theta}M$ is defined, i.e. the substitutions generated by the last clause of Def. 6.19 can be computed. Likewise, if (G, D) is a rewrite rule, we say that θ is *safe for* (G, D) if $\overline{\theta}D$ is defined.

Note that if the notion of substitution we are dealing with were not restricted then α-conversion could be required in order to apply a valuation to a pre-metaterm. Also, for any valuation θ and pre-metaterm M with $MVar(M) \subseteq Dom(\theta)$ that contains no occurrences of the metasubstitution operator θ is safe for M. Thus, we only ask θ to be safe for D (not G) in the previous definition.

The following condition is the classical notion of *admissibility* used in higher-order rewriting [Raa96] to avoid inconsistencies in rewrite steps. It runs under the name "variable-capture-freeness" in the case of *ERS* [KOO01a] and aims at ruling out certain valuations which after instantiating a rewrite rule leave some free and bound occurrences of the same variable. An example is the rewrite rule $\lambda\alpha.X \twoheadrightarrow X$ and the valuation which assigns x to α and X. The resulting rewrite step is $\lambda x.x \twoheadrightarrow x$ which has an occurrence of x which is bound on the left and and occurrence of x which is free on the right.

Definition 6.21 (Path condition for t-metavariables) Let X be a t-metavariable. Consider all the occurrences $p_1, \dots, p_n$ of X in (G, D), and their respective parameter paths $l_1, \dots, l_n$ in the trees corresponding to G and D. A valuation θ verifies *the path condition* for X in (G, D) if for every $x \in FV(\theta X)$, either (for all $1 \leq i \leq n$ we have $x \in \theta l_i$) or (for all $1 \leq i \leq n$ we have $x \notin \theta l_i$).

This definition may be read as: one occurrence of $x \in FV(\theta X)$ with X in (G, D) is in the scope of some binding occurrence of x iff every occurrence of X in (G, D) is in the scope of a bound o-metavariable α with $\theta\alpha = x$. For example, consider the *SERS* rule $\lambda\alpha.(\xi\beta.X) \twoheadrightarrow \xi\beta.X$ and the valuations $\theta_1 = \{\alpha/x, \beta/y, X/z\}$ and $\theta_2 = \{\alpha/x, \beta/y, X/x\}$. Then θ_1 verifies the path condition for X, but θ_2 does not since when instantiating the rewrite rule with θ_2 the variable x shall occur both bound (on the *LHS*) and free (on the *RHS*).

Note that our formalism allows us to specify the restricted garbage collection rule *rGc* of λx (Example 6.16) as originally done in [Ros92], while formalisms such as *CRS* force one to change this rule to a stronger one, namely *Gc*, written as $subs(\sigma\alpha.X, Z) \twoheadrightarrow_{Gc} X$, where the path condition (Def. 6.21) on valuations guarantees that if X/t is part of the valuation θ, then $\theta(\alpha)$ cannot be in $FV(t)$.

We may then single out the 'good' valuations by the following notion of admissible valuations.

Definition 6.22 (Admissible valuations) A valuation θ over Σ is *admissible for a rewrite rule* (G, D) over Σ iff the following conditions hold:

- θ is safe for (G, D),
- if α and β occur in (G, D) with $\alpha \neq \beta$ then $\theta_v\alpha \neq \theta_v\beta$, and
- θ verifies the path condition for every t-metavariable in (G, D).

Note that an admissible valuation is safe by definition, but a safe valuation may not be admissible: consider the rule $\lambda\alpha.app(X, \alpha) \twoheadrightarrow X$, the valuation $\theta = \{\alpha/x, X/x\}$ is trivially safe (there is no metasubstitution operator on the *RHS*) but is not admissible since the path condition is not verified: $x \in \theta(\alpha)$ but $x \notin \theta(\epsilon)$ (x occurs bound on the *LHS* and free on the *RHS*).

Having defined rewrite rules and (admissible) valuations we find ourselves ready to present the rewrite relation induced on terms by a rewrite rule.

Definition 6.23 (Rewriting terms) Let $(\Sigma, \mathcal{R})$ be a set of *SERS*-rewrite rules and s, t terms over Σ. We say that s *$\mathcal{R}$-rewrites* or *$\mathcal{R}$-reduces* or *$\mathcal{R}$-contracts to* t, written $s \twoheadrightarrow_{\mathcal{R}} t$, iff there exists a rewrite rule $(G, D) \in \mathcal{R}$, an admissible valuation θ for (G, D) and a context C such that $s =_\alpha C[\theta G]$ and $t =_\alpha C[\theta D]$. The term θG is called a (G, D)*-redex*. A redex in a term M is determined by a rewrite rule and a position in *tree*(M).

We shall occasionally drop the subscript in the rewrite relation when it is clear from the context. Note that, as in first-order rewriting, rewriting does not create new variables.

Lemma 6.24 Let θ be an admissible valuation for a rewrite rule (G, D). Then $FV(\theta D) \subseteq FV(\theta G)$.

Proof. Suppose $x \in FV(\theta D)$. Then

- if x comes from a free o-metavariable $\widehat{\beta}$ occurring in D with $\theta\widehat{\beta} = x$, then since $FMVar(D) \subseteq FMVar(G)$ we also have that $\widehat{\beta}$ occurs in G. Moreover, by definition of valuation, variables assigned to free o-metavariables cannot be captured, so that we necessarily have $x \in FV(\theta G)$.
- if x comes from instantiating a t-metavariable Z occurring in D, then Z occurs in D at position p_D, $x \in FV(\theta Z)$ and x does not appear in $\theta(l_D)$, where l_D is the parameter path of p_D in D. Now, since $FMVar(D) \subseteq FMVar(G)$, we also have that Z occurs in G, let us say at position p_G. Suppose that $x \notin FV(\theta G)$. The only possible case is that x was captured in θG so that $x \in BV(\theta G)$. Therefore x appears in $\theta(l_G)$, where l_G is the parameter path of p_G in G, which contradicts the fact that θ verifies the path condition for Z in (G, D).

The rewrite relation on terms satisfies the following property:

Corollary 6.25 Let $s, t \in \mathcal{T}$, if $s \rightarrow_{\mathcal{R}} t$, then $FV(t) \subseteq FV(s)$.

Proof. If $s \rightarrow_{\mathcal{R}} t$, then there exists a rule $(G, D) \in \mathcal{R}$ and a context C such that $s =_{\alpha} C[\theta G]$ and $t =_{\alpha} C[\theta D]$ where θ is an admissible valuation for (G, D). Since $FV(C[\theta G]) = FV(s)$ and $FV(C[\theta D]) = FV(t)$ we can reason directly on $C[\theta G]$ and $C[\theta D]$. Suppose $x \in FV(C[\theta D])$. Then either $x \in FV(C)$, in which case we trivially have $x \in FV(C[\theta G])$, or $x \in FV(\theta D)$. Then by Lemma 6.24 $x \in FV(\theta G)$ and since x is not captured in C, we also have $x \in FV(C[\theta G])$. $\bullet$

6.2 Simplified Expression Reduction Systems with Indices

We introduce the de Bruijn indices based higher-order rewrite formalism $SERS_{db}$. We shall follow Section 6.1 and introduce de Bruijn metaterms, de Bruijn terms, de Bruijn valuation, and finally, de Bruijn rewriting. We shall thus put in practice the following notational convention: in order to distinguish a concept defined for the *SERS* formalism from its corresponding version (if it exists) in the $SERS_{db}$ formalism we may prefix it using the qualifying term 'de Bruijn', eg. 'de Bruijn metaterms'.

Definition 6.26 (de Bruijn signature) A $SERS_{db}$ signature Σ consists of the denumerable (and possibly infinite) disjoint sets:

- $\Sigma_{bi} = \{\alpha_1, \alpha_2, \alpha_3, \ldots\}$ a set of symbols called *binder indicators*, denoted $\alpha, \beta, \ldots$
- $\Sigma_{imv} = \{\widehat{\alpha_1}, \widehat{\alpha_2}, \ldots\}$ a set of *i-metavariables* (i for index), denoted $\widehat{\alpha}, \widehat{\beta}, \ldots$
- $\Sigma_{tmv} = \{X_l^1, X_l^2, X_l^3, \ldots\}$ a set of *t-metavariables* (t for term), where l ranges over the set of labels built over binder indicators, denoted $X_l, Y_l, Z_l, \ldots$
- $\Sigma_f = \{f_1, f_2, f_3, \ldots\}$ a set of *function symbols* equipped with a fixed (possibly zero) arity, denoted $f, g, h, \ldots$
- $\Sigma_b = \{\lambda_1, \lambda_2, \lambda_3, \ldots\}$ a set of *binder symbols* equipped with a fixed (non-zero) arity, denoted $\lambda, \mu, \nu, \xi, \ldots$

We remark that the set of binder indicators is exactly the set of pre-bound o-metavariables introduced in Def. 6.1 (Σ_{bmv}). The reason for using the same alphabet in both formalisms shall become clear in Section 6.3, but intuitively, we need a mechanism to annotate binding paths in the de Bruijn setting to distinguish metaterms like $\xi\beta.(\xi\alpha.X)$ and $\xi\alpha.(\xi\beta.X)$ appearing in the same rule when translated into a $SERS_{db}$ system.

Definition 6.27 (de Bruijn pre-metaterms) The set of *de Bruijn pre-metaterms* over Σ, denoted $\mathcal{PMT}_{db}$, is defined by the following two-sorted grammar:

$$\begin{array}{llll} \text{metaindices} & I & ::= & 1 \mid \mathtt{S}(I) \mid \widehat{\alpha} \\ \text{pre-metaterms} & A & ::= & I \mid X_l \mid f(A, \ldots, A) \mid \xi(A, \ldots, A) \mid A[\![A]\!] \end{array}$$

The symbol $\bullet[\![\bullet]\!]$ in a pre-metaterm $A[\![A]\!]$ is called *de Bruijn metasubstitution operator*. The binder symbols together with the de Bruijn metasubstitution operator are called *binder operators*, thus the de Bruijn metasubstitution operator is a binder operator (since it has binding power) but is *not* a binder symbol since it is not an element of Σ_b.

We shall use $A, B, A_i, \ldots$ to denote de Bruijn pre-metaterms and the convention that $\mathtt{S}^0(1) = 1$, $\mathtt{S}^0(\widehat{\alpha}) = \widehat{\alpha}$ and $\mathtt{S}^{j+1}(n) = \mathtt{S}(\mathtt{S}^j(n))$. As usually done for indices, we shall abbreviate $\mathtt{S}^{j-1}(1)$ as j.

Positions may be defined by associating a tree to each de Bruijn pre-metaterm, as was done in the case of *SERS*. As one might expect, *tree*(A) must have one of the forms depicted in Figure 6.2. The '*subs*' in the rightmost tree may be seen as a dummy function symbol.

Even if the formal mechanism used to translate pre-metaterms with names into pre-metaterms with de Bruijn indices will be given in Section 6.3, let us introduce intuitively some ideas in order to justify the syntax used for i-metavariables. In the formalism *SERS* there is a clear distinction between free and bound o-metavariables. This fact must also be reflected in $SERS_{db}$, where bound o-metavariables are represented with indices and free

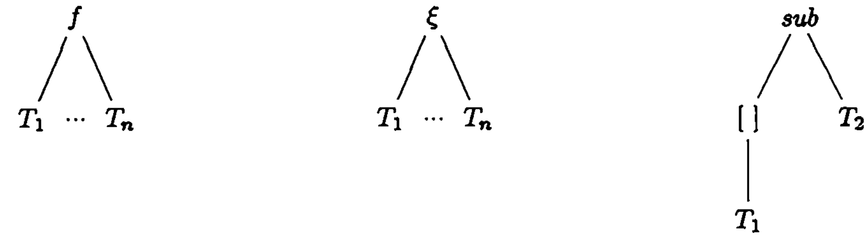


Figure 6.2: de Bruijn Pre-Metaterms as Trees

o-metavariables are represented with i-metavariables (this distinction between free and bound variables is also used in some formalizations of λ-calculus [Pol93]). However, free variables in $SERS_{db}$ appear always in a binding context, so that a de Bruijn valuation of such kind of variables has to reflect the adjustment needed to represent the same variables but in a different context. This can be done by surrounding the i-metavariable by as many operators S as necessary. As an example consider the pre-metaterm $\xi\alpha.\beta$. If we translate it to $\xi\widehat{\beta}$, then a de Bruijn valuation like $\kappa = \{\widehat{\beta}/1\}$ binds the variable whereas this is completely impossible in the name formalism thanks to the conditions imposed on a name valuation (condition on variable assignments in Def. 6.19). Our solution is then to translate the pre-metaterm $\xi\alpha.\beta$ by $\xi(\mathtt{S}(\widehat{\beta}))$ in such a way that there is no capture of variables since $\kappa(\xi(\mathtt{S}(\widehat{\beta})))$ is exactly $\xi(2)$. The solution adopted here for translating pre-free o-metavariables into the de Bruijn formalism is in some sense what is called *pre-cooking* in [DHK95]: the pre-cooking function takes a $\lambda\sigma$-term with t-metavariables and suffixes them with as many explicit shift operators as the number of binders present in its parameter path. This avoids variable capture when the higher-order unification procedure finds solutions for the t-metavariables.

We use $MVar(A)$ (resp. $MVar_i(A)$ and $MVar_t(A)$) to denote the set of all metavariables (resp. i- and t-metavariables) of the de Bruijn pre-metaterm A.

As in the *SERS* formalism, we also need here a notion of well-formed pre-metaterm. The first motivation is to guarantee that labels of t-metavariables are correct w.r.t the context in which they appear, the second one is to ensure that indices like j (resp. $\mathtt{S}^j(\widehat{\alpha})$) correspond to bound (resp. free) variables. Indeed, the pre-metaterms $\xi(X_{\alpha\beta})$, $\xi(\xi(4))$ and $\xi(\widehat{\alpha})$ shall not make sense for us, and hence shall not be considered well-formed.

Definition 6.28 (de Bruijn metaterms) A pre-metaterm $A \in \mathcal{PMT}_{db}$ over Σ is said to be a *metaterm* over Σ iff the predicate $\mathcal{WF}(A)$ holds, where $\mathcal{WF}(A)$ iff $\mathcal{WF}_\epsilon(A)$, and $\mathcal{WF}_l(A)$ is defined by induction on the structure of the pre-metaterm A for any label l as follows:

- $\mathcal{WF}_l(\mathtt{S}^j(1))$ iff $j+1 \leq |l|$
- $\mathcal{WF}_l(\mathtt{S}^j(\widehat{\alpha}))$ iff $j = |l|$ and l is a simple label
- $\mathcal{WF}_l(X_k)$ iff $l = k$ and l is a simple label
- $\mathcal{WF}_l(f(A_1, \ldots, A_n))$ iff for all $1 \leq i \leq n$ we have $\mathcal{WF}_l(A_i)$
- $\mathcal{WF}_l(\xi(A_1, \ldots, A_n))$ iff there exists $\alpha \notin l$ such that for all $1 \leq i \leq n$ we have $\mathcal{WF}_{\alpha l}(A_i)$
- $\mathcal{WF}_l(A_1[\![A_2]\!])$ iff $\mathcal{WF}_l(A_2)$ and there exists $\alpha \notin l$ such that $\mathcal{WF}_{\alpha l}(A_1)$

Therefore indices of the form $\mathtt{S}^j(1)$ may only occur in metaterms if they represent bound variables and well-formed metaindices of the form $\mathtt{S}^j(\widehat{\alpha})$ always represent a free variable. Note that when considering $\mathcal{WF}_l(M)$ and $\mathcal{WF}_l(A)$ it is Definitions 6.5 and 6.28 which are referenced, respectively.

Example 6.29 Pre-metaterms $\xi(X_\alpha, \lambda(Y_{\beta\alpha}, 2))$, $f(\widehat{\beta}, \lambda(Y_\alpha, \mathtt{S}(\widehat{\alpha})))$ and $g(\lambda(\xi c))$ are metaterms, whereas the pre-metaterms $f(\mathtt{S}(\widehat{\alpha}), \xi(X_\beta))$, $\lambda(\xi(X_{\alpha\alpha}))$, $f(\widehat{\beta}, \lambda(\xi(\mathtt{S}(\widehat{\beta}))))$ are not.

Definition 6.30 (Linear metaterms) A de Bruijn pre-metaterm (or metaterm) M is *linear* if it contains at most one occurrence of any X-based metavariable. Note that the de Bruijn metaterm $f(\lambda(\xi(X_{\alpha\beta})), \xi(\lambda(X_{\beta\alpha})))$ is not linear since there are two occurrences of X-based metavariables, neither is $f(\lambda(X_\alpha), \xi(X_\alpha))$. However, $app(\lambda X_\alpha, Y_\epsilon)$ is linear.

Definition 6.31 (Pattern of a metaterm) The pattern of a metaterm A is the metacontext[3] obtained by replacing every metavariable X_l by some hole $\Box_{X_l}$. Note that if there is more than one occurrence of a metavariable then there shall be more than one occurrence of its associated hole. We write $\mathtt{pattern}(A)$ for the pattern of A.

Definition 6.32 (de Bruijn terms and de Bruijn contexts) The set of *de Bruijn terms* over Σ, denoted $\mathcal{T}_{db}$, and the set of de Bruijn contexts over Σ are defined by:

$$\begin{array}{lcll} \text{de Bruijn indices} & n & ::= & 1 \mid \mathtt{S}(n) \\ \text{de Bruijn terms} & a & ::= & n \mid f(a,\ldots,a) \mid \xi(a,\ldots,a) \\ \text{de Bruijn contexts} & E & ::= & \Box \mid f(a,\ldots,E,\ldots,a) \mid \xi(a,\ldots,E,\ldots,a) \end{array}$$

We use $a, b, a_i, b_i, \ldots$ for de Bruijn terms and $E, F, \ldots$ for de Bruijn contexts. The notion of *tree*(a) may be defined as for de Bruijn pre-metaterms. We may refer to the *binder path number* of a context, which is the number of binders between the $\Box$ and the root. In contrast to Def. 6.8, we have here that de Bruijn terms are also de Bruijn pre-metaterms, that is, $\mathcal{T}_{db} \subset \mathcal{PMT}_{db}$, although note that some de Bruijn terms may not be de Bruijn metaterms, i.e. may not be well-formed de Bruijn pre-metaterms. Indeed, the valid term $\xi(\xi(4))$ is not a metaterm, however, the index 4 may be seen as a constant in the metaterm $\xi(\xi(4))$. If an arbitrary free variable is wished to be represented in a metaterm, then i-metavariables should be used.

Definition 6.33 (Free de Bruijn variables) We denote by $FV(a)$ the set of *free variables* of a de Bruijn term a, which is defined as follows:

$$\begin{array}{lcl} FV(n) & \stackrel{\text{def}}{=} & \{n\} \\ FV(f(a_1,\ldots,a_n)) & \stackrel{\text{def}}{=} & \bigcup_{i=1}^{n} FV(a_i) \\ FV(\xi(a_1,\ldots,a_n)) & \stackrel{\text{def}}{=} & (\bigcup_{i=1}^{n} FV(a_i))\setminus\!\!\setminus 1 \end{array}$$

where for every set of indices S, the operation $S\setminus\!\!\setminus j$ is defined as $\{n - j \mid n \in S \text{ and } n > j\}$.

When encoding $SERS_{db}$ systems as $SERS$ systems we shall need to speak of the free variable names (objects in Σ_v, from the definition of a $SERS$ signature) associated to the free de Bruijn indices. For example, if $a = \xi(1,2,3)$, then $FV(a) = \{1,2\}$. The named variable associated to the free index 1 is x_1, and likewise for 2 it is x_2. In general, we write $\mathtt{Names}(S)$ for the names of the variables whose indices are in the set S. For example, $\mathtt{Names}(FV(a)) = \{x_1, x_2\}$.

Definition 6.34 (de Bruijn substitution) The result of substituting a term b for the index $n \geq 1$ in a term a is denoted $a\{\!\{n \leftarrow b\}\!\}$ and defined:

$$\begin{array}{lcl} f(a_1,\ldots,a_n)\{\!\{n \leftarrow b\}\!\} & \stackrel{\text{def}}{=} & f(a_1\{\!\{n \leftarrow b\}\!\},\ldots,a_n\{\!\{n \leftarrow b\}\!\}) \\ \xi(a_1,\ldots,a_n)\{\!\{n \leftarrow b\}\!\} & \stackrel{\text{def}}{=} & \xi(a_1\{\!\{n+1 \leftarrow b\}\!\},\ldots,a_n\{\!\{n+1 \leftarrow b\}\!\}) \\ m\{\!\{n \leftarrow b\}\!\} & \stackrel{\text{def}}{=} & \begin{cases} m-1 & \text{if } m > n \\ \mathcal{U}_0^n(b) & \text{if } m = n \\ m & \text{if } m < n \end{cases} \end{array}$$

where for $i \geq 0$ and $n \geq 1$ we define the *updating functions* $\mathcal{U}_i^n(\bullet)$ as follows:

$$\begin{array}{lcl} \mathcal{U}_i^n(f(a_1,\ldots,a_n)) & \stackrel{\text{def}}{=} & f(\mathcal{U}_i^n(a_1),\ldots,\mathcal{U}_i^n(a_n)) \\ \mathcal{U}_i^n(\xi(a_1,\ldots,a_n)) & \stackrel{\text{def}}{=} & \xi(\mathcal{U}_{i+1}^n(a_1),\ldots,\mathcal{U}_{i+1}^n(a_n)) \\ \mathcal{U}_i^n(m) & \stackrel{\text{def}}{=} & \begin{cases} m+n-1 & \text{if } m > i \\ m & \text{if } m \leq i \end{cases} \end{array}$$

Due to the various notions of substitution and replacement introduced so far, in Figure 6.3 we give a brief synopsis of the situation. We abbreviate "not applicable" by "na". In the case of valuations, we use the same notation for $SERS$ valuations and $SERS_{db}$ valuations.

[3] In full precision we obtain a metaterm with possibly many holes. However, by abuse of notation we shall speak in terms of metacontexts.

Operator	Names	de Bruijn	Terms	Metaterms	Explicit	Implicit
$M_1[\alpha \leftarrow M_2]$	$\surd$			$\surd$	$\surd$	
$t_1\{x \leftarrow t_2\}$	$\surd$		$\surd$			$\surd$
$A_1\llbracket A_2 \rrbracket$		$\surd$		$\surd$	$\surd$	
$a_1\{\!\{n \leftarrow a_2\}\!\}$		$\surd$	$\surd$			$\surd$
$M \ll \alpha \leftarrow \beta \gg$	$\surd$			$\surd$		$\surd$
valuations: $\{\bullet/\bullet, \ldots, \bullet/\bullet\}$	$\surd$	$\surd$	na	na	na	na

Figure 6.3: Notions of substitution

Definition 6.35 (Free de Bruijn metavariables) Let A be a de Bruijn pre-metaterm. The set of *free metavariables* of A, $FMVar(A)$, is defined as:

$$\begin{array}{lcl} FMVar(1) & \stackrel{\text{def}}{=} & \emptyset \\ FMVar(\mathtt{S}(I)) & \stackrel{\text{def}}{=} & FMVar(I) \\ FMVar(\widehat{\alpha}) & \stackrel{\text{def}}{=} & \{\widehat{\alpha}\} \\ FMVar(X_l) & \stackrel{\text{def}}{=} & \{X_l\} \end{array} \qquad \begin{array}{lcl} FMVar(f(A_1, \ldots, A_n)) & \stackrel{\text{def}}{=} & \bigcup_{i=1}^{n} FMVar(A_i) \\ FMVar(\xi(A_1, \ldots, A_n)) & \stackrel{\text{def}}{=} & \bigcup_{i=1}^{n} FMVar(A_i) \\ FMVar(A_1\llbracket A_2 \rrbracket) & \stackrel{\text{def}}{=} & FMVar(A_1) \cup FMVar(A_2) \end{array}$$

Note that this definition also applies to de Bruijn metaterms. The set of *names* of free metavariables of A is the set of free metavariables of A where each X_l is replaced simply by X. We shall write $NFMVar(A)$ for the names of the free metavariables in A. For example, $NFMVar(f(\lambda X_\alpha, Y_\epsilon, \widehat{\alpha})) = \{X, Y, \widehat{\alpha}\}$. This notion will be used for the first time in Def. 6.36.

We now consider the rewrite rules of an $SERS_{db}$. This includes defining valuations, their validity, and the term rewrite relation in $SERS_{db}$. Rewrite rules are specified with de Bruijn metaterms, whereas the induced rewrite relation is on de Bruijn terms.

Definition 6.36 (de Bruijn rewrite rule) A *de Bruijn rewrite rule* over Σ is a pair of de Bruijn metaterms (L, R) over Σ (also written $L \rightarrow R$) such that:

- the first symbol (called head symbol) in L is a function symbol or a binder symbol,
- $NFMVar(R) \subseteq NFMVar(L)$, and
- the metasubstitution operator does not occur in L.

Definition 6.37 (*SERS*) A $SERS_{db}$ is a pair $(\Sigma, \mathcal{R})$ where Σ is a $SERS_{db}$-signature and $\mathcal{R}$ is a set of $SERS_{db}$-rewrite rules over Σ.

As in the case of *SERS*, we shall often omit Σ and write $\mathcal{R}$ instead of $(\Sigma, \mathcal{R})$, if no confusion arises.

Example 6.38 The λ_{db}-calculus is defined by considering the signature containing the function symbols $\Sigma_f = \{app\}$ and binder symbols $\Sigma_b = \{\lambda\}$, together with the following $SERS_{db}$-rewrite rule: $app(\lambda X_\alpha, Z_\epsilon) \rightarrow_{\beta_{db}} X_\alpha\llbracket Z_\epsilon \rrbracket$. The $\lambda_{db}\eta_{db}$-calculus is obtained by adding the following $SERS_{db}$-rewrite rule: $\lambda(app(X_\alpha, 1)) \rightarrow_{\eta_{db}} X_\epsilon$.

See also Examples 6.53 and 6.54.

Definition 6.39 (de Bruijn valuation) A *de Bruijn valuation* κ over Σ is a pair of (partial) functions (κ_i, κ_t) where κ_i is a function from i-metavariables to positive integers[4], and κ_t is a function from t-metavariables to de Bruijn terms. It is defined as:

$$\begin{array}{lcl} \kappa 1 & \stackrel{\text{def}}{=} & 1 \\ \kappa \mathtt{S}(I) & \stackrel{\text{def}}{=} & \mathtt{S}(\kappa I) \\ \kappa \widehat{\alpha} & \stackrel{\text{def}}{=} & \kappa_i \widehat{\alpha} \\ \kappa X_l & \stackrel{\text{def}}{=} & \kappa_t X_l \end{array}$$

[4] Integers greater than 0.

A valuation κ determines in a unique way a function $\overline{\kappa}$ from the set of pre-metaterms A with $FMVar(A) \subseteq Dom(\kappa)$, where $Dom(\kappa)$ denotes the domain of κ, to the set of terms as follows:

$$\begin{array}{lcl} \overline{\kappa}(f(A_1,\ldots,A_n)) & \overset{\text{def}}{=} & f(\overline{\kappa}A_1,\ldots,\overline{\kappa}A_n) \\ \overline{\kappa}(\xi(A_1,\ldots,A_n)) & \overset{\text{def}}{=} & \xi(\overline{\kappa}A_1,\ldots,\overline{\kappa}A_n) \\ \overline{\kappa}(A_1[\![A_2]\!]) & \overset{\text{def}}{=} & \overline{\kappa}(A_1)\{\!\{1 \leftarrow \overline{\kappa}A_2\}\!\} \end{array}$$

Note that in the above definition the substitution operator $\bullet\{\!\{\bullet \leftarrow \bullet\}\!\}$ refers to the usual substitution defined on terms with de Bruijn indices (Def. 6.34).

We now introduce the notion of *value function* which is used to give semantics to metavariables with labels in the $SERS_{db}$ formalism. The goal pursued by the labels of metavariables is that of incorporating 'context' information as a defining part of a metavariable. As a consequence, we must verify that the terms substituted for every occurrence of a fixed metavariable coincide 'modulo' their corresponding context. Dealing with such notion of 'coherence' of substitutions in a de Bruijn formalism is also present in other formalisms but in a more restricted form. Thus for example, as mentioned before, a pre-cooking function is used in [DHK95] in order to avoid variable capture in the higher-order unification procedure. In *XRS* [Pag98] the notions of binding arity and pseudo-binding arity are introduced in order to take into account the parameter path of the different occurrences of t-metavariables appearing in a rewrite rule. Then it is required (roughly) that the binding arity of a t-metavariable on the *LHS* of a rewrite rule (rewrite rules are required to be left-linear) equals the pseudo-binding arity of the same t-metavariable occurring on the *RHS* of the rule. Our notion of 'coherence' is implemented with *valid valuations* (Def. 6.41) and it turns out to be more general than the solutions proposed in [DHK95] and [Pag98].

Definition 6.40 (Value function) Let $a \in \mathcal{T}_{db}$ and l be a label of binder indicators. Then we define the *value function* $Value(l,a)$ as $Value^0(l,a)$ where

$$\begin{array}{lcl} Value^i(l,n) & \overset{\text{def}}{=} & \begin{cases} n & \text{if } n \leq i \\ \mathtt{at}(l, n-i) & \text{if } 0 < n-i \leq |l| \\ x_{n-i-|l|} & \text{if } n-i > |l| \end{cases} \\ Value^i(l, f(a_1,\ldots,a_n)) & \overset{\text{def}}{=} & f(Value^i(l,a_1),\ldots Value^i(l,a_n)) \\ Value^i(l, \xi(a_1,\ldots,a_n)) & \overset{\text{def}}{=} & \xi(Value^{i+1}(l,a_1),\ldots, Value^{i+1}(l,a_n)) \end{array}$$

It is worth noting that $Value^i(l,n)$ may give three different kinds of results. This is just a technical resource to make easier later proofs. Indeed, we have for example $Value(\alpha\beta, \xi(f(3,1))) = \xi(f(\beta,1)) = Value(\beta\alpha, \xi(f(2,1)))$ and $Value(\epsilon, f(\xi 1, \lambda 2)) = f(\xi 1, \lambda x_1) \neq f(\xi 1, \lambda\alpha) = Value(\alpha, f(\xi 1, \lambda 2))$. Thus the function $Value(l,a)$ interprets the de Bruijn term a in an l-context: bound indices are left untouched, free indices referring to the l-context are replaced by the corresponding binder indicator and the remaining free indices are replaced by their corresponding variable names. It might be observed that if repeated binder indicators are allowed in the label l of Def. 6.40 then this intuition would not seem to hold. Indeed, for our purposes the case of interest is when the label l is simple. Nevertheless, many auxiliary results may be proved without this requirement thus we prefer not to restrict this definition prematurely (by requiring l to be simple).

In order to introduce the notion of valid de Bruijn valuations let us consider the following rule:

$$\xi\alpha.(\xi\beta.X) \rightarrow_r \xi\beta.(\xi\alpha.X)$$

Even if translation of rewrite rules into de Bruijn rewrite rules has not been defined yet (Section 6.3), one may guess that a reasonable translation would be the following rule:

$$\xi(\xi(X_{\beta\alpha})) \rightarrow_{r_{db}} \xi(\xi(X_{\alpha\beta}))$$

which indicates that β (resp. α) is the first bound occurrence in the *LHS* (resp. *RHS*) while α (resp. β) is the second bound occurrence in the *LHS* (resp. *RHS*). Now, if X is instantiated by x, α by x and β by y in the *SERS* system, then we have an r-rewrite step $\xi x.(\xi y.(x)) \rightarrow \xi y.(\xi x.(x))$. However, to reflect this fact in the corresponding $SERS_{db}$ system we need to instantiate $X_{\beta\alpha}$ by 2 and $X_{\alpha\beta}$ by 1, thus obtaining an r_{db}-rewrite step $\xi(\xi 2) \rightarrow \xi(\xi 1)$. This clearly shows that de Bruijn t-metavariables having the same name but different label cannot be instantiated arbitrarily as they have to reflect the renaming of variables which is indicated by their labels. This is exactly the role of the property of validity:

Definition 6.41 (Valid de Bruijn valuation) A de Bruijn valuation κ over Σ is said to be *valid* if for every pair of t-metavariables X_l and $X_{l'}$ in $Dom(\kappa)$ we have $Value(l, \kappa X_l) = Value(l', \kappa X_{l'})$. Likewise, we say that a de Bruijn valuation κ is *valid for a rewrite rule* (L, R) if every metavariable in (L, R) is in $Dom(\kappa)$ and for every pair of t-metavariables X_l and $X_{l'}$ in (L, R) we have $Value(l, \kappa X_l) = Value(l', \kappa X_{l'})$.

It is interesting to note that there is no concept analogous to safeness (Def. 6.20) as used for named *SERS* due to the use of de Bruijn indices. Also, the last condition in the definition of an admissible valuation (Def. 6.22) is subsumed by the above Def. 6.41 in the setting of $SERS_{db}$.

Example 6.42 Returning to the above mentioned example we have that $\kappa = \{X_{\beta\alpha}/2, X_{\alpha\beta}/1\}$ is valid for the rule r_{db} since $Value(\beta\alpha, 2) = \alpha = Value(\alpha\beta, 1)$.

Another interesting example is the η-contraction rule $\lambda x.app(X, x) \rightarrow X$ if $x \notin FV(X)$. It can be expressed in the *SERS* formalism as the rule $\lambda\alpha.app(X, \alpha) \rightarrow_\eta X$, and in the $SERS_{db}$ formalism as the rule $\lambda(app(X_\alpha, 1)) \rightarrow_{\eta_{db}} X_\epsilon$.

Remark that this kind of rule cannot be expressed in the *XRS* formalism since it does not verify the binding arity condition. Our formalism allows us to write rules like η_{db} because valid valuations will test for coherence of values. Indeed, an admissible valuation for η is a valuation θ such that θX does not contain a free occurrence of $\theta(\alpha)$. This is exactly the condition used in any usual formalization of the η-rule. A valid valuation κ for η_{db} could, for example, be a valuation $\kappa = \{X_\alpha/m, X_\epsilon/n\}$ such that $Value(\alpha, \kappa X_\alpha) = Value(\epsilon, \kappa X_\epsilon)$, that is, $m = 1$ is not possible, and n is necessarily $m - 1$.

To summarize, valid valuations guarantee that the unique value assigned to a t-metavariable X in the framework with names is translated accordingly in the de Bruijn framework w.r.t the different label contexts of all the occurrences of X in the rewrite rule. This is, in some sense, an updating of X w.r.t the different label contexts where it appears, and it gives us the right notion of coherence for valuations.

Definition 6.43 (Rewriting de Bruijn terms) Let $\mathcal{R}$ be a set of de Bruijn rules over Σ and a, b de Bruijn terms, over Σ. We say that *a $\mathcal{R}$-rewrites* or *$\mathcal{R}$-reduces* or *$\mathcal{R}$-contracts to b*, written $a \rightarrow_\mathcal{R} b$, iff there is a de Bruijn rule $(L, R) \in \mathcal{R}$ and a de Bruijn valuation κ valid for (L, R) such that $a = E[\kappa L]$ and $b = E[\kappa R]$, where E is a de Bruijn context. The term κL is called an *(L, R)-redex*. A redex in a term a is determined by a rewrite rule and a position in $tree(a)$.

Thus, the term $\lambda(app(\lambda(app(1, 3)), 1))$ rewrites by the η_{db} rule to $\lambda(app(1, 2))$, using the (valid) valuation $\kappa = \{X_\alpha/\lambda(app(1, 3)), X_\epsilon/\lambda(app(1, 2))\}$.

Lemma 6.44 Let κ be a valid valuation for a rewrite rule (L, R). Then $FV(\kappa R) \subseteq FV(\kappa L)$.

Proof. Suppose $n \in FV(\kappa R)$. Then

- if n comes from a free o-metavariable $\widehat{\beta}$ occurring in R, then there is an m such that $S^m(\widehat{\beta})$ occurs in R and $\kappa\widehat{\beta} = n$, where m is the number of binders 'above' $\widehat{\beta}$ in R. Since L and R are de Bruijn metaterms and the names of $FMVar(R)$ are included in the names of $FMVar(L)$ there is an m' such that $S^{m'}(\widehat{\beta})$ occurs in L where m' is the number of binders 'above' this occurrence of $\widehat{\beta}$. Hence by definition of free variable $n \in FV(\kappa L)$.
- if n comes from instantiating a t-metavariable Z_l occurring in R then $n + |l| \in FV(\kappa Z_l)$. Since L and R are de Bruijn metaterms and the names of $FMVar(R)$ are included in the names of $FMVar(L)$ then there is an occurrence of $Z_{l'}$ in L for some label of binder indicators l'. Since κ is valid for (L, R) then $Value(l, \kappa Z_l) = Value(l', \kappa Z_{l'})$. Now as $n+|l| \in FV(\kappa Z_l)$ we have that x_n occurs in $Value(l, \kappa Z_l)$ and hence also in $Value(l', \kappa Z_{l'})$. Therefore $n + |l'| \in FV(\kappa Z_{l'})$. Finally, by well-formedness of the pre-metaterm L we have $n \in FV(\kappa L)$.

The rewrite relation on de Bruijn terms satisfies the following property:

Corollary 6.45 Let $a \in \mathcal{T}_{db}$. If $a \rightarrow_\mathcal{R} b$, then $FV(b) \subseteq FV(a)$.

Proof. If $a \rightarrow_\mathcal{R} b$, then there exists a valid valuation κ for a rewrite rule (L, R) in $\mathcal{R}$ such that $a = E[\kappa L]$ and $b = E[\kappa R]$. Then either $n \in FV(E)$, in which case we trivially have $n \in FV(a)$, or otherwise $n \in FV(\kappa R)$. Then by Lemma 6.44 also $n \in FV(\kappa L)$ and since n is not captured in E, we also have $n \in FV(a)$.

■

6.3 From Names to Indices...

In this section we show how rewriting in the *SERS* formalism may be simulated in the $SERS_{db}$ formalism. This requires two well-distinguished phases which we can refer to as the *definition phase* and the *rewrite-preservation phase*. The definition phase consists in defining appropriate translations from pre-metaterms, terms and valuations in the *SERS* setting into the corresponding notions in the $SERS_{db}$ setting, work which is carried out in the first part of this section. The second part deals with the rewrite-preservation phase, that is, showing how *SERS* rewrite steps can be simulated via $SERS_{db}$ rewrite steps. The rewrite-preservation phase shall use the results developed in the definition phase.

6.3.1 The Definition Phase

We shall begin by showing how to translate terms to de Bruijn terms.

Definition 6.46 (From terms (and contexts) to de Bruijn terms (and contexts)) The translation of a term t, denoted $T(t)$, is defined as $T_\epsilon(t)$ where

$$\begin{array}{lcl} T_k(x) & \overset{\text{def}}{=} & \begin{cases} \text{pos}(x,k) & \text{if } x \in k \\ \mathcal{O}(x)+|k| & \text{if } x \notin k \end{cases} \\ T_k(f(t_1,\ldots,t_n)) & \overset{\text{def}}{=} & f(T_k(t_1),\ldots,T_k(t_n)) \\ T_k(\xi x.(t_1,\ldots,t_n)) & \overset{\text{def}}{=} & \xi(T_{xk}(t_1),\ldots,T_{xk}(t_n)) \end{array}$$

The translation of a context, denoted $T(C)$, is defined as above but adding the clause $T_k(\Box) \overset{\text{def}}{=} \Box$.

As a consequence of the previous definition, there is a clear bijection between the set of free variables of a term t and the set of free variables of its de Bruijn representation $T(t)$.

The following lemma will be used in the main statements of this section; it states that variable renaming commutes with translation.

Lemma 6.47 Let $s \in \mathcal{T}$, let l, k be labels of variables and x, y variables such that y does not occur at all in s and $x, y \notin l$. If $s\{x \leftarrow y\}$ is defined then $T_{lyk}(s\{x \leftarrow y\}) = T_{lxk}(s)$.

Proof. By induction on s.

- $s = x$. Then we reason as follows:

$$\begin{array}{rcll} T_{lyk}(x\{x \leftarrow y\}) & = & T_{lyk}(y) & \\ & = & \text{pos}(y, lyk) & \\ & = & |l|+1 & (y \notin l) \\ & = & \text{pos}(x, lxk) & (x \notin l) \\ & = & T_{lxk}(s) & \end{array}$$

- $s = z \neq x$. Then $T_{lyk}(z\{x \leftarrow y\}) = T_{lyk}(z)$. We consider two further cases:
  - $z \in lyk$. Since $y \neq z$ (y does not occur in s) $T_{lyk}(z) = \text{pos}(z, lyk) = \text{pos}(z, lxk) = T_{lxk}(z)$.
  - $z \notin lyk$. Then $T_{lyk}(z) = \mathcal{O}(z) + |lyk| = \mathcal{O}(z) + |lxk| = T_{lxk}(z)$.
- $s = f(s_1, \ldots, s_n)$. We use the induction hypothesis.
- $s = \xi x.(s_1, \ldots, s_n)$. We reason as follows:

$$T_{lyk}(s\{x \leftarrow y\}) = T_{lyk}(\xi x.(s_1,\ldots,s_n)) = T_{lxk}(\xi x.(s_1,\ldots,s_n))$$

Now it may be shown that for any term s and variable x such that $x \notin FV(s)$, we have $T_{lxk}(s) = T_{lzk}(s)$ for any $z \notin FV(s)$. The last equality then follows from the fact that $x \notin FV(\xi x.(s_1, \ldots, s_n))$ and by considering $z = y$.

- $s = \xi z.(s_1, \dots, s_n)$ with $z \neq x$. Note that $z \neq y$ by hypothesis. We reason as follows:

$$\begin{array}{rcll} T_{lyk}(s\{x \leftarrow y\}) & = & \xi(T_{zlyk}(s_1\{x \leftarrow y\}), \dots, T_{zlyk}(s_n\{x \leftarrow y\})) & \\ & = & \xi(T_{zlxk}(s_1), \dots, T_{zlxk}(s_n)) & \text{(i.h.)} \\ & = & T_{lxk}(s) & \end{array}$$

As expected, the translation satisfies:

Lemma 6.48 (T is compatible with α-conversion) Given two terms $s, t \in \mathcal{T}$ such that $s =_\alpha t$ we have $T_k(s) = T_k(t)$ for any label of variables k.

Proof. By induction on the derivation of $s =_\alpha t$.

- Base cases. If $s = t$ then the result holds trivially, so suppose $s =_\alpha t$ and the conversion takes place at the root. Then $s = \xi x.(s_1, \dots, s_n) =_\alpha \xi y.(s_1\{x \leftarrow y\}, \dots, s_n\{x \leftarrow y\}) = t$ where y is a variable not occurring in $s_1, \dots s_n$. Then by Lemma 6.47 we have $T_k(s) = \xi(T_{xk}(s_1), \dots, T_{xk}(s_n)) = \xi(T_{yk}(s_1\{x \leftarrow y\}), \dots, T_{yk}(s_n\{x \leftarrow y\})) = T_k(t)$.
- Inductive cases:
  - $s =_\alpha t$ follows from $t =_\alpha s$. We use the induction hypothesis.
  - $s =_\alpha t$ follows from $s =_\alpha s'$ and $s' =_\alpha t$. We use the induction hypothesis.
  - the conversion is internal. Then two further cases are considered:
    * $s = f(s_1, \dots, s_i, \dots, s_n)$ and $t = f(s_1, \dots, s'_i, \dots, s_n)$ where $s_i =_\alpha s'_i$. We conclude by using the induction hypothesis.
    * $s = \xi x.(s_1, \dots, s_i, \dots, s_n)$ and $t = \xi x.(s_1, \dots, s'_i, \dots, s_n)$ where $s_i =_\alpha s'_i$. Then we have $T_k(s) = \xi(T_{xk}(s_1), \dots, T_{xk}(s_i), \dots, T_{xk}(s_n)) =_{i.h.} \xi(T_{xk}(s_1), \dots, T_{xk}(s'_i), \dots, T_{xk}(s_n)) = T_k(t)$.

∎

We now consider a translation from pre-metaterms to de Bruijn pre-metaterms. We shall also use the letter T for this translation in an attempt to avoid having to introduce yet another symbol.

Definition 6.49 (From pre-metaterms to de Bruijn pre-metaterms) A pre-metaterm M is translated as $T(M)$, where $T(M)$ is defined as $T_\epsilon(M)$ where $T_k(M)$ is defined by

$$\begin{array}{lcllcl} T_k(\alpha) & \stackrel{\text{def}}{=} & \mathsf{pos}(\alpha, k), \text{ if } \alpha \in k & T_k(f(M_1, \dots, M_n)) & \stackrel{\text{def}}{=} & f(T_k(M_1), \dots, T_k(M_n)) \\ T_k(\widehat{\alpha}) & \stackrel{\text{def}}{=} & \mathsf{S}^{|k|}(\widehat{\alpha}) & T_k(\xi\alpha.(M_1, \dots, M_n)) & \stackrel{\text{def}}{=} & \xi(T_{\alpha k}(M_1), \dots, T_{\alpha k}(M_n)) \\ T_k(X) & \stackrel{\text{def}}{=} & X_k & T_k(M_1[\alpha \leftarrow M_2]) & \stackrel{\text{def}}{=} & T_{\alpha k}(M_1)[\![T_k(M_2)]\!] \end{array}$$

Note that if M is a metaterm, then $T(M)$ will be defined and will only have t-metavariables with simple labels. Note also that, for some pre-metaterms, such as $\xi\alpha.\beta$, the translation $T(\bullet)$ is not defined. Moreover, if M is a metaterm then $T(M)$ is a de Bruijn metaterm.

Lemma 6.50 (T preserves well-formedness) If M is a metaterm, then $T(M)$ is a de Bruijn metaterm.

Proof. We need to prove a more general result: let M be a pre-metaterm, if $\mathcal{WF}_l(M)$, then $\mathcal{WF}_l(T_l(M))$. This is proved by induction on M.

- $M = \alpha$. Then $\alpha \in l$ and we have $\mathcal{WF}_l(T_l(\alpha))$ iff $\mathcal{WF}_l(\mathsf{pos}(\alpha, l))$. And the latter holds trivially.
- $M = \widehat{\alpha}$. Then $\mathcal{WF}_l(T_l(\widehat{\alpha}))$ iff $\mathcal{WF}_l(\mathsf{S}^{|l|}(\widehat{\alpha}))$. And the latter holds trivially.
- $M = X$. Then we have $\mathcal{WF}_l(T_l(X))$ iff $\mathcal{WF}_l(X_l)$ and l is simple. And the latter holds trivially.
- $M = f(M_1, \dots, M_n)$. By induction hypothesis.

- $M = \xi\alpha.(M_1, \ldots, M_n)$. Then $\mathcal{WF}_l(T_l(\xi\alpha.(M_1, \ldots, M_n)))$ iff $\mathcal{WF}_{\beta l}(T_{\alpha l}(M_i))$ with $1 \leq i \leq n$ for some $\beta \notin l$. Since $\mathcal{WF}_l(M)$ we have $\alpha \notin l$ and $\mathcal{WF}_{\alpha l}(M_i)$, so that we may take $\beta = \alpha$ and conclude by the induction hypothesis.

- $M = M_1[\alpha \leftarrow M_2]$. Similar to the previous case.

∎

Example 6.51 Let $M = \xi\alpha.(X, \lambda\beta.(Y, \alpha))$, $M' = f(\widehat{\beta}, \lambda\alpha.(Y, \widehat{\alpha}))$ and $M'' = g(\lambda\alpha.(\xi\beta.(h)))$. Then their respective translations are $A = \xi(X_\alpha, \lambda(Y_{\beta\alpha}, \mathtt{S}(1)))$, $A' = f(\widehat{\beta}, \lambda(Y_\alpha, \mathtt{S}(\widehat{\alpha})))$ and $A'' = g(\lambda(\xi(h)))$, which are metaterms as remarked in Example 6.29.

Definition 6.52 (From *SERS* rewrite rules to *SERS*$_{db}$ rewrite rules) Let (G, D) be a rewrite rule in the *SERS* formalism. Then $T(G, D)$ denotes the translation of the rewrite rule, defined as $(T(G), T(D))$.

As an immediate consequence of Lemma 6.50 and Def. 6.52, if (G, D) is an *SERS* rewrite rule, then $T(G, D)$ is an *SERS*$_{db}$ rewrite rule.

Example 6.53 (λx continued) Following Example 6.16, the specification of λx in the *SERS*$_{db}$ formalism is given below. It results from translating the rewrite rules of Example 6.16.

$$\begin{array}{lll}
app(\lambda X_\alpha, Z_\epsilon) & \longrightarrow & subs(\sigma X_\alpha, Z_\epsilon) \\
subs(\sigma(app(X_\alpha, Y_\alpha)), Z_\epsilon) & \longrightarrow & app(subs(\sigma X_\alpha, Z_\epsilon), subs(\sigma Y_\alpha, Z_\epsilon)) \\
subs(\sigma(\lambda(X_{\beta\alpha})), Z_\epsilon) & \longrightarrow & \lambda(subs(\sigma(X_{\alpha\beta}), Z_\beta)) \\
subs(\sigma(1), Z_\epsilon) & \longrightarrow & Z_\epsilon \\
subs(\sigma(\mathtt{S}(\widehat{\beta})), Z_\epsilon) & \longrightarrow & \widehat{\beta}
\end{array}$$

The rule $subs(\sigma(\lambda(X_{\beta\alpha})), Z_\epsilon) \longrightarrow \lambda(subs(\sigma(X_{\alpha\beta}), Z_\beta))$ is interesting since it illustrates the use of binder commutation from $X_{\beta\alpha}$ to $X_{\alpha\beta}$ and shows how some index adjustment shall be necessary when going from Z_β to Z_ϵ.

Example 6.54 (The $\lambda\Delta$-calculus continued) The translation of the $\lambda\Delta$-calculus (Example 6.17) yields the following rewrite rules in the *SERS*$_{db}$ formalism:

$$\begin{array}{lll}
app(\lambda(X_\alpha), Z_\epsilon) & \longrightarrow_{\beta_{db}} & X_\alpha[\![Z_\epsilon]\!] \\
app(\Delta(X_\alpha), Z_\epsilon) & \longrightarrow_{\Delta_1} & \Delta(X_{\alpha\beta}[\![\lambda(app(\mathtt{S}(1), app(1, Z_{\gamma\beta})))]\!]) \\
\Delta(app(1, X_\alpha)) & \longrightarrow_{\Delta_2} & X_\epsilon \\
\Delta(app(1, (\Delta(app(\mathtt{S}(1), X_{\beta\alpha}))))) & \longrightarrow_{\Delta_3} & X_\epsilon
\end{array}$$

We remark that the translation of Δ_1, Δ_2 and Δ_3 would not be possible in *XRS*.

Suppose some rewrite rule (L, R) is used to rewrite a term s. Then $s =_\alpha C[\theta(L)]$ for some context C and admissible valuation θ. When encoding this rewrite step in the *SERS*$_{db}$ setting we shall have to encode not only terms and metaterms, but also the valuation θ. Def. 6.55 below shows how one may encode valuations. This definition is parametrized over a label k, an issue which we would like to clarify. Suppose the metavariable X_l occurs in L, then when θ instantiates X_l the status of any variable x in the resulting term, $\theta(X_l)$, can be of one of four clases:

- either, x is bound in $\theta(X_l)$,

- or, x is free in $\theta(X_l)$ but is bound by some binder above X_l in the rewrite rule, in other words, there is a binder indicator $\alpha \in l$ such that $\theta(\alpha) = x$,

- or, x is free in $\theta(X_l)$ and x is not bound by the binders above X_l in the rule, i.e. $x \notin \theta(l)$, but x is bound by a binder above the □ in the context C,

- or, x is free in $\theta(X_l)$, it is not bound by the binders above X_l in the rewrite rule, and it is not bound by a binder in the context C above the □. Thus x is free in s.

Therefore, when translating a valuation to the $SERS_{db}$ setting we need to know what the names of the variables of the binders above the □ are. For example, if the third case holds then when translating to indices we must assign x an index which avoids being captured in the context. This is the role of the label k in the following definition.

Definition 6.55 (From valuations to de Bruijn valuations) Let θ be a valuation and k be a label of variables. Then the translation of θ w.r.t the label k (referred to as the context label) is defined as the de Bruijn valuation:

$$\begin{array}{lcl} T_k(\theta)(X_l) & \stackrel{\text{def}}{=} & T_{\theta(l)k}(\theta(X)) \text{ if } \theta(l) \text{ is defined} \\ T_k(\theta)(\widehat{\alpha}) & \stackrel{\text{def}}{=} & T_k(\theta(\widehat{\alpha})) \end{array}$$

where $X, \widehat{\alpha} \in Dom(\theta)$.

6.3.2 The Rewrite-Preservation Phase

In this section we study the rewrite-preservation phase, that is, we show that the translations of the definition phase ensure that the notion of rewriting in the formalism with de Bruijn indices has the same semantics as the corresponding one with names.

The following lemmas will be used in the proof of the main result of this section, namely Proposition 6.62, which states that $SERS$-rewriting may be simulated as $SERS_{db}$-rewriting.

Lemma 6.56 Let $s \in \mathcal{T}$ and $a \in \mathcal{T}_{db}$, let x be a variable and l, k be labels of variables with $|l| = i - 1$ and $x \in l$. Then $T_{lk}(s) = T_{lxk}(s)\{\!\{i \leftarrow a\}\!\}$.

Proof. By induction on s. We shall consider the case that s is a variable, the others follow from the induction hypothesis.

- $s = x$. Then $T_{lk}(x) = \mathrm{pos}(x, lk) = \mathrm{pos}(x, lxk) = \mathrm{pos}(x, lxk)\{\!\{i \leftarrow a\}\!\}$. The last equality holds since $\mathrm{pos}(x, lxk) < i$.
- $s = y \neq x$. We have three cases to consider:
  - $y \in l$. Then $T_{lk}(y) = \mathrm{pos}(y, lk) = \mathrm{pos}(y, lxk) = \mathrm{pos}(y, lxk)\{\!\{i \leftarrow a\}\!\}$. The last equality holds since $\mathrm{pos}(y, lxk) < i$.
  - $y \notin l$ and $y \in k$. Then $T_{lk}(y) = \mathrm{pos}(y, lk) = \mathrm{pos}(y, lxk) - 1 = \mathrm{pos}(y, lxk)\{\!\{i \leftarrow a\}\!\}$. The last equality holds since $\mathrm{pos}(y, lxk) > i$.
  - $y \notin lk$. Then $T_{lk}(y) = \mathcal{O}(y) + |lk| = \mathcal{O}(y) + |lxk| - 1 = T_{lxk}(y)\{\!\{i \leftarrow a\}\!\}$.

■

Lemma 6.57 Let $s \in \mathcal{T}$, let l_1, l_2, k be labels of variables such that $|l_1| = j$, $|l_2| = i - 1$ and $FV(s) \cap l_2 = \emptyset$. Then $T_{l_1 l_2 k}(s) = \mathcal{U}^i_j(T_{l_1 k}(s))$.

Proof. By induction on s. We shall consider the case that s is a variable, the others follow from the induction hypothesis. Suppose $s = x$. Note that since by hypothesis $x \notin l_2$ we have three cases to consider:

- $x \in l_1$. Then $T_{l_1 l_2 k}(x) = \mathrm{pos}(x, l_1 l_2 k) = \mathrm{pos}(x, l_1 k) = \mathcal{U}^i_j(\mathrm{pos}(x, l_1 k))$. The last equality holds since $\mathrm{pos}(x, l_1 k) \leq j$.
- $x \notin l_1$ and $x \in k$. Then $T_{l_1 l_2 k}(x) = \mathrm{pos}(x, l_1 l_2 k) = \mathrm{pos}(x, l_1 k) + i - 1 = \mathcal{U}^i_j(\mathrm{pos}(x, l_1 k))$.
- $x \notin l_1 k$. Then $T_{l_1 l_2 k}(x) = \mathcal{O}(x) + |l_1 l_2 k| = \mathcal{O}(x) + |l_1 k| + i - 1 = \mathcal{U}^i_j(T_{l_1 k}(x))$.

■

Lemma 6.58 Let $s, t \in \mathcal{T}$, l, k be labels of variables with $|l| = i - 1$, let x be a variable such that $x \notin l$, and suppose $FV(t) \cap l = \emptyset$. If $s\{x \leftarrow t\}$ is defined then $T_{lk}(s\{x \leftarrow t\}) = T_{lxk}(s)\{\!\{i \leftarrow T_k(t)\}\!\}$.

Proof. By induction on s.

- $s = x$. Then we have:

$$\begin{array}{rcll} T_{lk}(t) & = & \mathcal{U}_0^i(T_k(t)) & \text{(L.6.57)} \\ & = & i\{\!\{i \leftarrow T_k(t)\}\!\} & \\ & = & T_{lxk}(s)\{\!\{i \leftarrow T_k(t)\}\!\} & (x \notin l) \end{array}$$

- $s = y \neq x$. Then we have three subcases to consider:
  - $y \in l$. Then $T_{lk}(y) = \mathtt{pos}(y, lk) = \mathtt{pos}(y, lxk) = T_{lxk}(y) = T_{lxk}(y)\{\!\{i \leftarrow T_k(t)\}\!\}$. The last equality holds because $\mathtt{pos}(y, l) < i$.
  - $y \notin l$ and $y \in k$. Then $T_{lk}(y) = \mathtt{pos}(y, lk) = \mathtt{pos}(y, lxk) - 1 = T_{lxk}(y) - 1 = T_{lxk}(y)\{\!\{i \leftarrow T_k(t)\}\!\}$. The last equality holds because $\mathtt{pos}(y, lxk) > i$.
  - $y \notin lk$. Then $T_{lk}(y) = \mathcal{O}(y) + |lk| = \mathcal{O}(y) + |lxk| - 1 = T_{lxk}(y) - 1 = T_{lxk}(y)\{\!\{i \leftarrow T_k(t)\}\!\}$. The last equality holds because $\mathcal{O}(y) + |lxk| > i$.

- $s = f(s_1, \ldots, s_n)$. The we have

$$\begin{array}{rl} & T_{lk}(s\{x \leftarrow t\}) \\ = & f(T_{lk}(s_1\{x \leftarrow t\}), \ldots, T_{lk}(s_n\{x \leftarrow t\})) \\ =_{i.h.} & f(T_{lxk}(s_1)\{\!\{i \leftarrow T_k(t)\}\!\}, \ldots, T_{lxk}(s_n)\{\!\{i \leftarrow T_k(t)\}\!\}) \\ = & T_{lxk}(s)\{\!\{i \leftarrow T_k(t)\}\!\} \end{array}$$

- $s = \xi y.(s_1, \ldots, s_n)$. Note that since $s\{x \leftarrow t\}$ is defined by hypothesis we know that $y \notin FV(t)$ for otherwise $s\{x \leftarrow t\}$ would not be defined. We consider two further cases:
  - $y \neq x$. Then we have

$$\begin{array}{rcl} T_{lk}(s\{x \leftarrow t\}) & = & \xi(T_{ylk}(s_1\{x \leftarrow t\}, \ldots, s_n\{x \leftarrow t\})) \\ & =_{i.h.} & \xi(T_{ylxk}(s_1)\{\!\{i+1 \leftarrow T_k(t)\}\!\}, \ldots, T_{ylxk}(s_n)\{\!\{i+1 \leftarrow T_k(t)\}\!\}) \\ & = & T_{lxk}(s)\{\!\{i \leftarrow T_k(t)\}\!\} \end{array}$$

  - $y = x$. Then we have

$$\begin{array}{rcll} T_{lk}(s\{x \leftarrow t\}) & = & T_{lk}(\xi x.(s_1, \ldots, s_n)) & \\ & = & \xi(T_{xlk}(s_1), \ldots, T_{xlk}(s_n)) & \\ & = & \xi(T_{xlxk}(s_1)\{\!\{i+1 \leftarrow T_k(t)\}\!\}, \ldots, T_{xlxk}(s_n)\{\!\{i+1 \leftarrow T_k(t)\}\!\}) & \text{(L.6.56)} \\ & = & T_{lxk}(\xi x.(s_1, \ldots, s_n))\{\!\{i \leftarrow T_k(t)\}\!\} & \end{array}$$

■

As expected, the translation is well-behaved w.r.t contexts and valuations. We take the opportunity to recall the reader that the notion of a label of a context is given in Def. 6.8. Induction on the context C in the following result may be used for proving it.

Lemma 6.59 **(T is modular w.r.t contexts)** Let C be a context, l the label of C and $t \in \mathcal{T}$. Then for every label k we have, $T_k(C[t]) = T_k(C)[T_{lk}(t)]$.

Lemma 6.60 **(T is modular w.r.t. valuations)** Let M be a pre-metaterm, l a label of binder indicators, and suppose

1. $\mathcal{WF}_l(M)$,
2. $\theta = (\theta_v, \theta_t)$ is a valuation such that θ_v is injective on the bound o-metavariables, and
3. θ is safe for M.

Then for every label k we have $T_{\theta(l)k}(\theta M) = T_k(\theta)(T_l(M))$.

Proof. By induction on the pre-metaterm M. Since $\mathcal{WF}_l(M)$ we have the following cases to consider:

- $M = \alpha \in l$. Then $T_{\theta(l)k}(\theta\alpha) = \mathtt{pos}(\theta\alpha, \theta(l)) =_{hyp.2} \mathtt{pos}(\alpha, l) = T_k(\theta)(\mathtt{pos}(\alpha, l)) = T_k(\theta)(T_l(\alpha))$.

- $M = \widehat{\alpha}$. Then by the definition of valuation and since $\widehat{\alpha} \notin l$ because l is a label of binder indicators:

$$\begin{aligned} T_{\theta(l)k}(\theta\widehat{\alpha}) &= \begin{cases} \mathtt{pos}(\theta\widehat{\alpha}, k) + |\theta(l)| & \text{if } \theta\widehat{\alpha} \in k \\ \mathcal{O}(\theta\widehat{\alpha}) + |\theta(l)k| & \text{otherwise.} \end{cases} \\ &= \begin{cases} \mathtt{S}^{|l|}(\mathtt{pos}(\theta\widehat{\alpha}, k)) & \text{if } \theta\widehat{\alpha} \in k \\ \mathtt{S}^{|l|}(\mathcal{O}(\theta\widehat{\alpha}) + |k|) & \text{otherwise.} \end{cases} \\ &= T_k(\theta)(\mathtt{S}^{|l|}(\widehat{\alpha})) \\ &= T_k(\theta)(T_l(\widehat{\alpha})) \end{aligned}$$

- $M = X$. Then $T_{\theta(l)k}(\theta X) = T_k(\theta)(X_l) = T_k(\theta)(T_l(X))$.

- $M = f(M_1, \ldots, M_n)$. Then we have

$$\begin{aligned} T_{\theta(l)k}(\theta f(M_1, \ldots, M_n)) &= f(T_{\theta(l)k}(\theta M_1), \ldots, T_{\theta(l)k}(\theta M_n)) \\ &=_{i.h.} f(T_k(\theta)(T_l(M_1)), \ldots, T_k(\theta)(T_l(M_n))) \\ &= T_k(\theta)(T_l(f(M_1, \ldots, M_n))) \end{aligned}$$

- $M = \xi\alpha.(M_1, \ldots, M_n)$. Then we have

$$\begin{aligned} T_{\theta(l)k}(\theta\xi\alpha.(M_1, \ldots, M_n)) &= \xi(T_{\theta(\alpha)\theta(l)k}(\theta M_1), \ldots, T_{\theta(\alpha)\theta(l)k}(\theta M_n)) \\ &=_{i.h.} \xi(T_k(\theta)(T_{\alpha l}(M_1)), \ldots, T_k(\theta)(T_{\alpha l}(M_n))) \\ &= T_k(\theta)(T_l(\xi\alpha.(M_1, \ldots, M_n))) \end{aligned}$$

- $M = M_1[\alpha \leftarrow M_2]$. Then we have

$$\begin{aligned} T_{\theta(l)k}(\theta(M_1[\alpha \leftarrow M_2])) &= T_{\theta(l)k}(\theta M_1\{\theta_v(\alpha) \leftarrow \theta M_2\}) \\ &=_{L.\ 6.58} T_{\theta(\alpha)\theta(l)k}(\theta M_1)\{\!\{1 \leftarrow T_{\theta(l)k}(\theta M_2)\}\!\} \\ &=_{i.h.} T_k(\theta)(T_{\alpha l}(M_1))\{\!\{1 \leftarrow T_k(\theta)(T_l(M_2))\}\!\} \\ &= T_k(\theta)(T_{\alpha l}(M_1)[\![T_l(M_2)]\!]) \\ &= T_k(\theta)(T_l(M_1[\alpha \leftarrow M_2])) \end{aligned}$$

  Note that since θ is safe for M we may apply Lemma 6.58 above with $l = \epsilon$. Indeed, $\theta_v(\alpha) \notin \theta(l)$ and $FV(\theta M_2) \cap \theta(l) = \emptyset$ for $l = \epsilon$.

■

Lemma 6.61 Let k, k' be labels of binder indicators, l a label of variables and θ be an injective function on the set of binder indicators. Then for every $t \in \mathcal{T}$, every $p \geq 0$, every $x_1, \ldots, x_p$, if for every $z \in FV(t) \setminus \{x_1, \ldots, x_p\}$ we have $z \in \theta(k)$ iff $z \in \theta(k')$, then

$$Value^p(k, T_{x_1 \ldots x_p \theta(k) l}(t)) = Value^p(k', T_{x_1 \ldots x_p \theta(k') l}(t))$$

Proof. We use induction on t.

- $t = x$. We have the following further cases to consider:
  - $x = x_i$ with $1 \leq i \leq p$. Then

    $$Value^p(k, T_{x_1 \ldots x_p \theta(k) l}(x)) = Value^p(k, i) = i = Value^p(k', i) = Value^p(k', T_{x_1 \ldots x_p \theta(k') l}(x))$$

  - $x \in \theta(k) \cap \theta(k')$ and the previous case does not hold. Let $i = \mathtt{pos}(x, \theta(k))$ and $j = \mathtt{pos}(x, \theta(k'))$. Then $\mathtt{at}(k, i) = \mathtt{at}(k', j)$ by injectivity of θ and we have

    $$\begin{aligned} Value^p(k, T_{x_1 \ldots x_p \theta(k) l}(x)) &= Value^p(k, p + i) \\ &= \mathtt{at}(k, i) \\ &= \mathtt{at}(k', j) \\ &= Value^p(k', p + j) \\ &= Value^p(k', T_{x_1 \ldots x_p \theta(k') l}(x)) \end{aligned}$$

- $x \in l$ and the previous cases do not hold. Then for some i with $1 \leq i \leq |l|$ we have

$$\begin{array}{rcl} Value^p(k, T_{x_1 \ldots x_p \theta(k) l}(x)) & = & Value^p(k, p + |\theta(k)| + i) \\ & = & x_i \\ & = & Value^p(k', p + |\theta(k')| + i) \\ & = & Value^p(k', T_{x_1 \ldots x_p \theta(k') l}(x)) \end{array}$$

- $x \notin \{x_1, \ldots, x_p\}$, $x \notin \theta(k) \cup \theta(k')$, $x \notin l$. Then we have

$$\begin{array}{rcl} Value^p(k, T_{x_1 \ldots x_p \theta(k) l}(x)) & = & Value^p(k, \mathcal{O}(x) + p + |\theta(k) l|) \\ & = & x_{\mathcal{O}(x)+|l|} \\ & = & Value^p(k', \mathcal{O}(x) + p + |\theta(k') l|) \\ & = & Value^p(k', T_{x_1 \ldots x_p \theta(k') l}(x)) \end{array}$$

- $t = f(t_1, \ldots, t_n)$. By induction hypothesis we have $Value^p(k, T_{x_1 \ldots x_p \theta(k) l}(t_i)) = Value^p(k', T_{x_1 \ldots x_p \theta(k') l}(t_i))$ so the property trivially holds.
- $t = \xi x.(t_1, \ldots, t_n)$. Then we have

$$\begin{array}{rl} & Value^p(k, T_{x_1 \ldots x_p \theta(k) l}(\xi x.(t_1, \ldots, t_n))) \\ = & Value^p(k, \xi(T_{x x_1 \ldots x_p \theta(k) l}(t_1), \ldots, T_{x x_1 \ldots x_p \theta(k) l}(t_n))) \\ = & \xi(Value^{p+1}(k, T_{x x_1 \ldots x_p \theta(k) l}(t_1)), \ldots, Value^{p+1}(k, T_{x x_1 \ldots x_p \theta(k) l}(t_n))) \\ =_{i.h.} & \xi(Value^{p+1}(k', T_{x x_1 \ldots x_p \theta(k') l}(t_1)), \ldots, Value^{p+1}(k', T_{x x_1 \ldots x_p \theta(k') l}(t_n))) \\ = & Value^p(k', T_{x_1 \ldots x_p \theta(k') l}(\xi x.(t_1, \ldots, t_n))) \end{array}$$

■

We can finally conclude with the main result of this section which ensures that the $SERS_{db}$ formalism preserves $SERS$-rewriting.

Proposition 6.62 (Simulating $SERS$-rewriting via $SERS_{db}$-rewriting) Suppose $s \rightarrow t$ in the $SERS$ formalism using the rewrite rule (G, D). Then $T(s) \rightarrow T(t)$ in the $SERS_{db}$ formalism using the de Bruijn rewrite rule $T(G, D)$.

Proof. By definition of the rewrite relation (Def. 6.23) there is an admissible valuation θ for (G, D) and there is a context C such that $s =_\alpha C[\theta G]$ and $t =_\alpha C[\theta D]$. By Lemma 6.48 $T(s) = T(C[\theta G])$ and $T(t) = T(C[\theta D])$. Note that $T(G, D) = (T(G), T(D))$ is a de Bruijn rewrite rule by Lemma 6.50. The proof thus proceeds in two steps: in Step 1 we show that there exists a de Bruijn valuation κ and a de Bruijn context E such that $T(s) = E[\kappa T(G)]$ and $T(t) = E[\kappa T(D)]$; Step 2 consists in showing that κ is a valid de Bruijn valuation for $(T(G), T(D))$.

- Step 1. By Lemma 6.59 we have $T(s) = T(C)[T_k(\theta G)]$ and $T(t) = T(C)[T_k(\theta D)]$, where k is the label of the context C, and $T(C)$ is a de Bruijn context. By hypothesis G is well-formed and θ is safe for G (so that θ_v is injective on the set of bound o-metavariables). As a consequence we can apply Lemma 6.60 so that $T_k(\theta G) = T_k(\theta)(T(G))$ and $T_k(\theta D) = T_k(\theta)(T(D))$, where $T_k(\theta)$ is a de Bruijn valuation. Thus we may take $\kappa \overset{\text{def}}{=} T_k(\theta)$ and $E \overset{\text{def}}{=} T(C)$.
- Step 2. We have still to show that $T_k(\theta)$ is valid for $(T(G), T(D))$. By Def. 6.41 we have to check that $Value(l, T_k(\theta)(X_l)) = Value(l', T_k(\theta)(X_{l'}))$ for every pair of t-metavariables X_l and $X_{l'}$ appearing in the de Bruijn rewrite rule $(T(G), T(D))$, that is, by Def. 6.55, that $Value(l, T_{\theta(l)k}(\theta(X))) = Value(l', T_{\theta(l')k}(\theta(X)))$. Finally, verifying the following conditions allows us to conclude from Lemma 6.61 with $p = 0$:
  - $\theta(X)$ is a term in $\mathcal{T}$ by definition of valuations.
  - θ is injective on bound o-metavariables since it is admissible,
  - finally, we need to show that for every variable $z \in FV(\theta X)$ we have $z \in \theta(l)$ iff $z \in \theta(l')$. But this immediately follows from the fact that θ verifies the path condition for X in (G, D) because it is admissible.

6.4 ...And Back

In this section we show that *SERS* are operationally equivalent to $SERS_{db}$. For that, we show how the notion of rewriting in the $SERS_{db}$ formalism may be simulated in the *SERS*. As in Section 6.3 we shall develop the required results by distinguishing the *definition phase* and the *rewrite-preservation phase*.

6.4.1 The Definition Phase

In this section we study the definition phase, that is, we define translations from the $SERS_{db}$ to the *SERS* formalism. We shall begin with a translation from de Bruijn terms to terms with variable names. This shall make use of the $\mathtt{Names}(\bullet)$ function given below Def. 6.33.

Definition 6.63 (From de Bruijn terms (and contexts) to terms (and contexts)) The translation of $a \in \mathcal{T}_{db}$, denoted $U(a)$, is defined as $U_{\epsilon}^{\mathtt{Names}(FV(a))}(a)$ where, for every finite set of variables S, and every label of variables k, $U_k^S(a)$ is defined as follows:

$$U_k^S(n) \quad \overset{\text{def}}{=} \begin{cases} \mathtt{at}(k,n) & \text{if } n \leq |k| \\ x_{n-|k|} & \text{if } n > |k| \text{ and } x_{n-|k|} \in S \end{cases}$$

$$\begin{array}{l} U_k^S(f(a_1,\ldots,a_n)) \overset{\text{def}}{=} f(U_k^S(a_1),\ldots,U_k^S(a_n)) \\ U_k^S(\xi(a_1,\ldots,a_n)) \overset{\text{def}}{=} \xi x.(U_{xk}^S(a_1),\ldots,U_{xk}^S(a_n)) \;\; \text{for any } x \notin k \cup S \end{array}$$

The translation of a de Bruijn context E, denoted $U(E)$, is defined as above but adding the clause $U_k^S(\Box) \overset{\text{def}}{=} \Box$. We remark that we can always choose $x \notin k \cup S$ since both k and S are finite.

Note that $U(\bullet)$ is not a function in the sense that the choice of bound variables is non-deterministic. However, one can show that if t and t' belong both to $U(a)$, then $t =_\alpha t'$. Thus, $U(\bullet)$ can be seen as a function from de Bruijn terms to α-equivalence classes.

We remark that given a set of variables S, a de Bruijn term a and a label k, the translation $U_k^S(a)$ is always defined if $\mathtt{Names}(FV(a)\backslash\!\!\backslash|k|) \subseteq S$. It is quite evident that $FV(\xi(a))\backslash\!\!\backslash n$ is exactly $FV(a)\backslash\!\!\backslash(n+1)$. Also, if $U_k^S(C[a])$ is defined and $|l|$ is the binder path number of C (see Def. 6.32), then $U_{lk}^S(a)$ is also defined. Note, moreover, that if $x \in FV(U_k^S(a))$ then $x \in S \cup k$. .

Definition 6.64 (From de Bruijn pre-metaterms to pre-metaterms) The translation of the de Bruijn pre-metaterm A, denoted $U(A)$, is defined as $U_\epsilon(A)$, where $U_l(A)$ is defined as follows:

$$\begin{array}{lll} U_l(\mathtt{S}^i(1)) & \overset{\text{def}}{=} \mathtt{at}(l,i+1) & \text{if } i+1 \leq |l| \\ U_l(\mathtt{S}^{|l|}(\widehat{\alpha})) & \overset{\text{def}}{=} \widehat{\alpha} & \\ U_l(X_l) & \overset{\text{def}}{=} X & \\ U_l(f(A_1,\ldots,A_n)) & \overset{\text{def}}{=} f(U_l(A_1),\ldots,U_l(A_n)) & \\ U_l(\xi(A_1,\ldots,A_n)) & \overset{\text{def}}{=} \xi\alpha.(U_{\alpha l}(A_1),\ldots,U_{\alpha l}(A_n)) & \\ & \quad \text{if } 1 \leq i \leq n \;\; \mathcal{WF}_{\alpha l}(A_i) \text{ for some } \alpha \notin l & \\ U_l(A_1[\![A_2]\!]) & \overset{\text{def}}{=} U_{\alpha l}(A_1)[\alpha \leftarrow U_l(A_2)] & \\ & \quad \text{if } \mathcal{WF}_{\alpha l}(A_1) \text{ for some } \alpha \notin l & \end{array}$$

As in Def. 6.63 we remark that the translation of a de Bruijn pre-metaterm is not a function since it depends on the choice of the names for o-metavariables. Indeed, two different pre-metaterms obtained by this translation will be v-equivalent. Also, for some de Bruijn pre-metaterms such as $\xi(2)$, the translation may not be defined. However, it is defined on de Bruijn metaterms.

Definition 6.65 (From $SERS_{db}$ rewrite rules to *SERS* rewrite rules) Let (L,R) be a de Bruijn rewrite rule then its translation, denoted $U(L,R)$, is the pair of metaterms $(U(L),U(R))$.

Note that if A is such that $\mathcal{WF}_l(A)$ holds then its translation $U_l(A)$ is also a named metaterm, that is, $\mathcal{WF}_l(U_l(A))$ also holds. Therefore, by Def. 6.13 the translation of a de Bruijn rewrite rule is a rewrite rule in the *SERS* formalism. As mentioned above, if a de Bruijn pre-metaterm A is not a de Bruijn metaterm then $U_l(A)$ may not be defined.

Example 6.66 Consider the de Bruijn rule $app(\Delta X_\alpha, Z_\epsilon) \to \Delta(X_{\alpha\beta}[\![\lambda(app(\mathtt{S}(1), app(1, Z_{\gamma\beta})))]\!])$ from Example 6.54. The rule obtained by the translation of Def. 6.64 is

$$app(\Delta\alpha.X, Z) \to \Delta\beta.(X[\alpha \leftarrow \lambda\gamma.(app(\beta, app(\gamma, Z)))])$$

Whereas, for the rule $subs(\sigma(\mathtt{S}(\widehat{\beta})), Z_\epsilon) \to \widehat{\beta}$ we obtain $subs(\sigma\gamma.(\widehat{\beta}), Z) \to \widehat{\beta}$ for some bound o-metavariable γ.

Definition 6.67 (From de Bruijn valuations to valuations) Let $\kappa = (\kappa_i, \kappa_t)$ be a de Bruijn valuation, S be a finite set of variables and k a label of variables, and θ_v be a variable assignment such that:

1. $\theta_v(\alpha) \notin S \cup k$, for any $\alpha \in Dom(\theta_v)$, and
2. for every $\widehat{\alpha} \in Dom(\kappa_i)$, $\theta_v(\widehat{\alpha}) = \begin{cases} \mathtt{at}(k, \kappa_i(\widehat{\alpha})) & \text{if } \kappa_i(\widehat{\alpha}) \leq |k| \\ x_{\kappa_i(\widehat{\alpha})-|k|} & \text{otherw. with } x_{\kappa_i(\widehat{\alpha})-|k|} \in S \end{cases}$

The translation of κ is the valuation $U_{(\theta_v,S,k)}(\kappa) \overset{\text{def}}{=} (\theta_v, \theta_t)$, where

$$\theta_t X \overset{\text{def}}{=} U^S_{\theta_v(l)k}(\kappa X_l) \quad \text{for any } X_l \text{ in } Dom(\kappa)$$

Condition 2 on θ_v says that if an i-metavariable in A is bound (or free) in the context k as interpreted via κ then the new valuation $U_{(\theta_v,S,k)}(\kappa)$ must reflect this fact. We will now show that if κ is a valid de Bruijn valuation then this definition is *correct*, that is, the definition does not depend on the choice of the t-metavariable X_l in $Dom(\kappa)$. For that, we need some lemmas which are developed in the appendix (Section A.3.1).

Lemma 6.68 (Translation of de Bruijn valuations is correct) The valuation $U_{(\theta,S,k)}(\kappa)$ given in Def. 6.67 is correct if κ is valid, where correct means that for every X_l and $X_{l'}$ in $Dom(\kappa)$ we have $U^S_{\theta_v(l)k}(\kappa X_l) =_\alpha U^S_{\theta_v(l')k}(\kappa X_{l'})$, whenever both terms are defined.

Proof. Since κ is valid we have $Value(l, \kappa X_l) = Value(l', \kappa X_{l'})$ for every X_l and X'_l in $Dom(\kappa)$. Then by Lemma A.20 we may conclude $U^S_{\theta_v(l)k}(\kappa X_l) =_\alpha U^S_{\theta_v(l')k}(\kappa X_{l'})$. ▪

6.4.2 The Rewrite-Preservation Phase

In this section we study the rewrite-preservation phase, that is, we show that the translations of the definition phase ensure that the notion of rewriting in the formalism with names has the same semantics as the corresponding one with de Bruijn indices. More precisely, we seek to prove the following:

Proposition 6.73 (Simulating $SERS_{db}$-rewriting via $SERS$-rewriting) Assume $a \to b$ in the $SERS_{db}$ formalism using rewrite rule (L, R). Then $U(a) \to U(b)$ in the $SERS$ formalism using rule $U(L, R)$.

For that we need to develop some intermediate results. These results start with Lemma 6.69 and end with Lemma 6.72.

Lemma 6.69 (U is modular w.r.t de Bruijn contexts) Let E be a de Bruijn context, l, k labels where $|l|$ is the binder path number of E and $a \in \mathcal{T}_{db}$. If $U^S_k(E[a])$ and $U^S_{lk}(a)$ are defined, then $U^S_k(E[a]) =_\alpha U^S_k(E)[U^S_{lk}(a)]$.

Proof. By induction on the context E.

- $E = \Box$. Then $l = \epsilon$ and the result holds trivially.
- $E = f(a_1, \ldots, E', \ldots, a_n)$. Note that the binder path number of E and E' are the same. We reason as follows:

$$\begin{array}{rcll} U^S_k(f(a_1, \ldots, E'[a], \ldots, a_n)) & = & f(U^S_k(a_1), \ldots, U^S_k(E'[a]), \ldots, U^S_k(a_n)) & \\ & =_\alpha & f(U^S_k(a_1), \ldots, U^S_k(E')[U^S_{lk}(a)], \ldots, U^S_k(a_n)) & \text{(i.h.)} \\ & = & U^S_k(E)[U^S_{lk}(a)] & \end{array}$$

- $E = \xi(a_1, \ldots, E', \ldots, a_n)$. Since l is a label such that $|l|$ is the binder path number of E, we have $l = l'y$ for some label l' and variable y. We now reason as follows:

$$\begin{array}{lll}
 & U^S_k(\xi(a_1, \ldots, E'[a], \ldots, a_n)) & \\
= & \xi x.(U^S_{xk}(a_1), \ldots, U^S_{xk}(E'[a]), \ldots, U^S_{xk}(a_n)) & \\
=_\alpha & \xi z.(U^S_{xk}(a_1)\{x \leftarrow z\}, \ldots, U^S_{xk}(E'[a])\{x \leftarrow z\}, \ldots, U^S_{xk}(a_n)\{x \leftarrow z\}) & (z \text{ fresh}) \\
=_\alpha & \xi z.(U^S_{zk}(a_1), \ldots, U^S_{zk}(E'[a]), \ldots, U^S_{zk}(a_n)) & (\text{L.}A.18) \\
=_\alpha & \xi z.(U^S_{yk}(a_1)\{y \leftarrow z\}, \ldots, U^S_{yk}(E'[a])\{y \leftarrow z\}, \ldots, U^S_{yk}(a_n)\{y \leftarrow z\}) & (\text{L.}A.18) \\
=_\alpha & \xi y.(U^S_{yk}(a_1), \ldots, U^S_{yk}(E'[a]), \ldots, U^S_{yk}(a_n)) & \\
=_\alpha & \xi y.(U^S_{yk}(a_1), \ldots, U^S_{yk}(E')[U^S_{l'yk}(a)], \ldots, U^S_{yk}(a_n)) & (\text{i.h.}) \\
= & U^S_k(E)[U^S_{lk}(a)] &
\end{array}$$

Note that $x \notin k$ and $y \notin k$ for otherwise they could not have been chosen by the $U^\bullet_\bullet(\bullet)$ translation mapping as candidate variables for binding.

$\bullet$

Lemma 6.70 Let $a \in \mathcal{T}_{db}$, l_1, l_2 and k be labels of variables with $|l_1| = j$ and $|l_2| = i-1$. Then $U^S_{l_1 l_2 k}(\mathcal{U}^i_j(a)) =_\alpha U^S_{l_1 k}(a)$ if $U^S_{l_1 k}(a)$ is defined.

Proof. We proceed by induction on a. The case $a = f(a_1, \ldots, a_n)$ holds by the induction hypothesis, so we consider the other ones.

- $a = n$. We have two cases to consider:
  - $n \leq j$. Then $U^S_{l_1 l_2 k}(\mathcal{U}^i_j(n)) = U^S_{l_1 l_2 k}(n) = \mathtt{at}(l_1 l_2 k, n) = \mathtt{at}(l_1 k, n) = U^S_{l_1 k}(n)$.
  - $n > j$. Then $U^S_{l_1 l_2 k}(\mathcal{U}^i_j(n)) = U^S_{l_1 l_2 k}(n + i - 1)$ and we have two further subcases to consider:
    - ∗ $n + i - 1 \leq |l_1 l_2 k|$. Then since $n \leq |l_1 k|$ we have $U^S_{l_1 l_2 k}(n + i - 1) = \mathtt{at}(l_1 l_2 k, n + i - 1) = \mathtt{at}(l_1 k, n) = U^S_{l_1 k}(n)$.
    - ∗ $n + i - 1 > |l_1 l_2 k|$. Then since $n > |l_1 k|$ we have $U^S_{l_1 l_2 k}(n + i - 1) = x_{n+i-1-|l_1 l_2 k|} = x_{n-|l_1 k|} = U^S_{l_1 k}(n)$.
- $a = \xi(a_1, \ldots, a_n)$. Then we reason as follows:

$$\begin{array}{llll}
U^S_{l_1 l_2 k}(\mathcal{U}^i_j(a)) & = & \xi z.(U^S_{z l_1 l_2 k}(\mathcal{U}^i_{j+1}(a_1)), \ldots, U^S_{z l_1 l_2 k}(\mathcal{U}^i_{j+1}(a_1))) & \\
 & =_\alpha & \xi z.(U^S_{z l_1 k}(a_1), \ldots, U^S_{z l_1 k}(a_n)) & (\text{i.h.}) \\
 & =_\alpha & \xi z'.(U^S_{z l_1 k}(a_1)\{z \leftarrow z'\}, \ldots, U^S_{z l_1 k}(a_n)\{z \leftarrow z'\}) & (z' \text{ fresh}) \\
 & =_\alpha & \xi z'.(U^S_{z' l_1 k}(a_1), \ldots, U^S_{z' l_1 k}(a_n)) & (\text{L.}A.18) \\
 & =_\alpha & \xi z'.(U^S_{y l_1 k}(a_1)\{y \leftarrow z'\}, \ldots, U^S_{y l_1 k}(a_n)\{y \leftarrow z'\}) & (\text{L.}A.18) \\
 & =_\alpha & \xi y.(U^S_{y l_1 k}(a_1), \ldots, U^S_{y l_1 k}(a_n)) & \\
 & = & U^S_{l_1 k}(a) &
\end{array}$$

The phrase "z' fresh" should be read, in full rigor, as "z' does not occur in $U^S_{z l_1 k}(a_i)$ nor in $U^S_{y l_1 k}(a_i)$ for $1 \leq i \leq n$". The definition of $U^\bullet_\bullet(\bullet)$ and the hypothesis that $U^S_{l_1 k}(a)$ is defined, allows us to apply Lemma A.18 above.

$\bullet$

Lemma 6.71 Let $a, b \in \mathcal{T}_{db}$, l and k labels of variables with $|l| = i - 1$, x a variable such that $x \notin lk \cup S$. Then $U^S_{lk}(a\{\!\{i \leftarrow b\}\!\}) =_\alpha U^S_{lxk}(a)\{x \leftarrow U^S_k(b)\}$, assuming both sides of the equation are defined.

Proof. The proof is by induction on a.

- $a = n$. We have three further cases to consider:

- $n < i$. Then we reason as follows:

$$\begin{array}{rcl} U^S_{lk}(a\{\!\{i \leftarrow b\}\!\}) & = & U^S_{lk}(n) \\ & = & \mathtt{at}(lk, n) \\ & = & \mathtt{at}(lxk, n) \\ & = & \mathtt{at}(lxk, n)\{x \leftarrow U^S_k(b)\} \quad (x \notin l) \\ & = & U^S_{lxk}(n)\{x \leftarrow U^S_k(b)\} \end{array}$$

- $n > i$. Then since $U^S_{lk}(a\{\!\{i \leftarrow b\}\!\}) = U^S_{lk}(n-1)$ we consider two further cases:
  * $n - 1 \leq |lk|$. We reason as follows:

$$\begin{array}{rcl} U^S_{lk}(n-1) & = & \mathtt{at}(lk, n-1) \\ & = & \mathtt{at}(lxk, n) \\ & = & \mathtt{at}(lxk, n)\{x \leftarrow U^S_k(b)\} \quad (x \notin k) \\ & = & U^S_{lxk}(n)\{x \leftarrow U^S_k(b)\} \end{array}$$

  * $n - 1 > |lk|$. We reason as follows:

$$\begin{array}{rcl} U^S_{lk}(n-1) & = & x_{n-1-|lk|} \\ & = & x_{n-|lxk|} \\ & = & x_{n-|lxk|}\{x \leftarrow U^S_k(b)\} \quad (x \notin S) \\ & = & U^S_{lxk}(n)\{x \leftarrow U^S_k(b)\} \end{array}$$

- $n = i$. Then on one hand we have $U^S_{lk}(i\{\!\{i \leftarrow b\}\!\}) = U^S_{lk}(\mathcal{U}^i_0(b))$. And on the other $U^S_{lxk}(i)\{x \leftarrow U^S_k(b)\} = x\{x \leftarrow U^S_k(b)\} = U^S_k(b)$. Lemma 6.70 concludes this case.

- $a = f(a_1, \ldots, a_n)$. We use the induction hypothesis.
- $a = \xi(a_1, \ldots, a_n)$. Then we reason as follows:

$$\begin{array}{lll} & U^S_{lk}(a\{\!\{i \leftarrow b\}\!\}) & \\ = & \xi z.(U^S_{zlk}(a_1\{\!\{i+1 \leftarrow b\}\!\}), \ldots, U^S_{zlk}(a_n\{\!\{i+1 \leftarrow b\}\!\})) & (z \notin lk \cup S \text{ by..} \\ & & \text{(..Def. 6.63)} \\ =_\alpha & \xi y.(U^S_{zlk}(a_1\{\!\{i+1 \leftarrow b\}\!\})\{z \leftarrow y\}, \ldots, U^S_{zlk}(a_n\{\!\{i+1 \leftarrow b\}\!\})\{z \leftarrow y\}) & (y \text{ fresh}) \\ =_\alpha & \xi y.(U^S_{ylk}(a_1\{\!\{i+1 \leftarrow b\}\!\}), \ldots, U^S_{ylk}(a_n\{\!\{i+1 \leftarrow b\}\!\})) & (\text{L.}A\text{.18}) \\ =_\alpha & \xi y.(U^S_{ylxk}(a_1)\{x \leftarrow U^S_k(b)\}, \ldots, U^S_{ylxk}(a_n)\{x \leftarrow U^S_k(b)\}) & (\text{i.h.}) \\ =_\alpha & \xi y.(U^S_{(z'lxk)\{z' \leftarrow y\}}(a_1)\{x \leftarrow U^S_k(b)\}, \ldots, U^S_{(z'lxk)\{z' \leftarrow y\}}(a_n)\{x \leftarrow U^S_k(b)\}) & (\text{see below}) \\ =_\alpha & \xi y.(U^S_{z'lxk}(a_1)\{z' \leftarrow y\}\{x \leftarrow U^S_k(b)\}, \ldots, U^S_{z'lxk}(a_n)\{z' \leftarrow y\}\{x \leftarrow U^S_k(b)\}) & (\text{L.}A\text{.18}) \\ =_\alpha & \xi y.(U^S_{z'lxk}(a_1)\{x \leftarrow U^S_k(b)\}\{z' \leftarrow y\}, \ldots, U^S_{z'lxk}(a_n)\{x \leftarrow U^S_k(b)\}\{z' \leftarrow y\}) & (\text{Subst.L.}) \\ =_\alpha & \xi z'.(U^S_{z'lxk}(a_1)\{x \leftarrow U^S_k(b)\}, \ldots, U^S_{z'lxk}(a_n)\{x \leftarrow U^S_k(b)\}) & \\ = & U^S_{lxk}(a)\{x \leftarrow U^S_k(b)\} & \end{array}$$

Note that since the *RHS* of the equation to prove is defined, from the last line above we learn that $z' \notin lxk \cup S$ and $z' \notin FV(U^S_k(b))$.

"Subst.L." refers to the substitution lemma for the λ-calculus [Bar84], which is also valid for our restricted notion of substitution and reads as follows: $s\{x \leftarrow t\}\{y \leftarrow u\} = s\{y \leftarrow u\}\{x \leftarrow t\{y \leftarrow u\}\}$ if $x \notin FV(u)$ for distinct variables x and y, and both sides of the equation together with the term $t\{y \leftarrow u\}$ are defined.

■

Lemma 6.72 (**U is modular w.r.t valuations**) Let us consider a de Bruijn valuation $\kappa = (\kappa_i, \kappa_t)$, a de Bruijn pre-metaterm A, a finite set of variables S, a variable assignment θ_v verifying the hypothesis in Def. 6.67, a label of binder indicators l and a label of variables k. If the following conditions hold:

1. κ is valid,
2. κA is defined,
3. θ_v is defined over l and the bound o-metavariables in $U_l(A)$,

4. θ_v is injective on the bound o-metavariables,

5. $\mathtt{Names}(FV(\kappa A)\backslash\!\!\backslash|\theta_v(l)k|) \subseteq S$, and

6. $\mathcal{WF}_l(A)$.

Then, $U^S_{\theta_v(l)k}(\kappa A) =_\alpha U_{(\theta_v,S,k)}(\kappa)U_l(A)$.

Intuitively k represents the context information where the reduction is performed (thus k is a label of variables). We also require $\mathtt{Names}(FV(\kappa A)\backslash\!\!\backslash|\theta_v(l)k|) \subseteq S$ to ensure that $U^S_{\theta_v(l)k}(\kappa A)$ is defined.

Proof. By induction on A. Below we shall use LHS and RHS to denote the left and right hand side respectively, of the equation to prove.

- $A = X_h$. Since $\mathcal{WF}_l(X_h)$ by the Hypothesis 6, we have that $h = l$ and so $LHS = U^S_{\theta_v(l)k}(\kappa X_l)$. And on the other hand

$$\begin{array}{rcll} RHS & = & U_{(\theta_v,S,k)}(\kappa)(U_l(X_l)) & \\ & = & U_{(\theta_v,S,k)}(\kappa)X & \\ & = & U^S_{\theta_v(l')k}(\kappa X_{l'}) & \quad (\text{ with } X_{l'} \in Dom(\kappa)) \end{array}$$

Then since κ is valid (Hypothesis 1) we may apply Lemma 6.68 to conclude.

- $A = \mathtt{S}^j(\widehat{\alpha})$. Since $\mathcal{WF}_l(\mathtt{S}^j(\widehat{\alpha}))$ holds by the Hypothesis 6, then $j = |l|$. We have

$$\begin{array}{rcl} LHS & = & U^S_{\theta_v(l)k}(\kappa \mathtt{S}^{|l|}(\widehat{\alpha})) \\ & = & U^S_{\theta_v(l)k}(\mathtt{S}^{|l|}(\kappa_i(\widehat{\alpha}))) \\ & = & \begin{cases} \mathtt{at}(k, \kappa_i(\widehat{\alpha})) & \text{if } \kappa_i(\widehat{\alpha}) \leq |k| \\ x_{\kappa_i(\widehat{\alpha})-|k|} & \text{otherw. with } x_{\kappa_i(\widehat{\alpha})-|k|} \in S \end{cases} \end{array}$$

On the other hand we have

$$\begin{array}{rcl} RHS & = & U_{(\theta_v,S,k)}(\kappa)(U_l(\mathtt{S}^{|l|}(\widehat{\alpha}))) \\ & = & U_{(\theta_v,S,k)}(\kappa)\widehat{\alpha} \\ & = & \theta_v(\widehat{\alpha}) \end{array}$$

We can conclude $LHS = RHS$ because θ_v satisfies the requirements of Def. 6.67.

- $A = \mathtt{S}^j(1)$. Since $\mathcal{WF}_l(\mathtt{S}^j(1))$ holds by the Hypothesis 6, $j + 1 \leq |l|$. Thus,

$$\begin{array}{rcl} LHS & = & U^S_{\theta_v(l)k}(\kappa \mathtt{S}^j(1)) \\ & = & U^S_{\theta_v(l)k}(\mathtt{S}^j(1)) \\ & = & \mathtt{at}(\theta_v(l), j+1) \\ & = & \theta_v(\mathtt{at}(l, j+1)) \end{array} \qquad \begin{array}{rcl} RHS & = & U_{(\theta_v,S,k)}(\kappa)(U_l(\mathtt{S}^j(1))) \\ & = & U_{(\theta_v,S,k)}(\kappa)(\mathtt{at}(l, j+1)) \\ & = & \theta_v(\mathtt{at}(l, j+1)) \end{array}$$

- $A = \xi(A_1, \ldots, A_n)$. Then we reason as follows

$$\begin{array}{rcll} RHS & = & U_{(\theta_v,S,k)}(\kappa)(U_l(\xi(A_1, \ldots, A_n))) & \\ & = & U_{(\theta_v,S,k)}(\kappa)(\xi\alpha.(U_{\alpha l}(A_1), \ldots, U_{\alpha l}(A_n))) & (\text{where } \alpha \text{ satisfies } 1 \leq i \leq n..) \\ & & & (..\mathcal{WF}_{\alpha l}(A_i), \text{for } \alpha \notin l) \\ & = & \xi\theta_v(\alpha).(U_{(\theta_v,S,k)}(\kappa)(U_{\alpha l}(A_1)), \ldots, U_{(\theta_v,S,k)}(\kappa)(U_{\alpha l}(A_n)))) & \end{array}$$

In order to apply the induction hypothesis we need to verify the hypothesis for A_i. The Hypothesis 1 holds by definition and the Hypothesis 2 is evident since κA is defined. Hypothesis 3 holds since by hypothesis θ_v is defined over l and the bound o-metavariables in $U_l(A) = \xi\alpha.(U_{\alpha l}(A_1), \ldots, U_{\alpha l}(A_n))$, hence it is defined over the bound o-metavariables in $U_{\alpha l}(A_i) \cup \alpha l$ for $1 \leq i \leq n$. We have then to verify the Hypothesis 5, that is, $\mathtt{Names}(FV(\kappa A_i)\backslash\!\!\backslash|\theta_v(\alpha l)k|) \subseteq S$; but this is evident by the Hypothesis 5 for A and the general fact that $FV(\xi(a_1, \ldots, a_n))\backslash\!\!\backslash n = FV(a_1, \ldots, a_n)\backslash\!\!\backslash n+1$. Hypothesis 6 is also true because when translating the de Bruijn pre-metaterm A we choose α verifying the condition $\mathcal{WF}_{\alpha l}(A_i)$. Thus applying the induction hypothesis we have:

$$\begin{array}{rcll}
RHS & = & \xi\theta_v(\alpha).(U^S_{\theta_v(\alpha l)k}(\kappa A_1),\ldots,U^S_{\theta_v(\alpha l)k}(\kappa A_n)) & \\
& =_\alpha & \xi z'.((U^S_{\theta_v(\alpha l)k}(\kappa A_1))\{\theta_v(\alpha)\leftarrow z'\},\ldots,(U^S_{\theta_v(\alpha l)k}(\kappa A_n))\{\theta_v(\alpha)\leftarrow z'\}) & (z' \text{ does not occur..}) \\
& & & (\text{..in } U^S_{\theta_v(\alpha l)k}(\kappa A_i)) \\
& =_\alpha & \xi z'.(U^S_{(\theta_v(\alpha l)k)\{\theta_v(\alpha)\leftarrow z'\}}(\kappa A_1),\ldots,U^S_{(\theta_v(\alpha l)k)\{\theta_v(\alpha)\leftarrow z'\}}(\kappa A_n)) & (\text{L}.A.18) \\
& =_\alpha & \xi z'.(U^S_{z'\theta_v(l)k}(\kappa A_1),\ldots,U^S_{z'\theta_v(l)k}(\kappa A_n)) & (\theta_v \text{ injective and..}) \\
& & & (\text{..hyp. Def. } 6.67) \\
& = & \xi z'.(U^S_{(z\theta_v(l)k)\{z\leftarrow z'\}}(\kappa A_1),\ldots,U^S_{(z\theta_v(l)k)\{z\leftarrow z'\}}(\kappa A_n)) & (z\notin\theta_v(l)k\cup S) \\
& =_\alpha & \xi z'.(U^S_{z\theta_v(l)k}(\kappa A_1)\{z\leftarrow z'\},\ldots,U^S_{z\theta_v(l)k}(\kappa A_n)\{z\leftarrow z'\}) & (\text{L}.A.18) \\
& =_\alpha & \xi z.(U^S_{z\theta_v(l)k}(\kappa A_1),\ldots,U^S_{z\theta_v(l)k}(\kappa A_n)) &
\end{array}$$

On the other hand we have

$$\begin{array}{rcll}
LHS & = & U^S_{\theta_v(l)k}(\kappa(\xi(A_1,\ldots,A_n))) & \\
& = & U^S_{\theta_v(l)k}(\xi(\kappa A_1,\ldots,\kappa A_n)) & \\
& = & \xi z.(U^S_{z\theta_v(l)k}(\kappa A_1),\ldots,U^S_{z\theta_v(l)k}(\kappa A_n)) & (z\notin\theta_v(l)k\cup S)
\end{array}$$

- $A=f(A_1,\ldots,A_n)$. Then we have

$$\begin{array}{rcl}
LHS & = & U^S_{\theta_v(l)k}(\kappa(f(A_1,\ldots,A_n))) \\
& = & U^S_{\theta_v(l)k}(f(\kappa A_1,\ldots,\kappa A_n)) \\
& = & f(U^S_{\theta_v(l)k}(\kappa A_1),\ldots,U^S_{\theta_v(l)k}(\kappa A_n))
\end{array}
\qquad
\begin{array}{rcl}
RHS & = & U_{(\theta_v,S,k)}(\kappa)(U_l(f(A_1,\ldots,A_n))) \\
& = & U_{(\theta_v,S,k)}(\kappa)f(U_l(A_1),\ldots,U_l(A_n)) \\
& = & f(U_{(\theta_v,S,k)}(\kappa)U_l(A_1),\ldots,U_{(\theta_v,S,k)}(\kappa)U_l(A_n))
\end{array}$$

We can immediately conclude by induction hypothesis.

- $A=A_1\llbracket A_2\rrbracket$. Then we have

$$\begin{array}{rcl}
LHS & = & U^S_{\theta_v(l)k}(\kappa(A_1\llbracket A_2\rrbracket)) \\
& = & U^S_{\theta_v(l)k}(\kappa A_1\{\!\{1\leftarrow\kappa A_2\}\!\})
\end{array}$$

On the other hand we have

$$\begin{array}{rcll}
RHS & = & U_{(\theta_v,S,k)}(\kappa)(U_l(A_1\llbracket A_2\rrbracket)) & \\
& = & U_{(\theta_v,S,k)}(\kappa)(U_{\alpha l}(A_1)[\alpha\leftarrow U_l(A_2)]) & (\text{where } \alpha \text{ is such that } 1\le i\le n..) \\
& & & (..\mathcal{WF}_{\alpha l}(A_1), \text{for } \alpha\notin l) \\
& = & (U_{(\theta_v,S,k)}(\kappa)(U_{\alpha l}(A_1)))\{\theta_v(\alpha)\leftarrow U_{(\theta_v,S,k)}(\kappa)(U_l(A_2))\} & \\
& =_\alpha & U^S_{\theta_v(\alpha l)k}(\kappa A_1)\{\theta_v(\alpha)\leftarrow U^S_{\theta_v(l)k}(\kappa A_2)\} & (\text{i.h.})
\end{array}$$

Remark that in the last step the inductive hypothesis may be applied by the same reasons we used in the case of the binder.

Now, since θ_v is injective and satisfies the conditions of Def. 6.67, then $\theta_v(\alpha)\notin S\cup\theta_v(l)k$ and we can then conclude by applying Lemma 6.71.

•

The reader should note that the translation of a valid de Bruijn valuation is an admissible named valuation. Recall that a valuation is admissible for a rewrite rule (G,D) iff the following conditions hold:

- θ is safe for (G,D) (Def. 6.20),
- if α and β occur in (G,D) with $\alpha\neq\beta$ then $\theta_v\alpha\neq\theta_v\beta$, and
- θ verifies the path condition (Def. 6.21) for every t-metavariable in (G,D).

Safeness is considered in Lemma A.21 and Lemma A.23 goes on to consider admissibility. Both results are developed in the appendix (Section A.3.2). So we move on directly to the main result of this section, i.e. that the *SERS* formalism preserves *SERS*$_{db}$-rewriting.

Proposition 6.73 (Simulating *SERS$_{db}$*-rewriting via *SERS*-rewriting) Assume $a \longrightarrow b$ in the *SERS$_{db}$* formalism using rewrite rule (L, R). Then $U(a) \longrightarrow U(b)$ in the *SERS* formalism using rule $U(L, R)$.

Proof. Let us consider the de Bruijn rewrite step $a \longrightarrow b$ using a de Bruijn valuation κ which is valid for (L, R). Without loss of generality we can suppose that κ is only defined on the metavariables of (L, R). And, let $U(L, R) = (G, D)$. By definition of the rewrite relation we have a de Bruijn context E such that $a = E[\kappa L]$ and $b = E[\kappa R]$. We proceed as follows:

- Take S as the set of variables $\mathtt{Names}(FV(a))$ so that $U^S_\epsilon(a)$ is defined. Note that since $FV(b) \subseteq FV(a)$ holds by Corollary 6.45, $U^S_\epsilon(b)$ is also defined.
- Take any *simple* label k of variables such that $S \cap k = \emptyset$ and $|k|$ is the binder path number of E .
- Now, to apply Lemma 6.69 we need to show that $U^S_k(\kappa L)$ and $U^S_k(\kappa R)$ are defined, which follows from the first and second items. Therefore, $U^S_\epsilon(E[\kappa L]) =_\alpha U^S_\epsilon(E)[U^S_k(\kappa L)]$ and $U^S_\epsilon(E[\kappa R]) =_\alpha U^S_\epsilon(E)[U^S_k(\kappa R)]$.
- The next step is to apply Lemma 6.72 in order to decompose $U^S_k(\kappa L)$ and $U^S_k(\kappa R)$. First of all, let us fix any variable assignment θ_v such that it verifies the following requirements:
  - it is defined over all the o-metavariables in $U_\epsilon(L)$ and $U_\epsilon(R)$ and only on these.
  - it is injective on the bound o-metavariables,
  - $\theta_v(\alpha) \notin S \cup k$ for any bound o-metavariable $\alpha \in Dom(\theta_v)$ (i.e. the variables assigned to bound o-metavariables in the rewrite rule $(U(L), U(R))$ are not confused with the free variables in a and b, that is, with the variables in S, nor with the variables bound in (the label of) the context where the rewrite-step takes place, that is, the variables in k).
  - We also define θ_v on the free o-metavariables the rewrite rule $(U(L), U(R))$ as the hypothesis dictates, i.e. for all $\widehat{\alpha}$ we define

$$\theta_v(\widehat{\alpha}) \stackrel{\text{def}}{=} \begin{cases} \mathtt{at}(k, \kappa_i(\widehat{\alpha})) & \text{if } \kappa_i(\widehat{\alpha}) \leq |k| \\ x_{\kappa_i(\widehat{\alpha})-|k|} & \text{otherw. with } x_{\kappa_i(\widehat{\alpha})-|k|} \in S \end{cases}$$

  We shall now consider the case of $U^S_k(\kappa L)$, the other one being similar. We must thus meet the conditions of the lemma in order to resolve $U^S_k(\kappa L)$. Let $l = \epsilon$.

  1. κ is valid by hypothesis.
  2. κL is defined since $a = E[\kappa L]$.
  3. We also have that θ_v is defined over ϵ and the bound o-metavariables in $U_\epsilon(L)$ and $U_\epsilon(R)$,
  4. The assignment θ_v is injective over the bound o-metavariables,
  5. $\mathtt{Names}(FV(\kappa L)\setminus\!\!\setminus|k|)$ holds since by definition we set $S = \mathtt{Names}(FV(a))$ (Note that by Corollary 6.45 we have $\mathtt{Names}(FV(b)) \subseteq S$),
  6. Finally, $\mathcal{WF}_\epsilon(L)$ holds since (L, R) is a de Bruijn rewrite rule and hence L and R are well-formed de Bruijn pre-metaterms.

  We may thus apply Lemma 6.72.

Let us summarize our situation:

$$\begin{array}{lll} U^S_\epsilon(E[\kappa L]) & =_\alpha & U^S_\epsilon(E)[U^S_k(\kappa L)] & \text{(L.6.69)} \\ & =_\alpha & U^S_\epsilon(E)[U_{(\theta_v,S,k)}(\kappa)(U_\epsilon(L))] & \text{(L.6.72)} \\ & = & U^S_\epsilon(E)[U_{(\theta_v,S,k)}(\kappa)G] & \end{array}$$

and

$$\begin{array}{lll} U^S_\epsilon(E[\kappa R]) & =_\alpha & U^S_\epsilon(E)[U^S_k(\kappa R)] & \text{(L.6.69)} \\ & =_\alpha & U^S_\epsilon(E)[U_{(\theta_v,S,k)}(\kappa)(U_\epsilon(R))] & \text{(L.6.72)} \\ & = & U^S_\epsilon(E)[U_{(\theta_v,S,k)}(\kappa)D] & \end{array}$$

So we now define the named context $C \stackrel{\text{def}}{=} U^S_\epsilon(E)$ and we also define the named valuation $\theta' \stackrel{\text{def}}{=} U_{(\theta_v,S,k)}(\kappa)$. Then we have $U(a) = C[\theta' G]$ and $U(b) = C[\theta' D]$. In order to conclude that $U(a) \longrightarrow U(b)$, by definition of *SERS*-rewriting, we are left to verify that θ' is admissible for (G, D). Now,

1. We have that $U_{(\theta_v,S,k)}(\kappa)$ is defined for *all* the metavariables of G and D since $U_{(\theta_v,S,k)}(\kappa)(G)$ and $U_{(\theta_v,S,k)}(\kappa)(D)$ are defined.
2. We have that θ_v is injective on *all* the bound o-metavariables in (G, D) by definition of θ_v.

We can then apply Lemma A.23 and conclude that θ' is admissible for (G, D).

6.5 Preserving Confluence

This section studies the relationship between the translation functions over pre-metaterms and terms introduced above. This gives rise to two results stating, respectively, that given a metaterm M then $U(T(M))$ is v-equivalent to M (see Figure 6.4), and that given a de Bruijn metaterm A then $T(U(A))$ is identical to A. These results are used to show that confluence is preserved when translating an *SERS* rewrite system into a *SERS$_{db}$* rewrite system, and are listed below and proved in the appendix (Section A.3.3):

- Let $M \in \mathcal{PMT}$ such that $\mathcal{WF}(M)$. Then $U(T(M)) =_v M$ (Corollary A.26).
- Let $t \in \mathcal{T}$. Then $U(T(t)) =_\alpha t$ (Corollary A.28).
- Let $A \in \mathcal{PMT}_{db}$. If $\mathcal{WF}(A)$ then $T(U(A)) = A$ (Corollary A.30).
- Let $a \in \mathcal{T}_{db}$. Then $T(U(a)) = a$ (Corollary A.32).

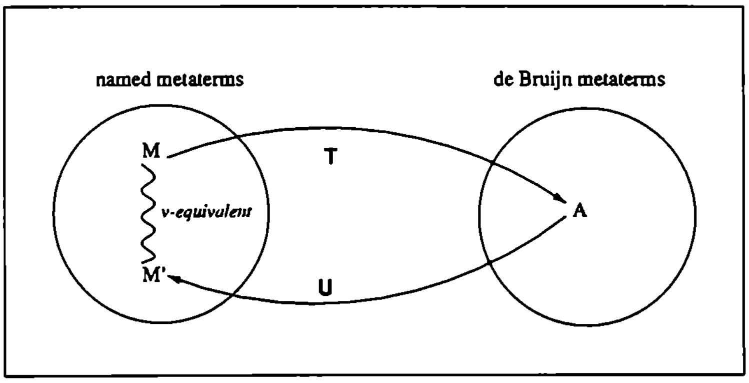


Figure 6.4: v-equivalence

Lemma 6.74 Let (G, D) and (G', D') be *SERS* rewrite rules such that $G =_v G'$ and $D =_v D'$. Then $\rightarrow_{(G,D)} = \rightarrow_{(G',D')}$.

Proof. Without loss of generality we prove that if $s \rightarrow_{(G,D)} t$ then $s \rightarrow_{(G',D')} t$. Thus let us assume that there is an admissible valuation θ for (G, D) and a context C such that $s =_\alpha C[\theta G]$ and $C[\theta D] =_\alpha t$.

The set of bound o-metavariables occurring in (G', D') may be divided into two (not necessarily disjoint) sets $\mathcal{B}_1$ and $\mathcal{B}_2$. In $\mathcal{B}_1$ we find those bound o-metavariables which occur in the parameter path of some t-metavariable in (G', D'), and in $\mathcal{B}_2$ the other bound variables occurring in (G', D'). The o-metavariables in $\mathcal{B}_1$ are not renamed in any way by the v-equivalence relation (Def. 6.11). We define the valuation $\theta' = (\theta'_t, \theta'_v)$ as follows:

$$\begin{array}{rcll} \theta'_t X & \stackrel{\text{def}}{=} & \theta_t X & \\ \theta'_v \alpha & \stackrel{\text{def}}{=} & \theta_v \alpha & \text{if } \alpha \in \mathcal{B}_1 \\ \theta'_v \widehat{\alpha} & \stackrel{\text{def}}{=} & \theta_v \widehat{\alpha} & \end{array}$$

In order to fully define θ' we must consider the value it assigns to those o-metavariables in $\mathcal{B}_2$ which are not in $\mathcal{B}_1$. For these we simply require θ' to assign any variables such that: the resulting valuation is safe for (G', D'), and θ'_v is injective on the bound o-metavariables.

Observe the following:

1. $MVar(G', D') \subseteq Dom(\theta')$,
2. θ' is by construction an admissible valuation for (G', D'), and
3. $s =_\alpha C[\theta G] =_\alpha C[\theta' G']$ and $t =_\alpha C[\theta D] =_\alpha C[\theta' D']$.

Hence $s \rightarrow_{(G',D')} t$. ▪

Corollary 6.75 Let (G, D) be an *SERS* rewrite rule. Then the rewrite relations generated by (G, D) and $U(T(G, D))$ are identical.

Proof. Use Corollary A.26, Lemma 6.74 and the fact that the translations preserve well-formedness.

Theorem 6.76 If $\mathcal{R}$ is a confluent *SERS* then $T(\mathcal{R})$ is a confluent $SERS_{db}$.

Proof. Suppose $a \twoheadrightarrow_{T(\mathcal{R})} b$ and $a \twoheadrightarrow_{T(\mathcal{R})} c$ for some de Bruijn terms a, b, c. Applying the translation mapping $U(\bullet)$ and using Proposition 6.73 we may obtain the diagram (b) of Figure 6.5. The reductions denoted by the dotted lines are obtained by Corollary 6.75 and the confluence of $\mathcal{R}$.

Now applying the translation mapping $T(\bullet)$ and using Proposition 6.62 we obtain the diagram (c) of Figure 6.5. Finally, Corollary A.32 and Corollary A.30 yield the desired diagram illustrated as diagram (a) in Figure 6.5.

$$\begin{tikzcd}
 & a \arrow[dl, "T(\mathcal{R})"'] \arrow[dr, "T(\mathcal{R})"] & \\
b \arrow[dr, dotted, "T(\mathcal{R})"'] & & c \arrow[dl, dotted, "T(\mathcal{R})"] \\
 & T(s) &
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
 & U(a) \arrow[dl, "U(T(\mathcal{R}))"'] \arrow[dr, "U(T(\mathcal{R}))"] & \\
U(b) \arrow[dr, dotted, "U(T(\mathcal{R}))"'] & & U(c) \arrow[dl, dotted, "U(T(\mathcal{R}))"] \\
 & s &
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
 & T(U(a)) \arrow[dl, "T(U(T(\mathcal{R})))"'] \arrow[dr, "T(U(T(\mathcal{R})))"] & \\
T(U(b)) \arrow[dr, dotted, "T(U(T(\mathcal{R})))"'] & & T(U(c)) \arrow[dl, dotted, "T(U(T(\mathcal{R})))"] \\
 & T(s) &
\end{tikzcd}$$

(c)

Figure 6.5: Diagrams for preservation of confluence

Theorem 6.77 If $\mathcal{R}$ is a confluent $SERS_{db}$ then $U(\mathcal{R})$ is a confluent *SERS*.

Proof. Suppose $s \rightarrow_{U(\mathcal{R})} t_1$ and $s \rightarrow_{U(\mathcal{R})} t_2$ for some terms s, t_1, t_2. Applying the translation mapping $T(\bullet)$ and using Proposition 6.62 we may obtain the diagram (b) of Figure 6.6. The reductions denoted by the dotted lines are obtained by the confluence of $\mathcal{R}$. Note also that Corollary A.30 has been used.

Now applying the translation mapping $U(\bullet)$ and using Proposition 6.73 we obtain the diagram (c) of Figure 6.6. Finally, Corollary A.28 and the definition of reduction Def. 6.23 yield the desired diagram illustrated as diagram (a) in Figure 6.6. ▪

Note that in fact the proofs of Theorem 6.76 and Theorem 6.77 are applicable to the more general diamond property (Def. 2.2(1)) hence we obtain preservation of this property in both directions.

$$\begin{tikzcd}
 & s \arrow[dl, "U(\mathcal{R})"'] \arrow[dr, "U(\mathcal{R})"] & \\
t_1 \arrow[dr, dotted, "U(\mathcal{R})"'] & & t_2 \arrow[dl, dotted, "U(\mathcal{R})"] \\
 & U(a) &
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
 & T(s) \arrow[dl, "\mathcal{R}"'] \arrow[dr, "\mathcal{R}"] & \\
T(t_1) \arrow[dr, dotted, "\mathcal{R}"'] & & T(t_2) \arrow[dl, dotted, "\mathcal{R}"] \\
 & a &
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
 & U(T(s)) \arrow[dl, "U(\mathcal{R})"'] \arrow[dr, "U(\mathcal{R})"] & \\
U(T(t_1)) \arrow[dr, dotted, "U(\mathcal{R})"'] & & U(T(t_2)) \arrow[dl, dotted, "U(\mathcal{R})"] \\
 & U(a) &
\end{tikzcd}$$

(c)

Figure 6.6: Diagrams for preservation of confluence

Chapter 7

From Higher-Order to First-Order Rewriting

As observed in Chapter 6 substitution may not be dismissed as simple replacement (also known as grafting in some circles) as in first-order theories. Thus many researchers became interested in the formalization of higher-order substitution by *explicit substitutions*, so that higher-order systems/formalisms could be expressible in first-order systems/formalisms: the notion of variable binding is dropped because substitution becomes replacement. A well-known example of the combination of de Bruijn indices and explicit substitutions is the formulation of different first-order calculi for the λ-calculus [ACCL91, BBLRD96, KR95, DG01]. Other examples are the translations of higher-order unification to first-order unification modulo [DHK00], higher-order logic to first-order logic modulo [DHK01], higher-order theorem proving to first-order theorem proving modulo [DHK98], etc.

Now the case of the λ-calculus is interesting but at the same time not fully representative of the problems we are faced with when encoding a higher-order system into a first-order setting. For in this particular case it is enough to take care of α-conversion and promote metalevel substitution to the object-level. Indeed, replacing the usual variables names by de Bruijn indices and introducing explicit substitutions suffices to yield a first-order rewrite system, as the above mentioned examples illustrate. However, this is not always the case for an arbitrary higher-order rewrite system. In other words, eliminating α-conversion and introducing explicit substitutions is not enough to yield an equivalent full first-order system (full in the sense of first-order rewriting modulo an empty equational theory). The reason is that in higher-order rewriting[1] the *LHS* of a rewrite rule is a higher-order pattern [Nip91, Oos94]. So we must somehow *also* encode higher-order pattern matching when encoding in the first-order framework. The fact that introducing de Bruijn indices plus explicit substitutions suffices for the λ-calculus is saying that for this particular rewrite system higher-order matching is doing nothing more than what first-order matching could do. We stress, once again, that this is not always the case. A simple example of such a fact, which we shall consider in this chapter, is the η_{db}-rewrite rule:

$$\lambda(app(X_\alpha, 1)) \rightarrow_{\eta_{db}} X_\epsilon$$

Note that X_ϵ on the right-hand side of the rule, which does not appear in any binding context, is related to the occurrence of X_α on the left-hand side, which appears inside a binding context. This may be seen as the reason why the η_{db}-rule has received so much attention [Río93, Har92, Bri95] since syntactic matching no longer suffices[2]. That is to say, 'occurs check' is a feature of higher-order pattern matching which first-order matching cannot cope with. In an example η_{db}-rewrite step the reader may verify that the term $\lambda(app(3, 1))$ rewrites to 2. In a first-order setting with explicit substitution, we have the alternative formulation:

$$\lambda(app(X[\uparrow], 1)) \rightarrow X$$

[1] That is, higher-order rewriting in the *SERS* higher-order rewrite formalism, though in an arbitrary higher-order rewrite formalism (such as *HRS*) the *LHS* need not be a (higher-order) pattern.

[2] When represented in the *HRS* formalism [Nip91] the *LHS* of η is a higher-order pattern, moreover it is a non-fully-extended pattern (there are free variables not applied to all bound variables above it). The problems introduced by η in this case are due to the latter fact. However, this is not the only problematic situation. It may be the case that the *LHS* of a rule is a fully-extended pattern yet introducing de Bruijn indices and promoting substitution to the object-level does not suffice to obtain a full first-order system. An example of the latter phenomenon is the rule $f(\lambda x.\lambda y.F(x, y), \lambda x.\lambda y.F(y, x)) \rightarrow c$. See Section 7.4.3.

However, in order for the term $X[\uparrow]$ to match the subterm 3 we need $\mathcal{E}$-matching, that is, matching modulo an equational theory $\mathcal{E}$. For an appropriate substitution calculus $\mathcal{E}$ we would need to solve the equation $3 \overset{?}{=}_{\mathcal{E}} X[\uparrow]$.

Another, perhaps less evident, example is given by a commutation rule C such as:

$$imply(\exists\alpha.\forall\beta.X, \forall\beta.\exists\alpha.X) \to true$$

which expresses that the formula appearing as the first argument of the *imply* function symbol implies the one in the second argument. The naive translation to first-order, namely $imply(\exists(\forall(X)), \forall(\exists(X))) \to true$, is evidently not correct, so that we take its encoding in the de Bruijn higher-order formalism $SERS_{db}$ and then translate it to first-order via the conversion presented in this chapter obtaining C_{fo}:

$$imply(\exists(\forall(X)), \forall(\exists(X[2 \cdot 1 \cdot \uparrow^2]))) \to true$$

Now, the rule C_{fo} has exactly the same intended meaning as the original higher-order rule C: in order for a term to be an instance of this rule the term t' instantiated for the rightmost X must be the one instantiated for the leftmost X, say t, except that all 1-level and 2-level indices in t shall be interchanged in order to obtain t'. Of course, The following rewrite rule C'_{fo} also does the job:

$$imply(\exists(\forall(X[2 \cdot 1 \cdot \uparrow^2])), \forall(\exists(X))) \to true$$

However, note that both C_{fo} and C'_{fo} induce the same rewrite relation on terms.

The goal of this chapter is to provide a conversion algorithm for encoding higher-order rewrite systems in first-order rewriting modulo an equational theory $\mathcal{E}$. This is interesting from a theoretical point of view because the expressive power of higher and first-order formalisms may be compared. However, another more practical issue arises, that of the possibility of *transferring results* developed in the first-order framework to the higher-order one. In Chapter 8 we shall transfer the Standardization Theorem from first-order rewriting to higher-order rewriting. Techniques concerning confluence, termination, completion, evaluation strategies, etc. should be looked at. Moreover, this is interesting for two further reasons: on one hand it is still not fully clear how to transfer techniques such as dependency pairs [AG00], semantic labelling [Zan95] or completion [BD88] to the higher-order framework, and on the other hand, termination techniques such as RPO for higher-order systems [JR99] turn out to be much more complicated than their respective first-order versions [Der82, KL80]. Also, we obtain a characterization of the class of $SERS_{db}$ (including the λ-calculus) for which a translation to a full ($\mathcal{E} = \emptyset$) first-order rewrite system exists. We shall argue that it is this class of systems, dubbed the *essentially first-order* $SERS_{db}$, that are better suited for the above mentioned transfer of properties.

To the best of our knowledge there are, at least, two formalisms, B.Pagano's *XRS* [Pag98] and M-O.Stehr's *CINNI* [Ste00], which study encoding of higher-order rewrite formalisms as first-order rewriting using explicit substitutions. The formalism *XRS*, which is a first-order formalism, is based on de Bruijn indices and is presented as a generalization of the first-order $\lambda\sigma_{\Uparrow}$-calculus [HL89] to higher-order rewriting and *not* as a first-order formulation of higher-order rewriting. Consequently, as we have seen in Chapter 6, many well-known higher-order rewriting systems cannot be expressed in such a formalism. In the case of *CINNI* a similar situation arises, no relation to established HORS in the literature is presented. Also, the fact that the definition of the higher-order rewriting formalism used is not fully clear does not allow us to consider transferring results from the first-order framework. Chapter 6 has provided a presentation of higher-order rewriting based on de Bruijn indices ($SERS_{db}$) which does away with α-conversion and has established precise links between the *ERS* formalism and $SERS_{db}$. Here we take the next step, and encode all $SERS_{db}$ as first-order rewrite systems with the aid of explicit substitutions. Moreover, we do not attach to the encoding any particular substitution calculus. Instead, we have chosen to work with an abstract formulation of substitution calculi, as done for example in [Kes96, Kes00] to deal with confluence proofs of λ-calculi with explicit substitutions. As a consequence, the method we propose can be put to work in the presence of different calculi of explicit substitution such as σ [ACCL91], $\sigma_{\Uparrow}$ [HL89], υ [BBLRD96], f [Kes96], d [Kes96], s [KR95], χ [LRD95].

Finally, we mention the work of van Oostrom and van Raamsdonk [OR93]. Although it is common to call rewriting in the presence of binders and substitution higher-order rewriting (practice which we have also followed), in full precision it is only over terms that we abstract. However, in higher-order rewrite formalisms such as *HRS* we may abstract over functions or functions that take functions as arguments, and so on. In [OR93] it is shown that *CRS* and *HRS* have the same "matching power" when attention is restricted to *pattern HRS*. However, *HRS* have more "rewriting power" than *CRS*, in other words one *HRS*-rewrite step needs (possibly) many *CRS*-rewrite steps in order to be simulated. This is because substitution is computed in *CRS* by means of a subclass of β-derivations called *developments* (cf. Def. 8.37) whereas *HRS* have full β-rewriting at its disposal for computing substitutions.

Structure of the chapter

The chapter begins by taking a closer look at i-metavariables (Def. 6.26). In particular we explain why the present chapter deals with $SERS_{db}$ *without* i-metavariables. We trust the reader is convinced of the convenience of such a decision. We then introduce the first-order rewriting framework with explicit substitutions *ExERS* which shall constitute the destination formalism of our Conversion Procedure. This requires defining Basic Substitution Calculi, a macro-based presentation of calculi of explicit substitutions adapted to the present setting from [Kes96, Kes00]. This general presentation shall allow us the freedom of choosing from a wide range of calculi of explicit substitution when converting a higher-order rewrite system to first-order.

Section 7.3 introduces the Conversion Procedure, heart of this chapter, and illustrates its use with some examples. This procedure takes a $SERS_{db}$ $\mathcal{R}$ and produces a first-order modulo rewrite system $fo(\mathcal{R})_{\mathcal{W}}$, where $\mathcal{W}$ is some Basic Calculus of Explicit Substitutions (such as for example σ). It is the Conversion Procedure's responsability to compute index ajustments in order correctly encode higher-order pattern matching in the first-order setting. The rewrite rules produced may or may not have occurrences of the explicit substitution operator on the *LHS*s. If this is not the case then syntactic matching suffices. Otherwise, as in the η_{db} example, we need matching modulo the induced equational theory of the basic substitution calculus $\mathcal{W}$. The former systems are dubbed *essentially first-order* higher-order rewrite systems.

This is followed by a study of the properties of this procedure: independence of pivot selection (a technicallity concerning the Conversion Procedure), the Simulation Proposition and the Projection Proposition. The Simulation Proposition states that $fo(\mathcal{R})_{\mathcal{W}}$ is able to simulate $\mathcal{R}$-rewriting. Conversely, the Projection Proposition states that if $a \rightarrow_{fo(\mathcal{R})_{\mathcal{W}}} b$ then $\mathcal{W}(a) \twoheadrightarrow_{\mathcal{R}} \mathcal{W}(b)$, in fact we shall see that one $fo(\mathcal{R})_{\mathcal{W}}$-rewrite step may be encoded as one *parallel* $\mathcal{R}$-rewrite step.

We conclude by presenting the definition of essentially first-order higher-order rewrite systems, the class of higher-order rewrite systems that lend themselves to a full first-order conversion (rewrite system modulo an empty equational theory). Chapter 8 shall transfer the Standardization Theorem for this class of systems.

7.1 On Index-Metavariables

In this chapter we shall deal with the $SERS_{db}$ formalism *without* i-metavariables. The main reason for excluding them is that they appear to be nothing more than a 'hack' in order to represent calculi such as λx and, in general, do not enjoy good properties. In order to delve a little deeper into this issue we shall make informal use of the notions of descendant and residual, however Chapter 8 presents full formal definitions. These notions shall not be used beyond the present section in this chapter.

The idea behind *orthogonality* in term rewriting is that the contraction of a redex does not destroy other redexes but instead leaves a number of their 'residuals' (for a precise definition see Section 8.2 in Chapter 8). This is referred to as the Residual Property. Having this in mind the following is a possible definition of orthogonality for $SERS_{db}$ (see [GKK00, KOvR93]):

Definition 7.1 (*$SERS_{db}$*-**Orthogonality**) Let $(\Sigma, \mathcal{R})$ be a $SERS_{db}$ such that $\mathcal{R} = \{(L_i, R_i) \mid i \in I\}$.

1. $\mathcal{R}$ is *non-overlapping* if the following holds:
    - Let $L_i = C[X^1_{l_1}] \ldots [X^n_{l_n}]$ where C is the pattern of L_i (Def. 6.31) and $X^j_{l_j}$ are all the metavariables in L_i. If the redex $\kappa(C[X^1_{l_1}] \ldots [X^n_{l_n}])$ contains an instance of L_j for some $j \neq i$, then this instance must be already contained in one of the $\kappa(X^p_{l_p})$.
    - Likewise if $\kappa(C[X^1_{l_1}] \ldots [X^n_{l_n}])$ properly contains an instance of L_i.
2. $\mathcal{R}$ is *left-linear* if all L_i are linear (Def. 6.30).
3. $\mathcal{R}$ is *orthogonal* if it is non-overlapping and left-linear.

These are what one might consider as the 'natural' syntactic conditions for an $SERS_{db}$ to be considered orthogonal. They are a straightforward extension of orthogonality for first-order rewriting [Klo92]. Consider the following $SERS_{db}$ S consisting of two rewrite rules: $S = \{app(\lambda X_\alpha, Y_\epsilon) \rightarrow_{\beta_{db}} X_\alpha[Y_\epsilon], f(\widehat{\alpha}) \rightarrow_f c\}$ where app, f, c are function symbols and λ is a binder symbol.

Intuitively, S should by all means be regarded as an orthogonal system. One is relieved to know that indeed S satisfies orthogonality. Yet to one's surprise S is not confluent! Indeed,$app(\lambda(f(1)), b)$ reduces to $f(b)$ by the β_{db}-rule and to c by the f-rule followed by an application of the β_{db}-rule.

The problem stems in that S does not satisfy the Residual Property, a fundamental property of orthogonal systems, which in the case of first-order systems (and *ERS* and *CRS*) is implied by $SERS_{db}$-orthogonality:

Definition 7.2 (Residual Property) Let $\mathcal{R}$ be a $SERS_{db}$ and let (L, R) be a rewrite rule in $\mathcal{R}$. The *Residual Property* for $\mathcal{R}$ reads: the descendants of an (L, R)-redex u in a under contraction of any other non-overlapping redex v in a, are (L, R)-redexes.

In [KOO01a], Z.Khasidashvili et al define Context Sensitive *ERS* (*CCERS*). A *CCERS* is a *ERS* where term formation may be restricted (such as when considering typed terms in the λ-calculus) and the rewrite relation may be restricted to operate in certain contexts (possibly all in which case it is said to be context-free), and where the valid assignments κ may be restricted to some relevant subclass. In order to account for orthogonality in *CCERS* they consider a different definition of orthogonality which explicitly requires the Residual Property to be fulfilled. The reason for bringing this issue to the reader's attention is that one may consider S as a *CCERS* (more precisely a context-free conditional *ERS*, term formation and contexts are not restricted but valid valuations are) where for the f-rule we may replace $\widehat{\alpha}$ by a t-metavariable Z_ϵ and define the set of admissible assignments as those that assign only indices to this metavariable.

Definition 7.3 (*CCERS*-Orthogonality) A *CCERS* is orthogonal if

1. every *LHS* is linear,
2. redex patterns do not overlap, and
3. $\mathcal{R}$ satisfies the Residual Property[3].

Under this new definition S is no longer orthogonal: the term $f(b)$ (descendant of $f(1)$ in $app(\lambda(f(1)), b)$) is not an admissible redex since the valuation which assigns the term b to the metavariable Z_ϵ does not belong to the subclass of allowed valuations. This means that the local conditions of left-linearity and non-overlapping do not ensure that S behaves as expected (that is, is orthogonal in the sense of the Residual Property).

In full precision, S suffers a problem we might call 'lack of sort generality'. Let the sort $\mathtt{I}$ be the subset of de Bruijn terms that are de Bruijn indices and $\mathtt{T}$ be sort of de Bruijn terms. Then $\mathtt{I}$ is a subsort of $\mathtt{T}$. Consider once again the term $a = app(\lambda(f(1)), b)$. When we apply the f-rule to a we are claiming that 1 is of sort $\mathtt{I}$ (for these are the only valid values that the metavariable Z_ϵ may be instantiated with). Yet when we apply the β_{db}-rule to a we replace 1 with b: hence a term of sort $\mathtt{I}$ has been 'transformed' to a term of sort $\mathtt{T}$. This is perfectly valid since $\mathtt{I}$ is a subsort of $\mathtt{T}$, however the f-rule no longer copes with terms of sort $\mathtt{T}$.

As regards literature on higher-order rewriting where the presence of 'variable' metavariables in rewrite rules are allowed the work by P.A.Melliès in his PhD thesis [Mel96] should be mentioned. Melliès defines *CRS* with *names* where names are just a new sort of terms (See Remark 4.14 in [Oos97]). All in all we have the following sorts in the *CRS* with names framework: variables, names, term metavariables, name metavariables, terms and metaterms. Now name metavariables may only be substituted by names. This allows the *LHS*s of rewrite rules to contain free name metavariables and guarantees that the above mentioned problem does not arise. Also, this may be generalized to n-sorts. Note that although the *LHS*s of rules may contain free name metavariables, free variable metavariables are not permitted in the formalism since this would introduce the difficulties mentioned above. Returning to the $SERS_{db}$ framework we see that this sort-scheme present in the *CRS* with names formalism is not straightforwardly applicable as long as indices may be bound and potentially substituted by terms.

So we have, at least, three approaches to this problem:

1. Approach à la Melliès: introduce a new sort of variables which, either may be bound by binders but may only be substituted by other variables of the same sort (and not by terms), or may not be bound at all, in which case they behave as constants. The problem with these solutions is that they do not address the original motivation for introducing i-metavariables: representing indices which are free in the context of

[3] Formulated as Def. 7.2 but for *CCERS*.

where X ranges over $\mathcal{V}$, f over Γ_f, ξ over Γ_b, and σ over Γ_s. The arguments of σ are assumed to respect the sorts prescribed in its substitution declaration (i.e. d_i is a term or substitution in compliance with its substitution declaration), and function and binder symbols are assumed to respect their arities too.

Letters $a, b, c, \ldots$ and $s, s_i, \ldots$ are used for terms and substitutions, respectively. Letters $o, o', \ldots$ are used for all objects of the term algebra without making distinction of sorts. The $\bullet[\bullet]$ operator is called the ***substitution operator***. Binder symbols and substitution operators are considered as having binding power. We shall use $a[s]^n$ to abbreviate $a[s]\ldots[s]$ (n-times). Terms without occurrences of the substitution operator (resp. objects in $\mathcal{V}$) are called *pure* (resp. *ground*) terms. Similarly for contexts. A *context* is a ground term with one (and only one) occurrence of a distinguished term variable called a 'hole' (and denoted $\Box$). Letters $E, E_i, \ldots$ are used for contexts. The notion of *binder path number* is defined for pure contexts exactly as in the case of de Bruijn contexts (Def. 6.32). Note that contexts have no variables (except $\Box$).

The formalism of *ExERS* that we are going to use in order to encode higher-order rewriting consists of two sets of rewrite rules, a set of *proper rewrite rules*, and a set of substitution rules. Let us define these two concepts formally.

Definition 7.6 (Substitution macros) Let Γ_s be a substitution signature. The following symbols not included in Γ_s are called *substitution macros*: *cons* : (TS), *lift* : (S), *id* : (ϵ) and *shift*^{j} : (ϵ) for $j \geq 1$. We shall abbreviate *shift*^{1} by *shift*. Also, if $j \geq 0$ then $\mathit{lift}^j(s)$ stands for s if $j = 0$ and for $\mathit{lift}(\mathit{lift}^{j-1}(s))$ otherwise. Furthermore, if $j \geq 1$ then $\mathit{cons}(a_1, \ldots, a_j, s)$ stands for $\mathit{cons}(a_1, \ldots \mathit{cons}(a_j, s))$.

Definition 7.7 (Term rewrite and equational systems) Let Γ be an *ExERS* signature. An *equation* is a pair of terms $L \doteq R$ over Γ such that L and R have the same sort and a *term rewrite rule* is a pair of terms (L, R) over Γ, such that:

1. L and R have the same sort,
2. the head symbol of L is a function, binder or substitution symbol, and
3. the set of variables of L includes those of R.

An *equational* (resp. *term rewrite*) *system* is a set of equations (resp. term rewrite rules).

As usual, we shall need some mechanism for instantiating rewrite rules.

Definition 7.8 (Assignment) Let ρ be a (partial) function mapping variables in $\mathcal{V}$ to terms. We define an *assignment* $\overline{\rho}$ as the unique extension of ρ over the set $\mathcal{T}$ such that:

$$\begin{array}{lcl}
\overline{\rho}(n) & \overset{\text{def}}{=} & n \\
\overline{\rho}(X) & \overset{\text{def}}{=} & \rho(X) \\
\overline{\rho}(a[s]) & \overset{\text{def}}{=} & \overline{\rho}(a)[\overline{\rho}(s)] \\
\overline{\rho}(f(a_1, \ldots, a_n)) & \overset{\text{def}}{=} & f(\overline{\rho}(a_1), \ldots, \overline{\rho}(a_n)) \\
\overline{\rho}(\xi(a_1, \ldots, a_n)) & \overset{\text{def}}{=} & \xi(\overline{\rho}(a_1), \ldots, \overline{\rho}(a_n)) \\
\overline{\rho}(\sigma(d_1, \ldots, d_n)) & \overset{\text{def}}{=} & \sigma(\overline{\rho}(d_1), \ldots, \overline{\rho}(d_n))
\end{array}$$

We shall often abbreviate $\overline{\rho}$ as ρ. Assignments are required in order to define the rewrite relation induced by a rewrite system.

Definition 7.9 (Rewriting and Equality) Let o and o' be two ground terms of sort T or S. Given a rewrite system $\mathcal{R}$, we say that *o rewrites to o' in one step*, denoted $o \rightarrow_{\mathcal{R}} o'$, iff $o = E[\rho L]$ and $o' = E[\rho R]$ for some assignment ρ, some context E and some rewrite rule (L, R) in $\mathcal{R}$. We shall use $\twoheadrightarrow_{\mathcal{R}}$ to denote the reflexive transitive closure of the one-step rewrite relation.

Given an equational system $\mathcal{E}$, we say that *o equals o' modulo $\mathcal{E}$ in one step*, denoted $o =^1_{\mathcal{E}} o'$, iff $o = E[\rho L]$ and $o' = E[\rho R]$ for some assignment ρ, some context E and some equation $L \doteq R$ in $\mathcal{E}$. We use $=_{\mathcal{E}}$ to denote the reflexive symmetric transitive closure of $=^1_{\mathcal{E}}$, and say that *o equals o' modulo $\mathcal{E}$* if $o =_{\mathcal{E}} o'$.

Definition 7.10 (Substitution calculus) A *substitution calculus over an ExERS signature* Γ consists of a set $\mathcal{W}$ of first-order term rewrite rules, and an interpretation of each substitution macro as some combination of substitution symbols from Γ_s of corresponding signature. Def. 7.11 shall require certain properties for these interpretations to be considered meaningful.

An example of a substitution calculus is σ [ACCL91] with $cons(t,s) = t \cdot s$, $lift(s) = 1 \cdot (s \circ \uparrow)$, $id = id$ and $shift^j = \uparrow \circ \ldots (\uparrow \circ \uparrow)$, where $\uparrow$ appears j times. In [Kes96, Kes00] the reader will find full detailed proofs of this fact (that σ is a substitution calculus), and examples of further calculi of explicit substitutions that are also substitution calculi.

The next step is to add further requirements on substitution calculi in order for them to deserve that name. These conditions are assembled in the definition of a *Basic Substitution Calculus*.

Definition 7.11 (Basic substitution calculus) A substitution calculus $\mathcal{W}$ over Γ is said to be *basic* if the following conditions are satisfied:

1. $\mathcal{W}$ is complete (strongly normalizing and confluent) over the ground terms in $\mathcal{T}$. We use $\mathcal{W}(a)$ to indicate the unique $\mathcal{W}$-normal form of a.
2. $\mathcal{W}$-normal forms of ground terms are pure terms.
3. For each $f \in \Gamma_f$ and $\xi \in \Gamma_b$:
$$\begin{aligned} \mathcal{W}(f(a_1,\ldots,a_n)) &= f(\mathcal{W}(a_1),\ldots,\mathcal{W}(a_n)) \\ \mathcal{W}(\xi(a_1,\ldots,a_n)) &= \xi(\mathcal{W}(a_1),\ldots,\mathcal{W}(a_n)) \end{aligned}$$
4. Rules for propagating substitutions over function and binder symbols are contained in $\mathcal{W}$, for each $f \in \Gamma_f$ and $\xi \in \Gamma_b$[4]:
$$\begin{array}{lll} (Func_f) & f(X^1,\ldots,X^n)[s] & \to & f(X^1[s],\ldots,X^n[s]) \\ (Bind_\xi) & \xi(X^1,\ldots,X^n)[s] & \to & \xi(X^1[lift(s)],\ldots,X^n[lift(s)]) \end{array}$$
5. For every substitution s, $1[lift(s)] =_{\mathcal{W}} 1$.
6. For every substitution s and every $m \geq 0$, $m+1[lift(s)] =_{\mathcal{W}} m[s][shift]$.
7. For every term a and substitution s we have $1[cons(a,s)] =_{\mathcal{W}} a$.
8. For every term a, substitution s, $m \geq 0$ we have $m+1[cons(a,s)] =_{\mathcal{W}} m[s]$.
9. For every $m, j \geq 1$ we have $m[shift^j] =_{\mathcal{W}} m+j$.
10. For every ground term a we have $a[id] =_{\mathcal{W}} a$.

The first four conditions may be seen as primitive conditions that $\mathcal{W}$ should satisfy in order to be called a substitution calculus. The remaining conditions describe the behaviour expected of the substitution macros.

Example 7.12 The σ [ACCL91], $\sigma_{\Uparrow}$ [HL89] and ϕ [Muñ97a] calculi are basic substitution calculi where the set of function and binder symbols are $\{app\}$ and $\{\lambda\}$, respectively.

The reader may have noted that the macro-based presentation of substitution calculi makes use of parallel substitutions (since $cons(\bullet,\bullet)$ has substitution declaration TS). Nevertheless, the results presented in this work may be achieved via a macro-based presentation using a simpler set of substitutions (such as for example the one used in [Kes00]), where $scons(\bullet)$ (the 's' in *scons* is for 'simple') has substitution declaration T and the macro $shift^j$ is only defined for $j=1$. Indeed, the expression $a[cons(b_1,\ldots,b_n,shift^j)]$ could be denoted by the expression
$$a[lift^n(shift)]^j[scons(b_1[shift]^{n-1})]\ldots[scons(b_n)]$$

Definition 7.13 (*ExERS* and *FExERS*) Let Γ be an *ExERS* signature, $\mathcal{W}$ a basic substitution calculus over Γ and $\mathcal{R}$ a set of term rewrite rules. If each rule of $\mathcal{R}$ has sort T then $\mathcal{R}_{\mathcal{W}} \stackrel{\text{def}}{=} (\Gamma,\mathcal{R},\mathcal{W})$ is called an Explicit Expression Reduction System (*ExERS*). If, in addition, the *LHS* of each rule in $\mathcal{R}$ contains no occurrences of the substitution operator $\bullet[\bullet]$ then $\mathcal{R}_{\mathcal{W}}$ is called a Fully Explicit Expression Reduction System (*FExERS*).

[4] In contrast to the previous item we use $\to$ instead of $=_{\mathcal{W}}$.

Since rewriting in $SERS_{db}$ only takes place on terms, and first-order term rewrite systems will be used to simulate higher-order rewriting, all the rules of a term rewrite system $\mathcal{R}$ are assumed to have sort $\mathtt{T}$. However, rewrite rules of $\mathcal{W}$ may have any sort (i.e. $\mathtt{T}$ or $\mathtt{S}$).

Example 7.14 Consider the signature Γ formed by $\Gamma_f = \{app\}$, $\Gamma_b = \{\lambda\}$ and Γ_s any substitution signature. Let $\mathcal{W}$ be a basic substitution calculus over Γ. Then for $\mathcal{R} : \{app(\lambda X, Y) \rightarrow_{\beta_{db}} X[cons(Y, id)]\}$ we have that $\mathcal{R}_\mathcal{W}$ is an *FExERS*, and for $\mathcal{R}' : \mathcal{R} \cup \{\lambda(app(X[shift], 1)) \rightarrow_{\eta_{db}} X\}$, $\mathcal{R}'_\mathcal{W}$ is an *ExERS*.

Rewriting in an *ExERS* $\mathcal{R}_\mathcal{W}$ is first-order rewriting in $\mathcal{R}$ modulo $\mathcal{W}$-equality. In contrast, rewriting in a *FExERS* $\mathcal{R}_\mathcal{W}$ is just first-order rewriting in $\mathcal{R} \cup \mathcal{W}$.

Definition 7.15 (*ExERS* and *FExERS*-rewriting) Let $\mathcal{R}_\mathcal{W}$ be an *ExERS*, $\mathcal{R}'\mathcal{W}$ a *FExERS* and o, o' ground terms of sort $\mathtt{S}$ or $\mathtt{T}$. We say that o *$\mathcal{R}_\mathcal{W}$-reduces* or *rewrites to* o', written $o \rightarrow_{\mathcal{R}_\mathcal{W}} o'$, iff $o \rightarrow_{\mathcal{R}/\mathcal{W}} o'$ (i.e. $o =_\mathcal{W} o_1 \rightarrow_\mathcal{R} o'_1 =_\mathcal{W} o'$); and o *$\mathcal{R}'_\mathcal{W}$-reduces* or *rewrites to* o', iff $o \rightarrow_{\mathcal{R}' \cup \mathcal{W}} o'$.

We apologize for the abuse of notation: $o \rightarrow_{\mathcal{R}/\mathcal{W}} o'$ intuitively suggests that it is equivalence classes of terms that are rewritten however, as defined above, this is not the case. Instead, it is terms that are rewritten.

Example 7.16 Fix $\mathcal{W}$ to be the σ-calculus and consider the *FExERS* $\mathcal{R}_\sigma$ of Example 7.14. Then we have $1[app(\lambda 1, c) \cdot id] \rightarrow_{\mathcal{R}_\sigma} 1[1[c \cdot id] \cdot id]$. Also, $\lambda(app(3, 1)) \rightarrow_{\mathcal{R}'_\sigma} 2$, where $\mathcal{R}'_\sigma$ is that of Example 7.14. This follows from observing that $\lambda(app(3, 1)) =_\sigma \lambda(app(2[\uparrow], 1)) \rightarrow_{\eta_{db}} 2$.

7.2.1 Properties of Basic Substitution Calculi

This subsection takes a look at properties enjoyed by basic substitution calculi and introduces a condition called the *Scheme* [Kes00]. Basic substitution calculi satisfying the scheme ease inductive reasoning when proving properties over them without compromising the genericity achieved by the macro-based presentation.

Definition 7.17 (The Scheme) We say that a basic substitution calculus $\mathcal{W}$ obeys the *scheme* iff for every index m and every substitution symbol $\sigma \in \Gamma_s$ of arity q one of the following two conditions hold:

1. There exists a de Bruijn index n, positive numbers $i_1, \ldots, i_r$ ($r \geq 0$) and substitutions $u_1, \ldots, u_k$ ($k \geq 0$) such that
   - $1 \leq i_1, \ldots, i_r \leq q$ and all the i_j's are distinct
   - for all $o_1, \ldots, o_q$ we have: $m[\sigma(o_1, \ldots, o_q)] =_\mathcal{W} n[o_{i_1}] \ldots [o_{i_r}][u_1] \ldots [u_k]$
2. There exists an index i ($1 \leq i \leq q$) such that for all $o_1, \ldots, o_q$ we have: $m[\sigma(o_1, \ldots, o_q)] =_\mathcal{W} s_i$

We assume these equations to be well-typed: whenever the first case holds, then $o_{i_1}, \ldots, o_{i_r}$ are substitutions, whenever the second case holds, o_i is of sort $\mathtt{T}$.

Example 7.18 Example of calculi satisfying the scheme are σ, $\sigma_\Uparrow$, v, f and d [Kes96, Kes00].

We now take a quick look at some properties of arbitrary basic substitution calculi (that is, of basic substitution calculi that may or may not satisfy the scheme). On a first reading the reader may wish to skim over this section and proceed to the main section of this chapter, namely Section 7.3.

Lemma 7.19 (Behavior of Substitutions in Basic Substitution Calculi) Let $\mathcal{W}$ be a basic substitution calculus and $m \geq 1$.

1. For all $n \geq 0$ and substitution s in $\mathtt{S}$: $m[lift^n(s)] =_\mathcal{W} \begin{cases} m - n[s][shift]^n & \text{if } m > n \\ m & \text{if } m \leq n \end{cases}$
2. For all $n \geq m \geq 1$ and all terms $a_1, \ldots, a_n$: $m[cons(a_1, \ldots, a_n, s)] =_\mathcal{W} a_m$
3. For all pure terms a, b and $m \geq 1$: $a\{\!\{m \leftarrow b\}\!\} =_\mathcal{W} a[lift^{m-1}(cons(b, id))]$.

The first and third items of Lemma 7.19 are proved in [Kes00], the second item follows from the definition of a basic substitution calculus. For the proof of the following lemma the reader is referred to [Kes00].

Lemma 7.20 Let $\mathcal{W}$ be a basic substitution calculus, a a pure term and s a term of sort S. Then the following holds: $\mathcal{W}(a[s][\mathit{shift}]) = \mathcal{W}(a[\mathit{shift}][\mathit{lift}(s)])$.

Corollary 7.21 Let $\mathcal{W}$ be a basic substitution calculus, a a pure term and s a term of sort S. For every $m \geq n \geq 0$ we have $a[\mathit{shift}]^n[\mathit{lift}^m(s)] =_{\mathcal{W}} a[\mathit{lift}^{m-n}(s)][\mathit{shift}]^n$.

Lemma 7.22 Let $\mathcal{W}$ be a basic substitution calculus, a a pure term, and b a term of sort T. For every $n \geq 0$, $a[\mathit{lift}^n(\mathit{shift})][\mathit{lift}^n(\mathit{cons}(b, \mathit{id}))] =_{\mathcal{W}} a$

Proof. The proof of this fact uses the following result:

If $\mathcal{W}$ is a basic substitution calculus, c is a term of sort T and s a term of sort S. Then for every $m \geq 1$ and $n \geq 0$

$$m[\mathit{lift}^n(\mathit{cons}(c, s))] =_{\mathcal{W}} \begin{cases} m - n - 1[s][\mathit{shift}]^n & \text{if } m > n+1 \\ m & \text{if } m < n+1 \\ c[\mathit{shift}]^n & \text{if } m = n+1 \end{cases}$$

•

Lemma 7.23 (Substitution commutation) Let $\mathcal{W}$ be a basic substitution calculus, a a pure term, b any term, s a term of sort S. Then for every $m \geq n \geq 0$ we have:
$a[\mathit{lift}^n(\mathit{cons}(b, \mathit{id}))][\mathit{lift}^m(s)] =_{\mathcal{W}} a[\mathit{lift}^{m+1}(s)][\mathit{lift}^n(\mathit{cons}(b[\mathit{lift}^{m-n}(s)], \mathit{id}))]$.

Proof. By induction on the structure of a.

- $a = j$. Then we consider three further cases:
  - $j > n + 1$.

$$\begin{array}{ll} & \mathcal{W}(a[\mathit{lift}^n(\mathit{cons}(b, \mathit{id}))][\mathit{lift}^m(s)]) \\ =_{L.\ 7.19(1)} & \mathcal{W}(j - n[\mathit{cons}(b, \mathit{id})][\mathit{shift}]^n[\mathit{lift}^m(s)]) \\ = & \mathcal{W}(j - n - 1[\mathit{shift}]^n[\mathit{lift}^m(s)]) \\ =_{L\ 7.21} & \mathcal{W}(j - n - 1[\mathit{lift}^{m-n}(s)][\mathit{shift}]^n) \end{array}$$

and

$$\begin{array}{ll} & \mathcal{W}(a[\mathit{lift}^{m+1}(s)][\mathit{lift}^n(\mathit{cons}(b[\mathit{lift}^{m-n}(s)], \mathit{id}))]) \\ = & \mathcal{W}(j - n - 1[\mathit{lift}^{m-n}(s)][\mathit{shift}][\mathit{shift}]^n[\mathit{lift}^n(\mathit{cons}(b[\mathit{lift}^{m-n}(s)], \mathit{id}))]) \\ = & \mathcal{W}(j - n - 1[\mathit{lift}^{m-n}(s)][\mathit{shift}][\mathit{cons}(b[\mathit{lift}^{m-n}(s)], \mathit{id})][\mathit{shift}]^n) \\ =_{L\ 7.22} & \mathcal{W}(j - n - 1[\mathit{lift}^{m-n}(s)][\mathit{id}][\mathit{shift}]^n) \\ =_{Def\ 7.11(10)} & \mathcal{W}(j - n - 1[\mathit{lift}^{m-n}(s)][\mathit{shift}]^n) \end{array}$$

  - $j = n + 1$.

$$\begin{array}{ll} & \mathcal{W}(j[\mathit{lift}^n(\mathit{cons}(b, \mathit{id}))][\mathit{lift}^m(s)]) \\ =_{L.\ 7.19(1)} & \mathcal{W}(1[\mathit{cons}(b, \mathit{id})][\mathit{shift}]^n[\mathit{lift}^m(s)]) \\ = & \mathcal{W}(b[\mathit{shift}]^n[\mathit{lift}^m(s)]) \\ = & \mathcal{W}(b[\mathit{lift}^{m-n}(s)][\mathit{shift}]^n) \end{array}$$

and

$$\begin{array}{ll} & \mathcal{W}(j[\mathit{lift}^{m+1}(s)][\mathit{lift}^n(\mathit{cons}(b[\mathit{lift}^{m-n}(s)], \mathit{id}))]) \\ = & \mathcal{W}(1[\mathit{lift}^{m-n+1}(s)][\mathit{shift}]^n[\mathit{lift}^n(\mathit{cons}(b[\mathit{lift}^{m-n}(s)], \mathit{id}))]) \\ = & \mathcal{W}(1[\mathit{shift}]^n[\mathit{lift}^n(\mathit{cons}(b[\mathit{lift}^{m-n}(s)], \mathit{id}))]) \\ = & \mathcal{W}(n + 1[\mathit{lift}^n(\mathit{cons}(b[\mathit{lift}^{m-n}(s)], \mathit{id}))]) \\ = & \mathcal{W}(1[\mathit{cons}(b[\mathit{lift}^{m-n}(s)], \mathit{id})][\mathit{shift}]^n) \\ = & \mathcal{W}(b[\mathit{lift}^{m-n}(s)][\mathit{shift}]^n) \end{array}$$

  - $j < n + 1$. Then we have:

$$\begin{array}{ll} & \mathcal{W}(j[\mathit{lift}^n(\mathit{cons}(b, \mathit{id}))][\mathit{lift}^m(s)]) \\ =_{L.\ 7.19(1)} & \mathcal{W}(j[\mathit{lift}^m(s)]) \\ =_{L.\ 7.19(1)} & j \\ =_{L.\ 7.19(1)} & \mathcal{W}(j[\mathit{lift}^n(\mathit{cons}(b[\mathit{lift}^{m-n}(s)], \mathit{id}))]) \\ =_{L.\ 7.19(1)} & \mathcal{W}(j[\mathit{lift}^{m+1}(s)][\mathit{lift}^n(\mathit{cons}(b[\mathit{lift}^{m-n}(s)], \mathit{id}))]) \end{array}$$

- $a = f(a_1, \ldots, a_n)$ or $a = \xi(a_1, \ldots, a_n)$. Use the induction hypothesis.

7.3 From Higher-Order to First-Order Rewriting

We now present the *Conversion Procedure*, an algorithm to translate any higher-order rewrite system in the formalism $SERS_{db}$ to a first-order $ExERS$. The Conversion Procedure is somewhat involved since several conditions, mainly related to the labels of metavariables, must be met in order for a valuation to be admitted as *valid* (Def. 6.41). Consider for instance the η_{db}-rewrite rule $\lambda(app(X_\alpha, 1)) \longrightarrow X_\epsilon$. The condition on valuations in $SERS_{db}$ in order to participate in the induced rewrite relation on terms is that they be valid, as we have seen in Chapter 6. Validity shall ensure, in this case, that the metavariable X_α is not instantiated to the index 1. The Conversion Procedure shall have to guarantee that this holds in a first-order setting. The idea is to replace all occurrences of metavariables X_l by a first-order variable X followed by an appropriate *index-adjusting explicit substitution* which computes valid valuations. Thus, the output would be: $\lambda(app(X[shift], 1)) \longrightarrow X$. However this is just a simple case, and in the general situation, incorporating *shift* macros shall not suffice. A witness to this fact is the commutation of binders rule in the introduction to this chapter.

We first give the conversion rules of the translation, then we prove its properties in Section 7.4.

7.3.1 The Conversion Procedure

Definition 7.24 (Binding allowance) Let A be a metaterm and $\{X_{l_1}, \dots, X_{l_n}\}$ the set of all the metavariables with name X occurring in A. Then, the *binding allowance of* X *in* A, noted $\mathtt{Ba}_A(X)$, is the set $\bigcap_{i=1}^n l_i$. Likewise, we define the *binding allowance of* X *in a rule* (L, R), written $\mathtt{Ba}_{(L,R)}(X)$, as the set $\bigcap_{i=1}^n l_i$ where $\{X_{l_1}, \dots, X_{l_n}\}$ is the set of all metavariables with the name X in L and R.

Example 7.25 Let $A = f(\xi(X_\alpha), g(\xi(\lambda(X_{\beta\alpha})), \xi(\lambda(X_{\alpha\gamma}))))$, then $\mathtt{Ba}_A(X) = \{\alpha\}$.

Definition 7.26 (Shifting index) Let A be a metaterm, X_l a metavariable occurring in A, and i a position in l. The *shifting index determined by* X_l *at position* i, denoted $\mathtt{Sh}(X_l, i)$, is defined as

$$\mathtt{Sh}(X_l, i) \quad \stackrel{\text{def}}{=} \quad |\{j \mid \mathtt{at}(l, j) \notin \mathtt{Ba}_A(X), j \in 1..i-1\}|$$

Thus $\mathtt{Sh}(X_l, i)$ is just the total number of binder indicators in l at positions $1..i-1$ that do not belong to $\mathtt{Ba}_A(X)$. Remark that $\mathtt{Sh}(X_l, 1)$ is always 0.

Example 7.27 If $A = f(\xi(X_\alpha), g(\xi(\lambda(X_{\beta\alpha})), \xi(\lambda(X_{\alpha\gamma}))))$ then $\mathtt{Sh}(X_\alpha, 1) = \mathtt{Sh}(X_{\alpha\gamma}, 2) = 0$, $\mathtt{Sh}(X_{\beta\alpha}, 2) = 1$.

Definition 7.28 (Pivot) Let (L, R) be a $SERS_{db}$-rewrite rule and $\{X_{l_1}, \dots, X_{l_n}\}$ be the set of all X-based metavariables in (L, R). If $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$, then X_{l_j} for some $j \in 1..n$ is called an *(X-based) pivot* if

1. $|l_j| \leq |l_i|$ for all $i \in 1..n$, and
2. $X_{l_j} \in L$ whenever possible.

A *pivot set for a rewrite rule* (L, R) is a set of pivot metavariables, one for each name X in L such that $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$. This notion extends to a set of rewrite rules as expected.

Note that Def. 7.28 admits the existence of more than one X-based pivot metavariable. A pivot set for (L, R) fixes a metavariable for each metavariable name having a non-empty binding allowance.

Example 7.29 Both metavariables $X_{\alpha\beta}$ and $X_{\beta\alpha}$ can be chosen as X-based pivot in the rewrite rule

$$Implies(\exists(\forall(X_{\alpha\beta})), \forall(\exists(X_{\beta\alpha}))) \longrightarrow true$$

In the rewrite rule $f(Y_\epsilon, g(\lambda(\xi(X_{\alpha\beta})), \lambda(\xi(X_{\beta\alpha}))) \longrightarrow \xi(X_\alpha, Y_\alpha)$ the metavariable X_α is the only possible X-based pivot, also, Y_ϵ is the only Y-based pivot.

Let us recall some notation from Def. 6.2. If $l = \alpha_1 \dots \alpha_n$ is a label of binder indicators then $\mathtt{at}(l, i) = \alpha_i$ for $i \in 1..n$. Also, $\mathtt{pos}(\alpha, l) = \alpha_i$ where i is the smallest number in $1..n$ such that $\alpha = \alpha_i$, and is undefined otherwise.

Definition 7.30 (Conversion of metavariables) Consider a $SERS_{db}$-rewrite rule (L, R) and a pivot set for (L, R). We consider the following cases for every metavariable name X occurring in L:

1. $\mathtt{Ba}_{(L,R)}(X) = \emptyset$. Then convert each metavariable X_l in (L,R) to the term $X[\mathit{shift}^{|l|}]$, and those metavariables X_l with $l = \epsilon$ simply to X.

   This shall allow, for example, the rewrite rule $f(\lambda(app(X_\alpha, 1), X_\epsilon)) \to X_\epsilon$ to be converted to the first-order rewrite rule $f(\lambda(app(X[\mathit{shift}], 1), X)) \to X$.

2. $\mathtt{Ba}_{(L,R)}(X) = \{\beta_1, \ldots, \beta_m\}$ with $m > 0$. Let X_l be the pivot metavariable for X given by the hypothesis. We convert all occurrences of a metavariable X_k in (L,R) to the term $X[\mathit{cons}(b_1, \ldots, b_{|l|}, \mathit{shift}^j)]$ where $j \overset{\text{def}}{=} |k| + |l \backslash \mathtt{Ba}_{(L,R)}(X)|$. The b_i's shall depend on whether X_k is a pivot metavariable or not, as described below. As an optimization and in the particular case that the resulting term $X[\mathit{cons}(b_1, \ldots, b_{|l|}, \mathit{shift}^j)]$ is of the form $X[\mathit{cons}(1, \ldots, |l|, \mathit{shift}^{|l|})]$, then we simply convert X_k to X.

   The substitution $\mathit{cons}(b_1, \ldots, b_{|l|}, \mathit{shift}^j)$ is called the *index-adjusting substitution corresponding to* X_k and each b_i is defined as follows:

   (a) if X_k is the pivot (hence $l = k$), then

   $$b_i = \begin{cases} i & \text{if } \mathtt{at}(l,i) \in \mathtt{Ba}_{(L,R)}(X) \\ |l| + 1 + \mathtt{Sh}(X_l, i) & \text{if } \mathtt{at}(l,i) \notin \mathtt{Ba}_{(L,R)}(X) \end{cases}$$

   (b) if X_k is not the pivot then

   $$b_i = \begin{cases} \mathtt{pos}(\beta_h, k) & \text{if } i = \mathtt{pos}(\beta_h, l) \text{ for some } \beta_h \in \mathtt{Ba}_{(L,R)}(X) \\ |k| + 1 + \mathtt{Sh}(X_l, i) & \text{otherwise} \end{cases}$$

   Recall that $\mathtt{at}(l,i)$ returns the symbol in label l at position i with $1 \le i \le |l|$, and $\mathtt{pos}(\alpha, l)$ returns the position of α in the label l assuming it is in l.

Note that for an index-adjusting substitution $\mathit{cons}(b_1, \ldots, b_{|l|}, \mathit{shift}^j)$ each b_i is a distinct de Bruijn index and less than or equal to j. Substitutions of this form, in the particular case where we fix the basic substitution calculus to σ, have been called pattern substitutions in [DHKP98], where unification of higher-order patterns via explicit substitutions is studied.

Now that we know how to convert metavariables we can address the conversion of rewrite rules. Before proceeding we recall that the name of a metavariable X_l is X. The names of the free metavariables of a metaterm M is written $\mathit{NFMVar}(M)$ (Def. 6.35).

Definition 7.31 (Conversion of rewrite rules) Let (L,R) be a SERS_{db}-rewrite rule and let P be a pivot set for (L,R). The *conversion of the rewrite rule* (L,R) *via* P, denoted $\mathcal{C}_P(L,R)$, is defined as $\mathcal{C}_P(L,R) \overset{\text{def}}{=} (\mathcal{C}_P^{(L,R)}(L), \mathcal{C}_P^{(L,R)}(R))$ where $\mathcal{C}_P^{(L,R)}(A)$ is defined by induction on A, where $\mathit{NFMVar}(A) \subseteq \mathit{NFMVar}(L)$, as:

$$\begin{array}{lcll}
\mathcal{C}_P^{(L,R)}(n) & \overset{\text{def}}{=} & n & \\
\mathcal{C}_P^{(L,R)}(X_l) & \overset{\text{def}}{=} & \begin{cases} X[\mathit{shift}^{|l|}] & \text{if } \mathtt{Ba}_{(L,R)}(X) = \emptyset \text{ and } l \neq \epsilon \\ X[\mathit{cons}(b_1, \ldots, b_{|l|}, \mathit{shift}^j)] & \text{if } \mathtt{Ba}_{(L,R)}(X) \neq \emptyset \text{ and} \\ & \mathit{cons}(b_1, \ldots, b_{|l|}, \mathit{shift}^j) \neq \\ & \mathit{cons}(1, \ldots, |l|, \mathit{shift}^{|l|}) \\ X & \text{otherwise} \end{cases} & \\
\mathcal{C}_P^{(L,R)}(f(A_1, \ldots, A_n)) & \overset{\text{def}}{=} & f(\mathcal{C}_P^{(L,R)}(A_1), \ldots, \mathcal{C}_P^{(L,R)}(A_n)) & \\
\mathcal{C}_P^{(L,R)}(\xi(A_1, \ldots, A_n)) & \overset{\text{def}}{=} & \xi(\mathcal{C}_P^{(L,R)}(A_1), \ldots, \mathcal{C}_P^{(L,R)}(A_n)) & \\
\mathcal{C}_P^{(L,R)}(A_1[\![A_2]\!]) & \overset{\text{def}}{=} & \mathcal{C}_P^{(L,R)}(A_1)[\mathit{cons}(\mathcal{C}_P^{(L,R)}(A_2), \mathit{id})] &
\end{array}$$

The term $X[\mathit{cons}(b_1, \ldots, b_{|l|}, \mathit{shift}^j)]$ on the *RHS* of the second clause is the index-adjusting substitution computed in Def. 7.30.

It should be noted how the de Bruijn metasubstitution operator $\bullet[\![\bullet]\!]$ is converted to the term substitution operator $\bullet[\bullet]$.

Example 7.32 Below we present some examples of conversion of rules. We have fixed $\mathcal{W}$ to be the σ-calculus.

$SERS_{db}$-rewrite rule	*Pivot selected*	*Converted rule*
$\lambda(app(X_\alpha, 1)) \rightarrow X_\epsilon$		$\lambda(app(X[\uparrow], 1)) \rightarrow X$
$\lambda(\lambda(X_{\alpha\beta})) \rightarrow \lambda(\lambda(X_{\beta\alpha}))$	$X_{\alpha\beta}$	$\lambda(\lambda X) \rightarrow \lambda(\lambda(X[2 \cdot 1 \cdot (\uparrow \circ \uparrow)]))$
$f(\lambda(\lambda(X_{\alpha\beta})), \lambda(\lambda(X_{\beta\alpha}))) \rightarrow \lambda(X_\gamma)$		$f(\lambda(\lambda(X[\uparrow \circ \uparrow])), \lambda(\lambda(X[\uparrow \circ \uparrow]))) \rightarrow \lambda(X[\uparrow])$
$app(\lambda X_\alpha, Z_\epsilon) \rightarrow_{\beta_{db}} X_\alpha\llbracket Z_\epsilon \rrbracket$	X_α, Z_ϵ on *LHS*	$app(\lambda X, Z) \rightarrow X[Z \cdot id]$

The dash in the 'pivot selected' column for the first and third rows indicates that the binding allowance of X in the respective rule is the empty set and hence no pivot is required.

Note that if the $SERS_{db}$-rewrite rule (L, R) which is input to the Conversion Procedure is such that for every name X in (L, R) there is a label l with *all* metavariables in (L, R) of the form X_l, then all X_l are replaced simply by X. This is the case of β_{db} of Example 7.32.

Example 7.33 (Foldl) Let us represent the usual *foldl*-recursion scheme over lists as defined for example in Haskell. Consider the *ExERS* signature containing $\Gamma_f = \{nil, const^5, foldl\}$ and $\Gamma_b = \{\xi\}$. Then the *foldl*-rewrite system:

$$\begin{array}{lcl} foldl(\xi(\xi(X_{\alpha\beta})), Y_\epsilon, nil) & \rightarrow & Y_\epsilon \\ foldl(\xi(\xi(X_{\alpha\beta})), Y_\epsilon, const(Z_\epsilon, W_\epsilon)) & \rightarrow & foldl(\xi(\xi(X_{\alpha\beta})), X_{\alpha\beta}\llbracket Y_\beta \rrbracket\llbracket Z_\epsilon \rrbracket, W_\epsilon) \end{array}$$

is converted to

$$\begin{array}{lcl} foldl(\xi(\xi(X)), Y, nil) & \rightarrow & Y \\ foldl(\xi(\xi(X)), Y, const(Z, W)) & \rightarrow & foldl(\xi(\xi(X)), X[Y[\uparrow] \cdot id][Z \cdot id], W) \end{array}$$

Example 7.34 (Natural numbers recursor) Consider the *ExERS* signature containing the function symbols $\Gamma_f = \{zero, suc, rec\}$ and binder symbols $\Gamma_b = \{\xi\}$. Then the *rec*-rewrite system:

$$\begin{array}{lcl} rec(\xi(\xi(X_{\alpha\beta})), Y_\epsilon, zero) & \rightarrow & Y_\epsilon \\ rec(\xi(\xi(X_{\alpha\beta})), Y_\epsilon, suc(Z_\epsilon)) & \rightarrow & X_{\alpha\beta}\llbracket Z_\beta \rrbracket\llbracket rec(\xi(\xi(X_{\alpha\beta})), Y_\epsilon, Z_\epsilon) \rrbracket \end{array}$$

is converted to

$$\begin{array}{lcl} rec(\xi(\xi(X)), Y, zero) & \rightarrow & Y \\ rec(\xi(\xi(X)), Y, suc(Z)) & \rightarrow & X[Z[\uparrow] \cdot id][rec(\xi(\xi(X)), Y, Z) \cdot id] \end{array}$$

Also, observe that if we replace our $cons(\bullet, \bullet)$ macro by a $scons(\bullet)$ of substitution declaration T as defined in [Kes96, Kes00] then the last clause of Def. 7.31 converts a metaterm of the form $A\llbracket B \rrbracket$ into $A[scons(B)]$, yielding first-order systems based on substitution calculi, such as v, which do not implement parallel substitution.

The system resulting from the Conversion Procedure is coded as an *ExERS*, a framework for defining first-order rewrite systems where $\mathcal{W}$-matching is used. Moreover, if it is possible, an *ExERS* may further be coded as a *FExERS* (Def. 7.13) where reduction is defined on first-order terms and matching is just syntactic first-order matching, obtaining a *full first-order system.*

Definition 7.35 (Conversion Procedure) Let Γ be an *ExERS* signature, let $\mathcal{R}$ be a $SERS_{db}$, and let $\mathcal{W}$ be a substitution calculus over Γ. The *Conversion Procedure* consists in selecting a pivot set for each rewrite rule in $\mathcal{R}$ and converting all its rewrite rules as dictated by Def. 7.31. The resulting set of rewrite rules is written $fo(\mathcal{R})$. The *ExERS* $fo(\mathcal{R})_\mathcal{W}$ is called a *first order-version* of $\mathcal{R}$.

In what follows we shall assume given some fixed basic substitution calculus $\mathcal{W}$. Thus, given a $SERS_{db}$ $\mathcal{R}$ we shall speak of *the* first-order version of $\mathcal{R}$.

Of course, we must also consider pivot selection. Assume given some rewrite rule (L, R) and different pivot sets P and Q for this rule. It is clear that $C_P(L, R)$ and $C_Q(L, R)$ shall not be identical.

[5] Although *cons* is the usual abbreviation for the list constructor, we shall use *const* so as not to cause confusion with the *cons*-macro.

Example 7.36 Consider the following binder-commutation rule

$$\mathit{imply}(\exists(\forall(X_{\beta\alpha})), \forall(\exists(X_{\alpha\beta}))) \rightarrow_C \mathit{true}$$

If we select $X_{\beta\alpha}$ as the X-based pivot we obtain the following conversion of C: $\mathit{imply}(\exists(\forall(X)), \forall(\exists(X[2 \cdot 1 \cdot \mathit{id}]))) \rightarrow_{C_{f_o}} \mathit{true}$. However, $X_{\alpha\beta}$ may also be selected as an X-based pivot metavariable. In this case, the resulting converted rewrite rule shall be different: $\mathit{imply}(\exists(\forall(X[2 \cdot 1 \cdot \mathit{id}])), \forall(\exists(X))) \rightarrow_{C'_{f_o}} \mathit{true}$

Nevertheless, the rewrite relation generated by both of these converted rewrite rules is identical.

Proposition 7.37 (Pivot Selection) Let (L, R) be a SERS_{db}-rewrite rule and let P and Q be different pivot sets for this rule. Then the rewrite relation generated by both $C_P(L, R)$ and $C_Q(L, R)$ are identical.

Proposition 7.37 is important, for it makes clear that the Conversion Procedure is not biased by the selection of pivot sets (as regards the induced rewrite relation). Thus only now may we speak of *the* first-order version of a SERS_{db} $\mathcal{R}$. The proof of this proposition is rather technical and is relegated to Section A.4.1 of the appendix.

7.4 Properties of the Conversion Procedure

This section studies the connection between higher-order rewriting and first-order rewriting modulo. Section 7.4.1 first shows that the *Simulation Proposition* holds: any higher-order rewrite step may be simulated or implemented by first-order rewriting. Section 7.4.2 considers the *Projection Proposition*, namely, that rewrite steps in the first-order version of a higher-order system $\mathcal{R}$ can be projected in $\mathcal{R}$. Finally, we give in Section 7.4.3 a syntactical characterization of higher-order rewrite systems that can be translated into first-order rewrite systems modulo an empty theory. We shall see that, for example, the λ-calculus is covered by this characterization.

7.4.1 The Simulation Proposition

In order to simulate higher-order rewriting in a first-order framework we have to deal with the conversion of valid valuations into assignments. Recall that valuations are the devices through which SERS_{db}-rewrite rules are instantiated in order to obtain the induced rewrite relation. Likewise, assignments are used for instantiating first-order rewrite rules, i.e. *ExERS*-rewrite rules. For converting valuations to assignments two families of index-adjustment operations are required, decrementors and adjusters.

Consider a metavariable X_l in a SERS_{db}-rewrite rule (L, R), and suppose we are given a valid de Bruijn valuation κ. Let $X[\mathit{cons}(b_1, \ldots, b_{|k|}, \mathit{shift}^j)]$ be the conversion of the metavariable X_l (Def. 7.30) where k is the label of the X-based pivot metavariable. We shall seek to define an assignment ρ such that the value that ρ assigns to X satisfies the following equation:

$$\rho(X)[\mathit{cons}(b_1, \ldots, b_{|k|}, \mathit{shift}^j)] =_{\mathcal{W}} \kappa(X_l)$$

The term assigned to $\rho(X)$ shall be obtained from $\kappa(X_l)$. This result is stated as Lemma 7.45.

Definition 7.38 (Decrementors) For every $i, j \geq 0$ and de Bruijn ground term a we define $\mathcal{D}^j_i(a)$ as follows:

$$\begin{array}{rcl}
\mathcal{D}^j_i(n) & \overset{\mathrm{def}}{=} & \begin{cases} n & \text{if } n \leq i + j \\ n - j & \text{if } n > i + j \end{cases} \\
\mathcal{D}^j_i(f(a_1 \ldots a_n)) & \overset{\mathrm{def}}{=} & f(\mathcal{D}^j_i(a_1) \ldots \mathcal{D}^j_i(a_n)) \\
\mathcal{D}^j_i(\xi(a_1 \ldots a_n)) & \overset{\mathrm{def}}{=} & \xi(\mathcal{D}^j_{i+1}(a_1) \ldots \mathcal{D}^j_{i+1}(a_n))
\end{array}$$

Lemma 7.39 Consider a SERS_{db}-rewrite rule (L, R), metavariables $X_l, X_k \in (L, R)$, and a valuation κ valid for (L, R). For all $i \geq 0$, if

1. $\kappa X_l = D[a]$ for some pure context D having binder path number i,
2. $\mathit{Value}^i(l, a) = \mathit{Value}^i(k, b)$, and
3. the binding allowance of X in (L, R) is the empty set (i.e. $\mathtt{Ba}_{(L,R)}(X) = \emptyset$),

then $\mathcal{D}_i^{|l|}(a) = \mathcal{D}_i^{|k|}(b)$.

Proof. By induction on a.

- $a = n$. We consider the following cases:
  - $n \leq i + |l|$. Then $\mathcal{D}_i^{|l|}(a) = n$. If $n \leq i$ then since $\mathit{Value}^i(l, a) = n = \mathit{Value}^i(k, b)$, we have $b = n$ and the result holds.

    Otherwise, if $i < n \leq i + |l|$ then since by Hypothesis 2 we have $\mathit{Value}^i(l, n) = \mathtt{at}(l, n-i) = \mathit{Value}^i(k, b)$ we must have $b = m$ with $i < m \leq i + |k|$ and $\mathtt{at}(l, n-i) = \mathtt{at}(k, m-i)$. But by Hypothesis 3 there must be some $X_{l'}$ in (L, R) such that $\mathtt{at}(l, n-i) \notin l'$, and hence $\mathit{Value}(l', \kappa X_{l'}) \neq \mathit{Value}(l, \kappa X_l)$ by Def. 6.40 (since $\mathtt{at}(l, n-i)$ occurs in $\mathit{Value}(l, \kappa X_l)$ but $\mathtt{at}(l, n-i)$ does not occur in $\mathit{Value}(l', \kappa X_{l'})$), contradicting the assumption that κ is valid.
  - $n > i + |l|$. Then $\mathcal{D}_i^{|l|}(a) = n - |l|$. Also, since $\mathit{Value}^i(l, a) = x_{n-i-|l|} = \mathit{Value}^i(k, b)$, we have $b = m$ with $m > |k| + i$ and $n - |l| - i = m - |k| - i$. Then $\mathcal{D}_i^{|k|}(b) = m - |k|$ and the result holds.
- $a = f(a_1, \ldots, a_n)$. Then $\mathcal{D}_i^{|l|}(a) = f(\mathcal{D}_i^{|l|}(a_1), \ldots, \mathcal{D}_i^{|l|}(a_n))$.

  Now by Hypothesis 2 we have that $b = f(b_1, \ldots, b_n)$ with $\mathit{Value}^i(l, a_j) = \mathit{Value}^i(k, b_j)$ for all $1 \leq j \leq n$. Then the induction hypothesis yields $\mathcal{D}_i^{|l|}(a_j) = \mathcal{D}_i^{|k|}(b_j)$ for $j \in 1..n$ and we may conclude the case by Def. 7.38.
- $a = \xi(a_1, \ldots, a_n)$. Then $\mathcal{D}_i^{|l|}(a) = \xi(\mathcal{D}_{i+1}^{|l|}(a_1), \ldots, \mathcal{D}_{i+1}^{|l|}(a_n))$.

  Now by Hypothesis 2 we have that $b = \xi(b_1, \ldots, b_n)$ with $\mathit{Value}^{i+1}(l, a_j) = \mathit{Value}^{i+1}(k, b_j)$ for all $1 \leq j \leq n$. Then the induction hypothesis concludes the case.

∎

Lemma 7.40 Consider a SERS_{db}-rewrite rule (L, R), metavariables $X_l, X_k \in (L, R)$ and a valuation κ valid for (L, R). For all $i \geq 0$, if

1. $\kappa X_l = D[a]$ for some pure context D having binder path number i,
2. $\mathit{Value}^i(l, a) = \mathit{Value}^i(k, b)$, and
3. the binding allowance of X in (L, R) is the empty set (i.e. $\mathtt{Ba}_{(L,R)}(X) = \emptyset$),

then $\mathcal{D}_i^{|l|}(a)[\mathit{lift}^i(\mathit{shift}^{|k|})] =_{\mathcal{W}} b$.

Proof. By induction on a.

- $a = n$. Then we have three further cases to consider:
  1. $n \leq i$. Then $\mathcal{D}_i^{|l|}(n)[\mathit{lift}^i(\mathit{shift}^{|k|})] = n[\mathit{lift}^i(\mathit{shift}^{|k|})] =_{\mathcal{W}}^{L.1} n$. Now by Hypothesis 2 we have $\mathit{Value}^i(l, n) = n = \mathit{Value}^i(k, b)$ and therefore $b = n$ and we are done.
  2. $i < n \leq i + |l|$. Then since by Hypothesis 2 we have $\mathit{Value}^i(l, n) = \mathtt{at}(l, n-i) = \mathit{Value}^i(k, b)$ we must have $b = m$ with $i < m \leq i + |k|$ and $\mathtt{at}(l, n-i) = \mathtt{at}(k, m-i)$. But by Hypothesis 3 there must be some $X_{l'}$ in (L, R) such that $\mathtt{at}(l, n-i) \notin l'$, and hence $\mathit{Value}(l', \kappa X_l') \neq \mathit{Value}(l, \kappa X_l)$ by Def. 6.40 (since $\mathtt{at}(l, n-i)$ occurs in $\mathit{Value}(l, \kappa X_l)$ but $\mathtt{at}(l, n-i)$ does not occur in $\mathit{Value}(l', \kappa X_{l'})$), contradicting the assumption that κ is valid.
  3. $n > i + |l|$. Then

$$\begin{array}{ll} & \mathcal{D}_i^{|l|}(n)[\mathit{lift}^i(\mathit{shift}^{|k|})] \\ = & (n - |l|)[\mathit{lift}^i(\mathit{shift}^{|k|})] \\ =_{\mathcal{W}}^{L.1} & (n - |l| - i)[\mathit{shift}^{|k|}][\mathit{shift}]^i \\ =_{\mathcal{W}}^{Def.7.11(9)} & (n - |l| - i + |k|)[\mathit{shift}]^i \\ =_{\mathcal{W}} & n - |l| + |k| \end{array}$$

     The last equality follows from i applications of Def. 7.11(9).

     Now by Hypothesis 2 we have $\mathit{Value}^i(l, n) = x_{n-i-|l|} = \mathit{Value}^i(k, b)$ and therefore $b = m$ with $m > i + |k|$ and $n - i - |l| = m - i - |k|$. From this it follows that $n - |l| = m - |k|$ and we are done.

- $a = f(a_1, \ldots, a_n)$. Then $\mathcal{D}_i^{|l|}(a)[\mathit{lift}^i(\mathit{shift}^{|k|})] =_{\mathcal{W}} f(\mathcal{D}_i^{|l|}(a_1)[\mathit{lift}^i(\mathit{shift}^{|k|})], \ldots, \mathcal{D}_i^{|l|}(a_n)[\mathit{lift}^i(\mathit{shift}^{|k|})])$ by condition 2 of Def. 7.10.

  Now by Hypothesis 2 we have that $b = f(b_1, \ldots, b_n)$ with $\mathit{Value}^i(l, a_j) = \mathit{Value}^i(k, b_j)$ for all $1 \leq j \leq n$. Then the induction hypothesis yields $\mathcal{D}_i^{|l|}(a_j)[\mathit{lift}^i(\mathit{shift}^{|k|})] =_{\mathcal{W}} b_j$ for $j \in 1..n$ and we may conclude the case.

- $a = \xi(a_1, \ldots, a_n)$. Then by condition 2 of Def. 7.10 we have that

  $$\mathcal{D}_i^{|l|}(a)[\mathit{lift}^i(\mathit{shift}^{|k|})] =_{\mathcal{W}} \xi(\mathcal{D}_{i+1}^{|l|}(a_1)[\mathit{lift}^{i+1}(\mathit{shift}^{|k|})], \ldots, \mathcal{D}_{i+1}^{|l|}(a_n)[\mathit{lift}^{i+1}(\mathit{shift}^{|k|})])$$

  Now by Hypothesis 2 we have that $b = \xi(b_1, \ldots, b_n)$ with $\mathit{Value}^{i+1}(l, a_j) = \mathit{Value}^{i+1}(k, b_j)$ for all $1 \leq j \leq n$. Then the induction hypothesis concludes the case.

•

Definition 7.41 (Adjusters) Let X_l be a pivot metavariable in a SERS_{db}-rewrite rule (L, R), $i \geq 1$, a a de Bruijn ground term and let $\mathit{cons}(b_1, \ldots, b_{|l|}, \mathit{shift}^{|l| + |l \setminus \mathtt{Ba}_{(L,R)}(X)|})$ be the index-adjusting substitution corresponding to X_l. Then $\mathcal{A}_i^l(a)$ is defined as follows:

$$\mathcal{A}_i^l(n) \stackrel{\text{def}}{=} \begin{cases} n & \text{if } n \leq i \\ n & \text{if } \mathtt{at}(l, n-i) \in \mathtt{Ba}_{(L,R)}(X) \text{ and } 0 < n-i \leq |l| \\ \mathtt{undefined} & \text{if } \mathtt{at}(l, n-i) \notin \mathtt{Ba}_{(L,R)}(X) \text{ and } 0 < n-i \leq |l| \\ \mathtt{pos}(n-i, b_1 \ldots b_{|l|}) + i & \text{if } |l| < n-i \leq |l| + |l \setminus \mathtt{Ba}_{(L,R)}(X)| \\ n - |l \setminus \mathtt{Ba}_{(L,R)}(X)| & \text{if } n-i > |l| + |l \setminus \mathtt{Ba}_{(L,R)}(X)| \end{cases}$$

$$\mathcal{A}_i^l(f(a_1 \ldots a_n)) \stackrel{\text{def}}{=} f(\mathcal{A}_i^l(a_1) \ldots \mathcal{A}_i^l(a_n))$$
$$\mathcal{A}_i^l(\xi(a_1 \ldots a_n)) \stackrel{\text{def}}{=} \xi(\mathcal{A}_{i+1}^l(a_1) \ldots \mathcal{A}_{i+1}^l(a_n))$$

Lemma 7.42 (Well-definedness of Adjusters) Consider a SERS_{db}-rewrite rule (L, R) and some pivot set P for (L, R). Let $X_l \in (L, R)$ be the X-based pivot metavariable for some $X \in \mathit{NFMVar}(L)$, and let κ be a valuation valid for (L, R). For all $i \geq 0$, if

1. $\kappa X_l = E[a]$ for some pure context E with i the binding path number of E, and
2. the binding allowance of X in (L, R) is not empty (i.e. $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$),

then $\mathcal{A}_i^l(a)$ is defined.

Proof. By induction on a. We shall only consider the base case, the others follow by using the induction hypothesis. Suppose $a = n$. We have four further cases to consider:

1. $n \leq i$. Then there is no problem.
2. $i < n \leq i + |l|$. The only case of conflict is if $\mathtt{at}(l, n-i) \notin \mathtt{Ba}_{(L,R)}(X)$. Then there must be a $X_{l'}$ in L such that $\mathtt{at}(l, n-i) \notin l'$. Consequently $\mathit{Value}(l, \kappa X_l) \neq \mathit{Value}(l', \kappa X_{l'})$ since $\mathtt{at}(l, n-i)$ occurs in $\mathit{Value}(l, \kappa X_l)$ but $\mathtt{at}(l, n-i)$ does not occur in $\mathit{Value}(l', \kappa X_{l'})$. This contradicts the assumption that κ is valid for (L, R).
3. $|l| < n - i \leq |l| + |l \setminus \mathtt{Ba}_{(L,R)}(X)|$. Then we must verify that $\mathtt{pos}(n-i, b_1 \ldots b_{|l|})$ is defined. Now let $r = |l \setminus \mathtt{Ba}_{(L,R)}(X)|$ then by Def. 7.30 there are subindices $j_1 < \ldots < j_r$ such that $b_{j_1} = |l| + 1 + \mathtt{Sh}(X_l, j_1), \ldots, b_{j_r} = |l| + 1 + \mathtt{Sh}(X_l, j_r)$. By noting that $1 + \mathtt{Sh}(X_l, j_r) = r$ we are done.
4. $n - i > |l| + |l \setminus \mathtt{Ba}_{(L,R)}(X)|$. This case presents no problems.

•

Lemma 7.43 Consider a SERS_{db}-rewrite rule (L, R) and some pivot set P for (L, R). Let $X_l \in (L, R)$ be the X-based pivot metavariable for some $X \in \mathit{NFMVar}(L)$, let $X_k \in (L, R)$, and let κ be a valuation valid for (L, R). For all $i \geq 0$, if

1. $\kappa X_l = D[a]$ for some pure context D having binder path number equal to i,
2. $\mathit{Value}^i(l, a) = \mathit{Value}^i(k, b)$, and
3. The binding allowance of X in (L, R) is not empty (i.e. $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$),

then $\mathcal{A}^l_i(a)[\mathit{lift}^i(s)] =_{\mathcal{W}} b$ where $s = \mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^{|k|+|l \setminus \mathtt{Ba}_{(L,R)}(X)|})$ is the index-adjusting substitution corresponding to X_k.

Proof. Let $j = |k| + |l \setminus \mathtt{Ba}_{(L,R)}(X)|$. We proceed by induction on a.

- $a = n$. Then we have four further cases to consider:

  1. $n \leq i$. Then $\mathcal{A}^l_i(n)[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))] = n[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))] =_{\mathcal{W}}^{L.1} n$.
     Now by Hypothesis 2 we have $\mathit{Value}^i(l, n) = n = \mathit{Value}^i(k, b)$ and therefore $b = n$ and we are done.
  2. $i < n \leq i + |l|$. Here we consider the two cases:
     - $\mathtt{at}(l, n - i) \in \mathtt{Ba}_{(L,R)}(X)$. Then

       $$\begin{array}{ll} & \mathcal{A}^l_i(n)[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))] \\ = & n[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))] \\ =_{\mathcal{W}}^{L.1} & (n - i)[\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j)][\mathit{shift}]^i \\ =_{\mathcal{W}} & c_{n-i}[\mathit{shift}]^i \\ =_{\mathcal{W}} & c_{n-i} + i \end{array}$$

       So we are left to verify that $c_{n-i} + i = b$.
       Now by Hypothesis 2 we have $\mathit{Value}^i(l, n) = \mathtt{at}(l, n - i) = \mathit{Value}^i(k, b)$ and therefore $b = m$ with $i < m \leq |k| + i$ and $\mathtt{at}(l, n - i) = \mathtt{at}(k, m - i)$.
       We consider where c_{n-i} might 'come from'.
       (a) $n - i = \mathtt{pos}(\beta_h, l)$ with $\beta_h \in \mathtt{Ba}_{(L,R)}(X)$ and $c_{n-i} = \mathtt{pos}(\beta_h, k)$. But then by Hypothesis 2 and the fact that k is a simple label we must have $c_{n-i} = m - i$, which concludes the case.
       (b) There is no $\beta_h \in \mathtt{Ba}_{(L,R)}(X)$ with $n - i = \mathtt{pos}(\beta_h, l)$. This contradicts our assumption that $\mathtt{at}(l, n - i) \in \mathtt{Ba}_{(L,R)}(X)$.
       Note that in the particular case that $X_k = X_l$ then $c_{n-i} = n - i$ and we have $n - i + i = n$.
     - $\mathtt{at}(l, n-i) \notin \mathtt{Ba}_{(L,R)}(X)$. By Well-definedness of Adjusters (Lemma 7.42) this case is not possible.
  3. $|l| < n - i \leq |l| + |l \setminus \mathtt{Ba}_{(L,R)}(X)|$. Then

     $$\begin{array}{ll} & \mathcal{A}^l_i(n)[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))] \\ = & (\mathtt{pos}(n - i, d_1 \ldots d_{|l|}) + i)[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))] \\ =_{\mathcal{W}} & \mathtt{pos}(n - i, d_1 \ldots d_{|l|})[\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j)][\mathit{shift}]^i \\ =_{\mathcal{W}} & c_r[\mathit{shift}]^i \\ =_{\mathcal{W}} & c_r + i \end{array}$$

     where $r = \mathtt{pos}(n - i, d_1 \ldots d_{|l|})$. Note that Lemma 7.42 is used here.
     So we are left to verify that $c_r + i = b$.
     We must consider where c_r might 'come from':
     (a) $r = \mathtt{pos}(\beta_h, l)$ with $\beta_h \in \mathtt{Ba}_{(L,R)}(X)$ and $c_r = \mathtt{pos}(\beta_h, k)$. Then clearly, $\mathtt{at}(l, r) \in \mathtt{Ba}_{(L,R)}(X)$. However, since $r = \mathtt{pos}(n - i, d_1 \ldots d_{|l|})$ this means that $d_r = n - i$. Also, recall that we are currently considering the case $d_r = n - i > |l|$. But then by Def. 7.30 $\mathtt{at}(l, r) \notin \mathtt{Ba}_{(L,R)}(X)$ contradicting our knowledge of the opposite fact.
     (b) $c_r = |k| + 1 + \mathtt{Sh}(X_l, r)$.
     Now note that it is not possible for $d_r = r$ (and hence $\mathtt{at}(l, r) \in \mathtt{Ba}_{(L,R)}(X)$) since then we may reason as in item 3a. So $d_r = n - i = |l| + 1 + \mathtt{Sh}(X_l, r)$ (*). Recall that we are left to verify that $|k| + 1 + \mathtt{Sh}(X_l, r) + i = b$.
     Now by Hypothesis 2 we have $\mathit{Value}^i(l, n) = x_{n-i-|l|} = \mathit{Value}^i(k, b)$ and therefore $b = m$ with $m > i + |k|$ and $n - i - |l| = m - i - |k|$. From this it follows that $n - |l| = m - |k|$. So now we must see that $|k| + 1 + \mathtt{Sh}(X_l, r) + i = n - |l| + |k|$, or simply $1 + \mathtt{Sh}(X_l, r) + i = n - |l|$. This follows from (*).

Note that in the particular case where $X_k = X_l$ then $c_r = n - i$ and we have $n - i + i = n$.

4. $n - i > |l| + |l \setminus \mathtt{Ba}_{(L,R)}(X)|$. Then

$$\begin{array}{ll} & \mathcal{A}_i^l(n)[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))] \\ = & (n - |l \setminus \mathtt{Ba}_{(L,R)}(X)|)[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))] \\ =_{\mathcal{W}} & (n - |l \setminus \mathtt{Ba}_{(L,R)}(X)| - i)[\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j)][\mathit{shift}]^i \\ =_{\mathcal{W}} & n - |l \setminus \mathtt{Ba}_{(L,R)}(X)| - i - |l| + |k| + |l \setminus \mathtt{Ba}_{(L,R)}(X)| + i \\ = & n - |l| + |k| \end{array}$$

Note that in the particular case that $X_k = X_l$ we have $k = l$ and the result holds directly. Otherwise, by Hypothesis 2 we have $\mathit{Value}^i(l, n) = x_{n-i-|l|} = \mathit{Value}^i(k, b)$ and therefore $b = m$ with $m > i + |k|$ and $n - i - |l| = m - i - |k|$. From this it follows that $n - |l| = m - |k|$ and we may conclude the case.

- $a = f(a_1, \ldots, a_n)$. Then

$$\begin{array}{ll} & \mathcal{A}_i^l(a)[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))] \\ =_{\mathcal{W}} & f(\mathcal{A}_i^l(a_1)[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))], \ldots, \mathcal{A}_i^l(a_n)[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))]) \end{array}$$

  Now by Hypothesis 2 we have that $b = f(b_1, \ldots, b_n)$ with $\mathit{Value}^i(l, a_j) = \mathit{Value}^i(k, b_j)$ for all $1 \leq j \leq n$. Then the induction hypothesis concludes the case.

- $a = \xi(a_1, \ldots, a_n)$. Then

$$\begin{array}{ll} & \mathcal{A}_i^l(a)[\mathit{lift}^i(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))] \\ =_{\mathcal{W}} & \xi(\mathcal{A}_{i+1}^l(a_1)[\mathit{lift}^{i+1}(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))], \ldots, \mathcal{A}_{i+1}^l(a_n)[\mathit{lift}^{i+1}(\mathit{cons}(c_1, \ldots, c_{|l|}, \mathit{shift}^j))]) \end{array}$$

  Now by Hypothesis 2 we have that $b = \xi(b_1, \ldots, b_n)$ with $\mathit{Value}^{i+1}(l, a_j) = \mathit{Value}^{i+1}(k, b_j)$ for all $1 \leq j \leq n$. Then the induction hypothesis concludes the case.

■

We know how to convert SERS_{db}-rewrite rules. In order to prove our simulation result we must convert SERS_{db}-valuations. This makes use of decrementors and adjusters.

Definition 7.44 (Valuation conversion) Let (L, R) be a SERS_{db}-rewrite rule, κ a valid valuation for (L, R) and P a pivot set for (L, R). The *conversion of κ via P* is defined as the assignment ρ where for each $X \in \mathit{NFMVar}(L)$:

- Case $\mathtt{Ba}_{(L,R)}(X) = \emptyset$. Then $\rho(X) \stackrel{\text{def}}{=} \mathcal{D}_0^{|l|}(\kappa X_l)$ where X_l is any metavariable from L. Note that Lemma 7.39 guarantees that this is a correct definition (take $D = \Box$).

- Case $\mathtt{Ba}_{(L,R)}(X) = \{\beta_1, \ldots, \beta_n\}$ with $n > 0$. Then we define $\rho(X) \stackrel{\text{def}}{=} \mathcal{A}_0^l(\kappa X_l)$ where X_l is the X-based pivot metavariable as dictated by P.

Lemma 7.45 Let (L, R) be a SERS_{db}-rewrite rule, κ a valid valuation for (L, R) and $\overline{\rho}$ the conversion of κ via P for some pivot set P for (L, R). If $L = C[A]$ for some metacontext C and metaterm A, then $\overline{\rho}(C_P^{(L,R)}(A)) =_{\mathcal{W}} \kappa A$. Likewise, if $R = C[A]$ then $\overline{\rho}(C_P^{(L,R)}(A)) =_{\mathcal{W}} \kappa(A)$.

Proof. Both items are proved by induction on A.

- $A = n$. Then $\mathit{LHS} = \overline{\rho}(n) = n = \kappa n = \mathit{RHS}$.

- $A = X_k$. Note that since X_k is a subterm of a metaterm (i.e. a well-formed pre-metaterm) k is a simple label. According to Def. 7.31 we have three subcases to consider:

1. $\mathtt{Ba}_{(L,R)}(X) = \emptyset$. Then

$$\begin{array}{lll} LHS & = & \overline{\rho}(X[shift^{|k|}]) \\ & = & \overline{\rho}(X)[shift^{|k|}] \\ & =_{Def.\ 7.44} & \mathcal{D}_0^{|l|}(\kappa X_l)[shift^{|k|}] \\ & =_{\mathcal{W}}^{L.\ 7.40} & \kappa X_k \end{array}$$

   where X_l is any metavariable from L.

2. $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$ and $cons(b_1, \ldots, b_{|l|}, shift^j) \neq cons(1, \ldots, |l|, shift^{|l|})$ where X_l is the X-based pivot metavariable as dictated by P. Then

$$\begin{array}{lll} LHS & = & \overline{\rho}(X[cons(b_1, \ldots, b_{|l|}, shift^j)]) \\ & = & \overline{\rho}(X)[cons(b_1, \ldots, b_{|l|}, shift^j)] \\ & = & \mathcal{A}_0^l(\kappa X_l)[cons(b_1, \ldots, b_{|l|}, shift^j)] \\ & =_{\mathcal{W}}^{L.\ 7.43} & \kappa X_k \end{array}$$

3. $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$ and $cons(b_1, \ldots, b_{|l|}, shift^j) = cons(1, \ldots, |l|, shift^{|l|})$ where X_l is the X-based pivot metavariable as dictated by P. Then

$$\begin{array}{lll} LHS & = & \overline{\rho}(X) \\ & = & \mathcal{A}_0^l(\kappa X_l) \\ & =_{\mathcal{W}} & \mathcal{A}_0^l(\kappa X_l)[cons(1, \ldots, |l|, shift^{|l|})] \\ & =_{\mathcal{W}}^{L.\ 7.43} & \kappa X_k \end{array}$$

   Note that the third equality holds by the fact that $cons(1, \ldots, |l|, shift^{|l|})$ behaves as the identity substitution.

- $A = f(A_1, \ldots, A_n)$. Then $LHS = f(\overline{\rho}(C_P^{(L,R)}(A_1)), \ldots, \overline{\rho}(C_P^{(L,R)}(A_n))) =_{\mathcal{W}}^{i.h.} f(\kappa A_1, \ldots, \kappa A_n) = RHS$.
- $A = \xi(A_1, \ldots, A_n)$. Then $LHS = \xi(\overline{\rho}(C_P^{(L,R)}(A_1)), \ldots, \overline{\rho}(C_P^{(L,R)}(A_n))) =_{\mathcal{W}}^{i.h.} \xi(\kappa A_1, \ldots, \kappa A_n) = RHS$.
- $A = A_1[\![A_2]\!]$. This case is considered for the second item only since the de Bruijn metasubstitution operator may not occur on the LHS of a $SERS_{db}$-rewrite rule.

$$\begin{array}{lll} LHS & = & \overline{\rho}(C_P^{(L,R)}(A_1[\![A_2]\!])) \\ & = & \overline{\rho}(C_P^{(L,R)}(A_1))[cons(\overline{\rho}(C_P^{(L,R)}(A_2)), id)] \\ & =_{\mathcal{W}}^{i.h.} & (\kappa A_1)[cons(\kappa A_2, id)] \\ & =^{L.\ 7.19(3)} & \kappa A_1\{\!\{1 \leftarrow \kappa A_2\}\!\} \\ & = & \kappa(A_1[\![A_2]\!]) \\ & = & RHS \end{array}$$

∎

Proposition 7.46 (Simulation Proposition) Let $\mathcal{R}$ be a $SERS_{db}$ and let $fo(\mathcal{R})_{\mathcal{W}}$ be its first-order version. Suppose $a \rightarrow_{\mathcal{R}} b$ then

1. if $fo(\mathcal{R})_{\mathcal{W}}$ is an *ExERS* then $a \rightarrow_{fo(\mathcal{R})/\mathcal{W}} b$.
2. if $fo(\mathcal{R})_{\mathcal{W}}$ is a *FExERS* then $a \rightarrow_{fo(\mathcal{R})} \circ \twoheadrightarrow_{\mathcal{W}} b$ where $\circ$ denotes relation composition.

Proof. For the first item, suppose $a \rightarrow_{\mathcal{R}} b$. Then there must be a $SERS_{db}$-rewrite rule $(L, R) \in \mathcal{R}$, a valuation κ valid for (L, R) and a pure context E such that $a = E[\kappa L]$ and $b = E[\kappa R]$. Let $(L', R') = \mathcal{C}_P(L, R)$ be the converted version of rule (L, R) via some pivot set P for (L, R). Let $\overline{\rho}$ be the conversion of κ via P (Def. 7.44). By Lemma 7.45 we have:

1. $\overline{\rho}(L') =_{\mathcal{W}} \kappa L$ and
2. $\overline{\rho}(R') =_{\mathcal{W}} \kappa R$.

Thus from $\overline{\rho}(L') =_{\mathcal{W}} \kappa L$ and $\overline{\rho}(R') =_{\mathcal{W}} \kappa R$ we have $E[\overline{\rho}(L')] =_{\mathcal{W}} E[\kappa L]$ and $E[\overline{\rho}(R')] =_{\mathcal{W}} E[\kappa R]$, respectively. Finally, we have on the one hand $a = E[\kappa L] =_{\mathcal{W}} E[\overline{\rho}(L')]$, so $a =_{\mathcal{W}} E[\overline{\rho}(L')]$, and on the other, $b = E[\kappa R] =_{\mathcal{W}} E[\overline{\rho}(R')]$, so $b =_{\mathcal{W}} E[\overline{\rho}(R')]$.

As for the second item note that if $fo(\mathcal{R})_{\mathcal{W}}$ is a *FExERS* then L' is a pure term. Also, by definition, κ is a pure assignment. Thus $\overline{\rho}(L') = \kappa L$. And $\overline{\rho}(R') \twoheadrightarrow_{\mathcal{W}} \kappa R$ since κR is a pure term. Therefore we have $a = E[\kappa L] = E[\overline{\rho}(L')] \rightarrow_{(L',R')} E[\overline{\rho}(R')] \twoheadrightarrow_{\mathcal{W}} E[\kappa R]$.

■

7.4.2 The Projection Proposition

We now wish to prove that derivations in an *ExERS* or *FExERS* $fo(\mathcal{R})_{\mathcal{W}}$ may be projected into derivations in $\mathcal{R}$. This ensures in some sense that we did not add meaningless computations in the translated first-order system. As a consequence we prove that $fo(\mathcal{R})_{\mathcal{W}}$ is conservative over $\mathcal{R}$ (Def. 7.57). Further properties of the projected derivations shall be studied in Chapter 8, where standard derivations shall be considered.

We shall first begin by showing that if $a \rightrightarrows_{(L,R)} b$ then for any term s of sort S we have $\mathcal{W}(a[s]) \rightrightarrows_{(L,R)} \mathcal{W}(b[s])$. The meaning of $a \rightrightarrows_{(L,R)} b$ shall be made precise shortly, however on an intuitive level it means that a rewrites to b by applying a number of *parallel* (L,R)-rewrite steps.

Remark 7.47 Let A be a pre-metaterm and suppose $\mathcal{WF}_k(A)$. Then any metavariable occurring in A must be of the form X_{lk} for some label l. Moreover lk is a simple label.

Lemma 7.48 Let A be a pre-metaterm and suppose $\mathcal{WF}_k(A)$. Consider a valuation κ with $MVar(A) \subseteq Dom(\kappa)$. Then $\mathcal{W}((\kappa A)[lift^{|k|}(s)]) = \iota_k A$ where ι_k is a valuation defined as: $\iota_k(X_{lk}) \stackrel{\text{def}}{=} \mathcal{W}((\kappa X_{lk})[lift^{|lk|}(s)])$ for all l such that X_{lk} occurs in A.

Proof. By induction on A.

- $A = n$. Note that since $\mathcal{WF}_k(n)$ we have $n \leq |k|$. Then $LHS = \mathcal{W}((\kappa n)[lift^{|k|}(s)]) = \mathcal{W}(n[lift^{|k|}(s)]) = n = \iota_k n = RHS$.
- $A = X_{k'}$. Then since $\mathcal{WF}_k(X_{k'})$ we have $k = k'$ and $LHS = \mathcal{W}((\kappa X_k)[lift^{|k|}(s)]) = \iota_k A$.
- $A = f(A_1, \ldots, A_n)$. Then

$$\begin{array}{lll} LHS & =^{Def.\ 7.11(3,4)} & f(\mathcal{W}((\kappa A_1)[lift^{|k|}(s)]), \ldots, \mathcal{W}((\kappa A_n)[lift^{|k|}(s)])) \\ & =^{i.h.}_{\mathcal{W}} & f(\iota^1_k A_1, \ldots, \iota^n_k A_n) \\ & = & f(\iota_k A_1, \ldots, \iota_k A_n) \\ & = & RHS \end{array}$$

  where $\iota_k = \bigcup_{i=1}^n \iota^i_k$. Note that if $X_p \in Dom(\iota^j_k) \cap Dom(\iota^{j'}_k)$ for $j, j' \in 1..n$ with $j \neq j'$ then $\iota^j_k(X_p) = \iota^{j'}_k(X_p)$.

- $A = \xi(A_1, \ldots, A_n)$. By hypothesis there is an α such that $\mathcal{WF}_{\alpha k}(A_i)$ for all $i \in 1..n$. Then

$$\begin{array}{lll} LHS & =^{Def.\ 7.11(3,4)} & \xi(\mathcal{W}((\kappa A_1)[lift^{|k|+1}(s)]), \ldots, \mathcal{W}((\kappa A_n)[lift^{|k|+1}(s)])) \\ & =^{i.h.}_{\mathcal{W}} & \xi(\iota^1_{\alpha k} A_1, \ldots, \iota^n_{\alpha k} A_n) \\ & = & \xi(\iota_{\alpha k} A_1, \ldots, \iota_{\alpha k} A_n) \end{array}$$

  where $\iota_{\alpha k} = \bigcup_{i=1}^n \iota^i_{\alpha k}$. Note that if $X_p \in Dom(\iota^j_{\alpha k}) \cap Dom(\iota^{j'}_{\alpha k})$ for $j, j' \in 1..n$ with $j \neq j'$ then $\iota^j_{\alpha k}(X_p) = \iota^{j'}_{\alpha k}(X_p)$.

  By the well-formedness predicate we know that since any metavariable in A_i has the form $X_{p\alpha k}$ for some label p we have $\iota_k(A_i) = \iota_{\alpha k} A_i$ for all $i \in 1..n$. More precisely, in the definition of $\iota_{\alpha k}$ let p be a label such that $X_{p\alpha k}$ is a metavariable in A_i for some $i \in 1..n$, then in the definition of ι_k we take $p' = p\alpha$ and obtain $\iota_k(X_{p'k}) = \iota_{\alpha k}(X_{p\alpha k})$. Hence we may continue as follows:

$$\xi(\iota_{\alpha k} A_1, \ldots, \iota_{\alpha k} A_n) = \xi(\iota_k A_1, \ldots, \iota_k A_n) = \iota_k A$$

- $A = A_1[\![A_2]\!]$. By hypothesis there is an α such that $\mathcal{WF}_{\alpha k}(A_1)$, and $\mathcal{WF}_k(A_2)$. Then

$$\begin{array}{lll}
LHS & = & \mathcal{W}((\kappa(A_1[\![A_2]\!]))[\mathit{lift}^{|k|}(s)]) \\
 & = & \mathcal{W}((\kappa A_1\{\!\{1 \leftarrow \kappa A_2\}\!\})[\mathit{lift}^{|k|}(s)]) \\
 & =_{L.7.19(3)} & \mathcal{W}((\kappa A_1[\mathit{cons}(\kappa A_2, \mathit{id})])[\mathit{lift}^{|k|}(s)]) \\
 & =_{L.7.23} & \mathcal{W}((\kappa A_1)[\mathit{lift}^{|k|+1}(s)][\mathit{cons}((\kappa A_2)[\mathit{lift}^{|k|}(s)], \mathit{id})]) \\
 & = & \mathcal{W}(\mathcal{W}((\kappa A_1)[\mathit{lift}^{|k|+1}(s)])[\mathit{cons}(\mathcal{W}((\kappa A_2)[\mathit{lift}^{|k|}(s)]), \mathit{id})]) \\
 & =_{L.7.19(3)} & \mathcal{W}((\kappa A_1)[\mathit{lift}^{|k|+1}(s)])\{\!\{1 \leftarrow \mathcal{W}((\kappa A_2)[\mathit{lift}^{|k|}(s)])\}\!\} \\
 & =_{i.h.} & \iota_{\alpha k}(A_1)\{\!\{1 \leftarrow \iota_k(A_2)\}\!\} \\
 & = & \iota_k(A_1)\{\!\{1 \leftarrow \iota_k(A_2)\}\!\} \\
 & = & \iota_k(A_1[\![A_2]\!])
\end{array}$$

The before last equality may be justified as in the previous case.

∎

We now verify that the valuation ι from Lemma 7.48 ($k = \epsilon$) is a valid valuation assuming κ is, and hence can be used in rewriting terms. More precisely,

Lemma 7.49 Let κ be a valid valuation for a $SERS_{db}$-rewrite rule (L, R) and let s be any substitution. Then ι is also valid for (L, R), where $\iota(X_l) \stackrel{\text{def}}{=} \mathcal{W}((\kappa X_l)[\mathit{lift}^{|l|}(s)])$ for all X_l in (L, R).

Proof. This follows from the following more general result by considering the case $i = 0$. Let a, b be pure terms. Then for all $i \geq 0$, $\mathit{Value}^i(k_1, a) = \mathit{Value}^i(k_2, b)$ implies $\mathit{Value}^i(k_1, \mathcal{W}(a[\mathit{lift}^{|k_1|+i}(s)])) = \mathit{Value}^i(k_2, \mathcal{W}(b[\mathit{lift}^{|k_2|+i}(s)]))$.

The latter is proved by induction on a. We shall consider the case where a is an index for the other cases follow by using the induction hypothesis. Let $a = n$, we consider three further subcases:

- $n \leq i$. Then $b = n$ and $\mathit{Value}^i(k_1, \mathcal{W}(a[\mathit{lift}^{|k_1|+i}(s)])) = n = \mathit{Value}^i(k_2, \mathcal{W}(b[\mathit{lift}^{|k_2|+i}(s)]))$. The latter holds by Lemma 7.19(1), and therefore, the result holds.

- $i < n \leq |k_1| + i$. Then $b = m$ with $i < m \leq |k_2| + i$ and $\mathtt{at}(k_1, n - i) = \mathtt{at}(k_2, m - i)$. We have $\mathcal{W}(a[\mathit{lift}^{|k_1|+i}(s)]) = n$ and $\mathit{Value}^i(k_2, \mathcal{W}(b[\mathit{lift}^{|k_2|+i}(s)])) = m$, by Lemma 7.19(1). Thus, we have $\mathit{Value}^i(k_1, \mathcal{W}(a[\mathit{lift}^{|k_1|+i}(s)])) = \mathtt{at}(k_1, n - i) = \mathtt{at}(k_2, m - i) = \mathit{Value}^i(k_2, \mathcal{W}(b[\mathit{lift}^{|k_2|+i}(s)]))$.

- $n > |k_1| + i$. Then $b = m$ with $m > |k_2| + i$ and $x_{n-|k_1|-i} = x_{m-|k_2|-i}$. Then we reason as follows:

$$\begin{array}{lll}
\mathcal{W}(a[\mathit{lift}^{|k_1|+i}(s)]) & = & \mathcal{W}(n - |k_1| - i[s][\mathit{shift}]^{|k_1|+i}) \\
 & = & \mathcal{W}(\mathcal{W}(n - |k_1| - i[s])[\mathit{shift}]^{|k_1|+i})
\end{array}$$

And likewise,

$$\begin{array}{lll}
\mathcal{W}(b[\mathit{lift}^{|k_2|+i}(s)]) & = & \mathcal{W}(m - |k_2| - i[s][\mathit{shift}]^{|k_2|+i}) \\
 & = & \mathcal{W}(\mathcal{W}(m - |k_2| - i[s])[\mathit{shift}]^{|k_2|+i})
\end{array}$$

Now $\mathcal{W}(n - |k_1| - i[s]) = \mathcal{W}(m - |k_2| - i[s])$, since $n - |k_1| - i = m - |k_2| - i$.

Observation: for any pure term a, $\mathit{Value}^i(k_1, a) = \mathit{Value}^i(k_2, b)$ implies $\mathit{Value}^i(k_1, \mathcal{W}(a[\mathit{shift}]^{|k_1|+i})) = \mathit{Value}^i(k_2, \mathcal{W}(b[\mathit{shift}]^{|k_2|+i}))$. This may be verified by induction on a and using condition 9 of the definition of a Basic Substitution Calculus (Def. 7.11).

By the observation we may conclude the case.

Definition 7.50 (Parallel $SERS_{db}$-rewriting) Let $\mathcal{R}$ be a $SERS_{db}$ and let a and b be de Bruijn terms. We say that *a $\mathcal{R}$-rewrites in parallel* to b iff $a \rightrightarrows_{\mathcal{R}} b$, where the latter relation is defined as:

$$\frac{}{a \rightrightarrows_{\mathcal{R}} a}\ (\texttt{refl}) \qquad \frac{\kappa \text{ valid for } (L,R) \in \mathcal{R}}{\kappa L \rightrightarrows_{\mathcal{R}} \kappa R}\ (\texttt{red})$$

$$\frac{a_i \rightrightarrows_{\mathcal{R}} b_i \quad \text{for all } 1 \leq i \leq n}{f(a_1,\dots,a_n) \rightrightarrows_{\mathcal{R}} f(b_1,\dots,b_n)}\ (\texttt{clos-f}) \qquad \frac{a_i \rightrightarrows_{\mathcal{R}} b_i \quad \text{for all } 1 \leq i \leq n}{\xi(a_1,\dots,a_n) \rightrightarrows_{\mathcal{R}} \xi(b_1,\dots,b_n)}\ (\texttt{clos-b})$$

Note that $\rightarrow_{\mathcal{R}} \subseteq \rightrightarrows_{\mathcal{R}} \subseteq \twoheadrightarrow_{\mathcal{R}}$, and that $\rightrightarrows_{\mathcal{R}}$ is reflexive. In the case of $\mathcal{R} = \{(L,R)\}$ we shall abbreviate $a \rightrightarrows_{\mathcal{R}} b$ as $a \rightrightarrows_{(L,R)} b$.

Lemma 7.51 Let a, b be pure terms and let (L,R) be a $SERS_{db}$-rewrite rule. If $a \rightrightarrows_{(L,R)} b$ then for any term s of sort S we have $\mathcal{W}(a[s]) \rightrightarrows_{(L,R)} \mathcal{W}(b[s])$.

Proof. By induction on the derivation of $a \rightrightarrows_{(L,R)} b$.

- `refl`. Then the result holds trivially.
- `red`. Let $G = \texttt{pattern}(L)$ (Def. 6.31). Then $a = G[\kappa X^{i_1}_{l_1}]_{X^{i_1}_{l_1}} .. [\kappa X^{i_n}_{l_n}]_{X^{i_n}_{l_n}}$ where $X^{i_1}_{l_1}, \dots, X^{i_n}_{l_n}$ are all the metavariables in L, and κ is a valid valuation for (L,R). Then

$$\mathcal{W}(a[s]) = G[\mathcal{W}((\kappa X^{i_1}_{l_1})[\mathit{lift}^{|l_1|}(s)])]_{X^{i_1}_{l_1}} \dots [\mathcal{W}((\kappa X^{i_n}_{l_n})[\mathit{lift}^{|l_1|}(s)])]_{X^{i_n}_{l_n}}$$

  So define $\iota X^{i_j}_{l_j} \stackrel{\text{def}}{=} \mathcal{W}((\kappa X^{i_j}_{l_j})[\mathit{lift}^{|l_j|}(s)])$. Then since ι is valid for (L,R) by Lemma 7.49, an application of `red` allows us to conclude: $\mathcal{W}(a[s]) \rightrightarrows_{(L,R)} \iota(R) =_{L.\ 7.48(k=\epsilon)} \mathcal{W}((\kappa R)[s]) = \mathcal{W}(b[s])$.
- `clos-f`. Then by the induction hypothesis we have $\mathcal{W}(a_i[s]) \rightrightarrows_{(L,R)} \mathcal{W}(b_i[s])$ for all $1 \leq i \leq n$. We conclude using `clos-f` $\mathcal{W}(f(a_1,\dots,a_n)[s]) = f(\mathcal{W}(a_1[s]),\dots,\mathcal{W}(a_n[s])) \rightrightarrows_{(L,R)} f(\mathcal{W}(b_1[s]),\dots,\mathcal{W}(b_n[s])) = \mathcal{W}(f(b_1,\dots,b_n)[s])$.
- `clos-b`. As in the case `clos-f`.

∎

Note that in particular Lemma 7.51 holds when $a \rightarrow_{(L,R)} b$ since the one-step rewrite relation is included in the parallel rewrite relation.

Lemma 7.52 (Projecting assignments) Let (L,R) be a $SERS_{db}$-rewrite rule, $(L',R') = \mathcal{C}_P(L,R)$ for some pivot set P for (L,R), let ρ be an assignment for (L',R').

Define the valuation κ as:

$$\kappa X_k \stackrel{\text{def}}{=} \mathcal{W}(\overline{\rho}(\mathcal{C}^{(L,R)}_P(X_k)))$$

If $L = C[A]$ for some metacontext C and metaterm A, then $\mathcal{W}(\overline{\rho}(\mathcal{C}^{(L,R)}_P(A))) = \kappa A$. Likewise, if $R = C[A]$ then $\mathcal{W}(\overline{\rho}(\mathcal{C}^{(L,R)}_P(A))) = \kappa A$.

Proof. Both items are proved by induction on A.

- $A = n$. Then $LHS = \mathcal{W}(\overline{\rho}(n)) = n = \kappa n = RHS$.
- $A = X_k$. Then $LHS = \mathcal{W}(\overline{\rho}(\mathcal{C}^{(L,R)}_P(X_k))) =_{\mathit{hypothesis}} \kappa X_k$
- $A = f(A_1,\dots,A_n)$. Then

$$LHS =^{Def.\ 7.11(3)} f(\mathcal{W}(\overline{\rho}(\mathcal{C}^{(L,R)}_P(A_1))),\dots,\mathcal{W}(\overline{\rho}(\mathcal{C}^{(L,R)}_P(A_n)))) =^{i.h.}_{\mathcal{W}} f(\kappa A_1,\dots,\kappa A_n) = RHS$$

- $A = \xi(A_1,\dots,A_n)$. Then

$$LHS =^{Def.\ 7.11(3)} \xi(\mathcal{W}(\overline{\rho}(\mathcal{C}^{(L,R)}_P(A_1))),\dots,\mathcal{W}(\overline{\rho}(\mathcal{C}^{(L,R)}_P(A_n)))) =^{i.h.}_{\mathcal{W}} \xi(\kappa A_1,\dots,\kappa A_n) = RHS$$

- $A = A_1[\![A_2]\!]$. This case is considered for the second item only since the de Bruijn metasubstitution operator may not occur on the *LHS* of a $SERS_{db}$-rewrite rule.

$$\begin{array}{lll}
LHS & = & \mathcal{W}(\overline{\rho}(C_P^{(L,R)}(A_1[\![A_2]\!]))) \\
 & = & \mathcal{W}(\overline{\rho}(C_P^{(L,R)}(A_1))[cons(\overline{\rho}(C_P^{(L,R)}(A_2)), id)]) \\
 & =^{Def.\ 7.11(1)} & \mathcal{W}(\mathcal{W}(\overline{\rho}(C_P^{(L,R)}(A_1)))[cons(\mathcal{W}(\overline{\rho}(C_P^{(L,R)}(A_2))), id)]) \\
 & =^{i.h.}_{\mathcal{W}} & \mathcal{W}(\kappa A_1[cons(\kappa A_2, id)]) \\
 & =_{L.\ 7.19(3)} & \kappa A_1\{\!\{1 \leftarrow \kappa A_2\}\!\} \\
 & = & \kappa(A_1[\![A_2]\!]) \\
 & = & RHS
\end{array}$$

In order to use the valuation of Lemma 7.52 we need to prove that it is valid. This is the issue of Lemma 7.53 and Corollary 7.54.

Lemma 7.53 Consider a $SERS_{db}$-rewrite rule (L, R), metavariables X_{k_1}, X_{k_2} occurring in (L, R) and a designated pivot metavariable X_l. Let a be any pure term. Then for all $i \geq 0$ we have:

$$Value^i(k_1, \mathcal{W}(a[lift^i(s_1)])) = Value^i(k_2, \mathcal{W}(a[lift^i(s_2)]))$$

where $s_1 = cons(b_1, \ldots, b_{|l|}, shift^{|k_1|+|l \setminus \mathtt{Ba}_{(L,R)}(X)|})$ and $s_2 = cons(c_1, \ldots, c_{|l|}, shift^{|k_2|+|l \setminus \mathtt{Ba}_{(L,R)}(X)|})$ are the index-adjusting substitutions (using pivot X_l) of X_{k_1} and X_{k_2}, respectively.

Proof. We shall assume that $X_{k_1} \neq X_l$ and $X_{k_2} \neq X_l$. The case where $X_{k_1} = X_l$ or $X_{k_2} = X_l$ is analogous. We proceed by induction on a.

- $a = n$. We have three subcases to consider.
  - $n \leq i$. Then by Lemma 1 $Value^i(k_1, n) = n = Value^i(k_2, n)$.
  - $i < n \leq |l| + i$. Now we consider two further cases:
    - $n - i = \mathrm{pos}(\beta_h, l)$ for some $\beta_h \in \mathtt{Ba}_{(L,R)}(X)$. Then $b_{n-i} = \mathrm{pos}(\beta_h, k_1)$ and $c_{n-i} = \mathrm{pos}(\beta_h, k_2)$ by Def. 7.30. Therefore $Value^i(k_1, b_{n-i} + i) = \beta_h = Value^i(k_2, c_{n-i} + i)$.
    - There is no $\beta_h \in \mathtt{Ba}_{(L,R)}(X)$ such that $n - i = \mathrm{pos}(\beta_h, l)$. Then $b_{n-i} = |k_1| + 1 + \mathtt{Sh}(X_l, n-i)$ and $c_{n-i} = |k_2| + 1 + \mathtt{Sh}(X_l, n-i)$. Hence, $Value^i(k_1, b_{n-i} + i) = x_{1+\mathtt{Sh}(X_l, n-i)} = Value^i(k_2, c_{n-i} + i)$.
  - $n > |l| + i$. Then $\mathcal{W}(a[lift^i(s_1)]) = n - |l| + |k_1| + |l \setminus \mathtt{Ba}_{(L,R)}(X)|$ and $\mathcal{W}(a[lift^i(s_2)]) = n - |l| + |k_2| + |l \setminus \mathtt{Ba}_{(L,R)}(X)|$. Thus $Value^i(k_1, n - |l| + |k_1| + |l \setminus \mathtt{Ba}_{(L,R)}(X)|) = x_{n-i-|l|+|l \setminus \mathtt{Ba}_{(L,R)}(X)|} = Value^i(k_2, n - |l| + |k_2| + |l \setminus \mathtt{Ba}_{(L,R)}(X)|)$.
- $a = f(a_1, \ldots, a_n)$. Then

$$\begin{array}{ll}
 & Value^i(k_1, \mathcal{W}(a[lift^i(s_1)])) \\
= & Value^i(k_1, f(\mathcal{W}(a_1[lift^i(s_1)]), \ldots, \mathcal{W}(a_n[lift^i(s_1)]))) \\
= & f(Value^i(k_1, \mathcal{W}(a_1[lift^i(s_1)])), \ldots, Value^i(k_1, \mathcal{W}(a_n[lift^i(s_1)]))) \\
=_{ih} & f(Value^i(k_2, \mathcal{W}(a_1[lift^i(s_2)])), \ldots, Value^i(k_2, \mathcal{W}(a_n[lift^i(s_2)]))) \\
= & Value^i(k_2, \mathcal{W}(a[lift^i(s_2)]))
\end{array}$$

- $a = \xi(a_1, \ldots, a_n)$. Similar to the previous case.

Corollary 7.54 (From assignments to valid valuations) Consider a $SERS_{db}$-rewrite rule (L, R), metavariables X_{k_1}, X_{k_2} occurring in (L, R) and a designated pivot metavariable X_l. Let ρ be an assignment. Then

$$Value(k_1, \mathcal{W}(\overline{\rho}(X)[s_1])) = Value(k_2, \mathcal{W}(\overline{\rho}(X)[s_2]))$$

where $s_1 = cons(b_1, \ldots, b_{|l|}, shift^{|k_1|+|l \setminus \mathtt{Ba}_{(L,R)}(X)|})$ and $s_2 = cons(c_1, \ldots, c_{|l|}, shift^{|k_2|+|l \setminus \mathtt{Ba}_{(L,R)}(X)|})$ are the index-adjusting substitutions (using pivot X_l) of X_{k_1} and X_{k_2}, respectively.

Proof. Since $\mathcal{W}$ is a basic substitution calculus it has unique normal forms. Also, by condition 2 of Def. 7.11 we have that $\mathcal{W}(a)$ is a pure term for any term a. Thus

$$\begin{array}{rcl} \mathit{Value}(k_1, \mathcal{W}(\overline{\rho}(X)[s_1])) & = & \mathit{Value}(k_1, \mathcal{W}(\mathcal{W}(\overline{\rho}(X))[s_1])) \\ & = & \mathit{Value}^0(k_1, \mathcal{W}(\mathcal{W}(\overline{\rho}(X))[s_1])) \\ & =_{L\ 7.53} & \mathit{Value}^0(k_2, \mathcal{W}(\mathcal{W}(\overline{\rho}(X))[s_2])) \\ & = & \mathit{Value}(k_2, \mathcal{W}(\mathcal{W}(\overline{\rho}(X))[s_2])) \\ & = & \mathit{Value}(k_2, \mathcal{W}(\overline{\rho}(X)[s_2])) \end{array}$$

■

Lemma 7.55 (Interpretation of *ExERS*-rewriting) Let $\mathcal{W}$ be a basic substitution calculus satisfying the scheme. Let o be a term of $\mathcal{W}$ of sort $\mathtt{T}$ or $\mathtt{S}$. Let $(L', R') = \mathcal{C}_P(L, R)$. If $o \rightarrow_{(L',R')} o'$ then

1. if o is of sort $\mathtt{T}$ then $\mathcal{W}(o) \rightrightarrows_{(L,R)} \mathcal{W}(o')$.
2. for every pure term d of sort $\mathtt{T}$ such that $d[o]$ is a term of sort $\mathtt{T}$, and every $n \geq 0$, $\mathcal{W}(d[\mathit{lift}^n(o)]) \rightrightarrows_{(L,R)} \mathcal{W}(d[\mathit{lift}^n(o')])$.

Proof. We show simultaneously the two items by induction on the lexicographic ordering (o, d), where o and d denote, respectively, the ordering induced by their structures.

- o is a de Bruijn index or a substitution constant. Then both items holds vacuously since by definition the *LHS* of a SERS_{db}-rewrite rule must have a function or binder symbol as head symbol. Thus o is a normal form.

- $o = f(a_1, \ldots, a_n)$ or $o = \xi(a_1, \ldots, a_n)$. There is nothing to prove for the second item. For the first:

  - the reduction is at the root. Then $o = \rho L'$. Define κ for all $X_k \in L$ as:

    $$\kappa X_k \stackrel{\text{def}}{=} \mathcal{W}(\overline{\rho}(\mathcal{C}_P^{(L,R)}(X_k)))$$

    Note that κ is a valid valuation by Corollary 7.54, and also, $\mathcal{W}(\rho L') = \kappa L$ by Lemma 7.52. So $\kappa L \rightrightarrows_{(L,R)} \kappa R =_{L.\ 7.52} \mathcal{W}(\rho R') = \mathcal{W}(o')$.
  - the reduction is internal. Then we use the induction hypothesis.

- $o = a[s]$. There is nothing to prove for the second item. We consider two cases for the first property:

  - $o' = a'[s]$ with $a \rightarrow_{(L',R')} a'$. By the i.h. $\mathcal{W}(a) \rightrightarrows_{(L,R)} \mathcal{W}(a')$. Then $\mathcal{W}(a[s]) = \mathcal{W}(\mathcal{W}(a)[s]) \rightrightarrows_{(L,R)} \mathcal{W}(\mathcal{W}(a')[s])$ by applying Lemma 7.51.
  - $o' = a[s']$ with $s \rightarrow_{(L',R')} s'$. Since $\mathcal{W}(a)$ is a pure term we have that $\mathcal{W}(o) = \mathcal{W}(\mathcal{W}(a)[s]) \rightrightarrows_{(L,R)} \mathcal{W}(\mathcal{W}(a)[s']) = \mathcal{W}(o')$ by the induction hypothesis of item 2 since $(s, \mathcal{W}(a)) < (a[s], \mathcal{W}(a))$.

- o is a substitution $\sigma(s_1, \ldots, s_j, \ldots, s_q)$ $(q > 0)$, and $o' = \sigma(s_1, \ldots, s'_j, \ldots, s_q)$, where $s_j \rightarrow_{(L',R')} s'_j$, then there is nothing to prove for the first property since o is not a term. For the second property we proceed by induction on d.

  - $d = f(d_1, \ldots, d_n)$ or $d = \xi(d_1, \ldots, d_n)$ then the property holds by the induction hypothesis since $(o, d_i) < (o, d)$ for all $1 \leq i \leq n$, and applying `clos-f` or `clos-b`.
  - $d = m$. Then we must verify that for all $n \geq 0$: $\mathcal{W}(m[\mathit{lift}^n(o)]) \rightrightarrows_{(L,R)} \mathcal{W}(m[\mathit{lift}^n(o')])$.

    We proceed by induction on n.

    1. if $n = 0$, then we proceed by cases as dictated by the definition of the scheme (Def. 7.17).
       (a) Suppose there exists a de Bruijn index r, indices $i_1, \ldots, i_p$ $(p > 0)$ and also substitutions $u_1, \ldots, u_k$ $(k \geq 0)$ such that $1 \leq i_1, \ldots, i_p \leq q$, the i_j's are all distinct and for all $s_1 \ldots s_q$ $m[\sigma(s_1, \ldots s_q)] =_{\mathcal{W}} r[s_{i_1}] \ldots [s_{i_p}][u_1] \ldots [u_k]$.
          i. if $j \notin \{i_1, \ldots, i_p\}$, then $\mathcal{W}(m[o'])$ is also equal to the term $\mathcal{W}(r[s_{i_1}] \ldots [s_{i_p}][u_1] \ldots [u_k])$ and the property is trivial since $\mathcal{W}(m[o]) = \mathcal{W}(m[o'])$.

ii. if $j \in \{i_1, \dots, i_p\}$, let us say $j = i_h$, then the term $\mathcal{W}(m[o'])$ is equal to the term $\mathcal{W}(r[s_{i_1}] \dots [s'_{i_h}] \dots [s_{i_p}][u_1] \dots [u_k])$ and $\mathcal{W}(r[s_{i_1}] \dots [s_{i_{h-1}}]) = e$ is a pure term (Def. 7.11(2)). We have $(s_{i_h}, e) < (\sigma(s_1, \dots, s_j, \dots, s_q), m)$, so that we can apply the induction hypothesis (2) to obtain:

$$\mathcal{W}(e[s_{i_h}]) \rightrightarrows_{(L,R)} \mathcal{W}(e[s'_{i_h}])$$

Now, the term $\mathcal{W}(e[s_{i_h}])$ is pure, so that we can repeatedly apply Lemma 7.51 to obtain:

$$\begin{array}{lll} \mathcal{W}(m[o]) & = & \mathcal{W}(\mathcal{W}(e[s_{i_h}])[s_{i_{h+1}}] \dots [s_{i_p}][u_1] \dots [u_k]) \\ & \rightrightarrows_{(L,R)} & \mathcal{W}(\mathcal{W}(e[s_{i_h}])[s_{i_{h+1}}] \dots [s_{i_p}][u_1] \dots [u_k]) \\ & = & \mathcal{W}(m[o']) \end{array}$$

(b) Suppose there exists an index i with $1 \leq i \leq q$ such that for all $s_1 \dots s_q$ we have that $m[\sigma(s_1, \dots s_q)] =_{\mathcal{W}} s_i$.

i. if $i \neq j$, then the term $\mathcal{W}(m[o'])$ is also equal to $\mathcal{W}(s_i)$ and the property is trivial since $\mathcal{W}(m[o]) = \mathcal{W}(m[o'])$.

ii. if $i = j$, then $\mathcal{W}(m[o']) = \mathcal{W}(s'_j)$ (where s_j is a term because the equations are well-typed) and $(s_j, m) < (\sigma(s_1, \dots, s_j, \dots, s_q), m)$, so the property holds by the induction hypothesis (1) since $\mathcal{W}(m[o]) = \mathcal{W}(s_j) \rightrightarrows_{(L,R)} \mathcal{W}(s'_j) = \mathcal{W}(m[o'])$.

2. if $n > 0$, then we consider two cases:

(a) if $m \leq n$, then by Lemma 7.19(1) we obtain:

$$\mathcal{W}(m[\mathit{lift}^n(o)]) = m = \mathcal{W}(m[\mathit{lift}^n(o')])$$

(b) if $m > n$, then by Lemma 7.19(1) we obtain:

$$\mathcal{W}(m[\mathit{lift}^n(o)]) = \mathcal{W}(m - 1[\mathit{lift}^{n-1}(o)][\mathit{shift}])$$

and

$$\mathcal{W}(m[\mathit{lift}^n(o')]) = \mathcal{W}(m - 1[\mathit{lift}^{n-1}(o')][\mathit{shift}])$$

Since variables are equivalent with respect to our ordering (o, d), $(\sigma(s_1, \dots, s_j, \dots, s_q), m) = (\sigma(s_1, \dots, s_j, \dots, s_q), m-1)$, and then the induction hypothesis on n can be applied to obtain

$$\mathcal{W}(m - 1[\mathit{lift}^{n-1}(o)]) \rightrightarrows_{(L,R)} \mathcal{W}(m - 1[\mathit{lift}^{n-1}(o')])$$

Since every $\mathcal{W}$-normal form is a pure term by Def. 7.11(2), we may finally apply Lemma 7.51, so that

$$\begin{array}{lll} \mathcal{W}(m[\mathit{lift}^n(o)]) & = & \mathcal{W}(\mathcal{W}(m - 1[\mathit{lift}^{n-1}(o)])[\mathit{shift}]) \\ & \rightrightarrows_{(L,R)} & \mathcal{W}(\mathcal{W}(m - 1[\mathit{lift}^{n-1}(o')])[\mathit{shift}]) \\ & = & \mathcal{W}(m[\mathit{lift}^n(o')]) \end{array}$$

•

Proposition 7.56 (Projection Proposition) Let $\mathcal{R}$ be a *SERS$_{db}$* and let $fo(\mathcal{R})_{\mathcal{W}}$ be its first-order version where $\mathcal{W}$ is a basic substitution calculus satisfying the scheme. If $a \rightarrow_{fo(\mathcal{R})_{\mathcal{W}}} b$ then $\mathcal{W}(a) \rightrightarrows_{\mathcal{R}} \mathcal{W}(b)$.

Proof. We consider two cases, one for *ExERS*-rewriting and one for *FExERS*-rewriting.

***ExERS*-rewriting** Suppose that $a \rightarrow_{fo(\mathcal{R})/\mathcal{W}} b$ using rewrite rule $(L', R') = \mathcal{C}_P(L, R)$ where P is a pivot set for $(L, R) \in \mathcal{R}$, a context E and assignment ρ. Thus $a =_{\mathcal{W}} E[\rho(L')]$ and $b =_{\mathcal{W}} E[\rho(R')]$.

Now since $E[\rho(L')] \rightarrow_{(L',R')} E[\rho(R')]$ then by Lemma 7.55 we may conclude that $\mathcal{W}(E[\rho(L')]) \rightrightarrows_{(L,R)} \mathcal{W}(E[\rho(R')])$. Also since $a =_{\mathcal{W}} E[\rho(L')])$ we know that $\mathcal{W}(a) = \mathcal{W}(E[\rho(L')])$, likewise we know that $\mathcal{W}(b) = \mathcal{W}(E[\rho(R')])$. Therefore $\mathcal{W}(a) = \mathcal{W}(E[\rho(L')]) \rightrightarrows_{(L,R)} \mathcal{W}(E[\rho(R')]) = \mathcal{W}(b)$ as desired.

***FExERS*-rewriting** Suppose $fo(\mathcal{R})$ is a *FExERS* and that $a \rightarrow_{fo(\mathcal{R}) \cup \mathcal{W}} b$. Then if $a \rightarrow_{\mathcal{W}} b$ the result holds trivially. Thus let us assume that $a \rightarrow_{fo(\mathcal{R})} b$ using rewrite rule $(L', R') = \mathcal{C}_P(L, R)$ where P is a pivot set for $(L, R) \in \mathcal{R}$, a context E and assignment ρ. Then $a = E[\rho(L')]$ and $b = E[\rho(R')]$. Now since $E[\rho(L')] \rightarrow_{(L',R')} E[\rho(R')]$ then by Lemma 7.55 we may conclude that $\mathcal{W}(a) = \mathcal{W}(E[\rho(L')]) \rightrightarrows_{(L,R)} \mathcal{W}(E[\rho(R')]) = \mathcal{W}(b)$ as desired.

Since $\rightrightarrows_{\mathcal{R}} \subseteq \twoheadrightarrow_{\mathcal{R}}$, we may replace $\mathcal{W}(a) \rightrightarrows_{\mathcal{R}} \mathcal{W}(b)$ by $\mathcal{W}(a) \twoheadrightarrow_{\mathcal{R}} \mathcal{W}(b)$ in the statement of the Projection Proposition.

Definition 7.57 Let R and S be binary relations defined over sets A and B with $A \subseteq B$, respectively. We say S is *conservative over* R if aSb implies aRb for all $a \in A$.

Noting that $\mathcal{W}(a) = a$ for pure terms a (Def. 7.11(2)) we may conclude.

Corollary 7.58 (Conservativity) Let $\mathcal{R}$ be a *SERS*$_{db}$. Then $fo(\mathcal{R})_{\mathcal{W}}$-rewriting is conservative over $\mathcal{R}$-rewriting.

7.4.3 Essentially First-Order HORS

This last subsection gives a very simple syntactical criterion that can be used to decide if a given higher-order rewrite system can be translated into a full first-order rewrite system (modulo an empty equational theory). In particular, we can check that many higher-order calculi in the literature, such as the lambda calculus, verify this property.

Definition 7.59 (Essentially first-order HORS) A *SERS*$_{db}$ $\mathcal{R}$ is called *essentially first-order* if $fo(\mathcal{R})_{\mathcal{W}}$ is a *FExERS* for $\mathcal{W}$ a basic substitution calculus.

Definition 7.60 (fo-condition) A *SERS*$_{db}$ $\mathcal{R}$ satisfies the *fo-condition* if every rewrite rule $(L, R) \in \mathcal{R}$ satisfies: for every name X in L let $X_{l_1}, \ldots, X_{l_n}$ be all the X-based metavariables in L, then

1. $l_1 = l_2 \ldots = l_n$ and (the underlying set of) l_1 is $\mathtt{Ba}_{(L,R)}(X)$, and
2. for all $X_k \in R$ we have $|k| \geq |l_1|$.

In the above definition note that $l_1 = l_2 \ldots = l_n$ means that labels $l_1, \ldots, l_n$ must be *identical* (for example $\alpha\beta \neq \beta\alpha$). Also, by Def. 6.36, l_1 is simple, in other words, it does not have repeated elements.

Example 7.61 Consider the λ_{db}-calculus consisting of the sole rule: $app(\lambda X_\alpha, Y_\epsilon) \to_{\beta_{db}} X_\alpha[Y_\epsilon]$. The β_{db}-calculus satisfies the fo-condition.

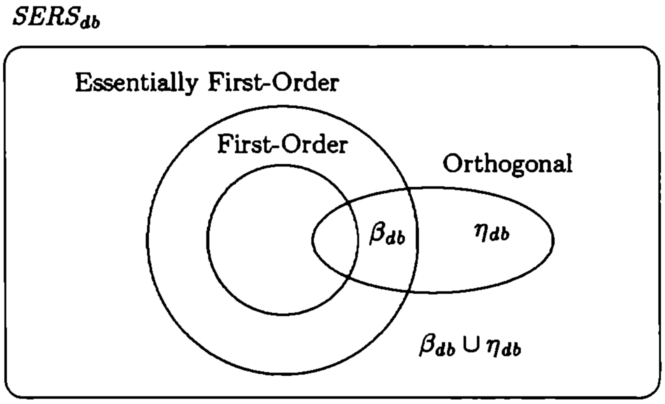
Proposition 7.62 puts forward the importance of the fo-condition. Its proof relies on a close inspection of the Conversion Procedure.

Proposition 7.62 Let $\mathcal{R}$ be a *SERS*$_{db}$ satisfying the fo-condition. Then $\mathcal{R}$ is essentially first-order.

Further examples of essentially first-order *SERS*$_{db}$ are the *foldl*-rewrite system of Example 7.33 and the natural numbers recursor rewrite system *rec* of Example 7.34.

Note that many results on higher-order systems (e.g. perpetuality [KOO01a], standardization [Mel96]) require *left-linearity* (a metavariable may occur at most once on the *LHS* of a rewrite rule), and *fully-extendedness or locality* (if a metavariable $X(t_1, \ldots, t_n)$ occurs on the *LHS* of a rewrite rule then $t_1, \ldots, t_n$ is the list of variables bound above it). The reader may find it interesting to observe that these conditions together seem to imply the fo-condition. A proof of this fact would require either developing the results of this work in the above mentioned HORS or via some suitable translation to the *SERS*$_{db}$ formalism, and is left to future work.

Of course, all first-order rewrite systems are essentially first-order *SERS*$_{db}$: indeed all metavariables in first-order rewrite systems carry ϵ as label. Hence the latter systems need not be left-linear. Also, an orthogonal *SERS*$_{db}$ (Def. 7.1) need not be essentially first-order, the prime example of this fact being the rewrite system consisting of the sole rule η_{db}. Below we picture this situation.



Informally, it seems fair to say that a $SERS_{db}$ system is essentially first-order if higher-order pattern matching may be reduced to syntactic first-order matching. We claim that essentially first-order $SERS_{db}$ systems are appropriate for transferring results from first-order systems. As evidence of our claim we shall undertake the task of transferring a non-trivial property of (left-linear) first-order rewrite systems to this class of higher-order rewrite systems, namely the Standardization Theorem. This shall constitute the focus of our attention in Chapter 8.

However, before considering the Standardization Theorem the transfer of a more basic property presents itself, namely the *Critical Pair Theorem* (CPT): a first-order rewrite system is locally confluent iff all its critical pairs are joinable [BN98, Th.6.2.4]. This result is useful for proving that a finite and strongly normalizing TRS is confluent since by Newman's Lemma we only have to check that all critical pairs are joinable. One may obtain rather immediately the following:

Proposition 7.63 (Transfer of the CPT) Let $\mathcal{R}$ be an essentially first-order $SERS_{db}$. If all the critical pairs of $fo(\mathcal{R})_{\mathcal{W}}$ are joinable then $\mathcal{R}$ locally confluent.

Proof. Suppose $A \rightarrow_{r_1} B_1$ and $A \rightarrow_{r_2} B_2$, where r_1, r_2 are rewrite rules in $\mathcal{R}$. Then by the Simulation Proposition $A \rightarrow_{fo(r_1)} B'_1 \twoheadrightarrow_{\mathcal{W}} B_1$ and $A \rightarrow_{fo(r_2)} B'_2 \twoheadrightarrow_{\mathcal{W}} B_2$ as depicted in Figure 7.1. Then 1 may be completed by hypothesis (we use CPT for first-order rewriting and the fact that all critical pairs of $fo(\mathcal{R})_{\mathcal{W}}$ are joinable). Note that since $\mathcal{R}$ is essentially first-order then it is the usual first-order critical pairs that we are referring to. The items marked 2 follow from the Projection Proposition and the fact that $\mathcal{W}$ implements metalevel substitution (Lemma 7.19(3)).

$$\begin{tikzcd}
 & & A \arrow[dl, "fo(r_1)"'] \arrow[dr, "fo(r_2)"] & & \\
 & B'_1 \arrow[dl, two heads, "\mathcal{W}"'] \arrow[dr, dotted, two heads, "fo(\mathcal{R})_{\mathcal{W}}"'] & 1 & B'_2 \arrow[dr, two heads, "\mathcal{W}"] \arrow[dl, dotted, two heads, "fo(\mathcal{R})_{\mathcal{W}}"] & \\
B_1 \arrow[drr, two heads, "\mathcal{R}"'] & 2 & C' \arrow[d, two heads, "\mathcal{W}"'] & 2 & B_2 \arrow[dll, two heads, "\mathcal{R}"] \\
 & & C & &
\end{tikzcd}$$

Figure 7.1: Transferring the CPT

So let us now move on to the Standardization Theorem.

Chapter 8

Transferring Standardization

The Conversion Procedure (Def. 7.35) is of interest from at least two perspectives. From the *expressability perspective* it establishes how higher-order rewriting may be encoded as first-order rewriting modulo an equational theory, and it moreover characterizes a subclass of $SERS_{db}$ for which the equational theory is empty. From the *practical perspective* it opens up the possibility of transferring results from the first-order framework to the higher-order one. This chapter attempts to pursue the latter perspective in more detail. In full precision, we shall study how to lift the *Standardization Theorem* from first-order rewriting to higher-order. Before plunging ourselves into such a task we provide the reader with an informal discussion on standardization and then consider a more detailed (although still brief) survey.

When studying rewrite systems the Standardization Theorem allows one to single out certain *canonical* derivations in the class of a larger set of derivations. This is useful since many properties dealing with derivations may then focus their attention on the canonical ones. In particular, standardization is a convenient tool when considering normalizing rewrite strategies. For instance, given some term M in a rewrite system, it may have a normal form while at the same time admit infinite derivations. As an example, consider the λ-term $M = Kc(\Delta\Delta)$ where $K = \lambda x.\lambda y.x$ and $\Delta = \lambda x.xx$. Then M admits the normal form c:

$$Kc(\Delta\Delta) \to_\beta (\lambda y.c)(\Delta\Delta) \to_\beta c$$

However, we may also reduce the β-redex $\Delta\Delta$ in M, obtaining the same term, hence repeating the process we may obtain a derivation of arbitrary length. Assuming we are interested in obtaining a normal form from M we would like to have some strategy at our disposal indicating which redex in M to contract next in order to achieve our goal. Avoiding the nondeterminism caused by a term with more than one redex is of particular importance when implementing rewrite systems on a computer. A reasonable normalizing strategy for M, as suggested by the above mentioned example, is to contract the leftmost redex. So what is a standard derivation and how does it relate to this normalizing strategy? A standard derivation is one in which redex contraction takes place in a left to right direction. Once a β-redex has been contracted in a term M, all the (residuals of) β-redexes to the left of M are forbidden to be contracted in subsequent rewrite steps. The latter β-redexes thus remain 'frozen' for the rest of the derivation. In 1958 the first standardization theorem was proved by H.B.Curry and R.Feys [CF58] for the λ-calculus: for every β-derivation from a term M to a term N there exists a *standard* β-derivation from M to N. As a consequence of this result, it was shown that the leftmost rewrite strategy is normalizing for all terms. Indeed, a standard derivation from a term M to a term N in β-normal form must be a leftmost derivation for otherwise there would be some frozen redex which is present in N.

In first-order rewriting one would like similar results to hold. Since non-orthogonal rewrite systems may not be confluent the fundamental results on standardization began with the class of *orthogonal* (left-linear and non-overlapping) systems. The major question was how to extend the notion of standard derivations to the first-order case. For a restricted class of orthogonal systems (the *left-normal* systems) a straightforward generalization of the notion of standard derivation of the λ-calculus is available. An orthogonal first-order rewrite system is left-normal if all function symbols occur to the left of all variables in every *LHS* of a rule. For this class of systems the standardization theorem holds, and moreover, the leftmost-outermost rewrite strategy may be shown to normalize any term [O'D77]. To see that the leftmost-outermost strategy may fail to normalize for non-left-normal systems consider the following example. Let $\mathcal{R} = \{f(X,a) \to b,\ c \to c,\ d \to a\}$. Note that the constant a occurs to the right of the variable X in the *LHS* of the first rewrite-rule, hence $\mathcal{R}$ is not left-normal. There is a derivation from the term $f(c,d)$ to a normal form: $f(c,d) \to f(c,a) \to b$ yet the leftmost-outermost strategy leads to an infinite derivation: $f(c,d) \rightarrow f(c,d) \rightarrow \ldots$. A strategy that normalizes all terms in any orthogonal first-order term rewrite system is the parallel-outermost rewrite strategy [O'D77, Klo92].

A generalization of the notion of standard derivations for *all* orthogonal systems was presented by G.Huet and J-J. Lévy [HL79]. They show that the Standardization Theorem holds for all orthogonal systems. Normalizing strategies may be obtained from this result, by contracting, so called *needed* redexes. However, determining if a redex is needed or not is not decidable in general, so a subclass of orthogonal systems was also considered, the ***strongly sequential systems***, for which efficient normalizing strategies are exhibited.

This notion of standard derivations was later extended to non-orthogonal left-linear first-order rewrite systems by G.Boudol [Bou85]. Also, work by J.W.Klop [Klo80] has allowed a simpler formulation of the notion of standard derivation, based on rewriting derivations.

Finally, we consider the case of higher-order rewriting. Just as in the first-order case, a straightforward generalization of the notion of standard derivation of the λ-calculus may be obtained for the class of left-normal orthogonal *CRS*. For this notion of standard derivations the Standardization Theorem holds, and normalization of the leftmost-outermost rewrite strategy too [Klo80]. For the full class of orthogonal systems the Standardization Theorem also holds by providing a definition of standard derivation based on rewriting derivations (an idea apparently due initially to J.W.Klop) [Oos96]. It should also be mentioned that work on standardization in *Axiomatic Reduction Systems* allows one to obtain the standardization theorem for left-linear fully-extended (called *local* in [Mel96]) higher-order rewrite systems (hence not necessarily orthogonal) [GLM92, Mel96]. In [Oos96] standardisation of second and higher-order rewriting are related: each *HRS* [Nip91]-rewrite step is decomposed into a replacement step (in which no substitution takes place) followed by β-rewrite steps to β-normal form. It is claimed that by decomposing an *HRS*-derivation, standardizing the decomposed derivation and then 'projecting' back a standard *HRS*-derivation is obtained. Our work relates first and second-order rewriting.

We shall prove the standardization theorem for the class of (left-linear) essentially first-order higher-order term rewrite systems (Def. 7.59). Our contribution is not in the standardization theorem itself since this may be obtained, for example, as an instance of the axiomatic standardization theorem of P-A.Melliès [Mel96] but rather by the proof method we use: we shall *transfer* the standardization theorem from the first-order setting to the higher-order setting. This is put forward as evidence that, for the class of essentially first-order higher-order rewrite systems, techniques developed for the first-order setting are applicable. Further evidence of this fact, as already mentioned in Chapter 7, is available: a number of results proved for higher-order systems require that certain conditions be imposed on the higher-order formalism in question (for example, *CRS* [Mel96] or *HRS* [KOO01a]) for the proofs to go through; it so happens that these conditions force, as it seems, the resulting class of restricted higher-order systems to be, what we have called in the $SERS_{db}$ framework, essentially first-order systems.

It may be observed that a further benefit of our result is the possibility of applying the theory of needed derivations for non-orthogonal left-linear systems, as developed in [Mel96, Mel00], in order to show that all needed derivations are normalizing for calculi of explicit substitution implementing *any* orthogonal essentially higher-order rewrite system (and not just for $\lambda\sigma$ as shown in [Mel00]). This is left to future work.

The transfer of standardization is achieved by applying ideas due to P-A.Melliès. In [Mel00] P-A.Melliès shows the following:

Proposition 8.1 (Melliès) Every standard derivation $v : M \twoheadrightarrow N$ in $\lambda\sigma$ with N in σ-normal form is projected[1] onto a standard derivation $\sigma(v) : \sigma(M) \twoheadrightarrow N$ in the $\lambda\beta$-calculus.

We shall show that in fact this not only holds for the $\lambda\beta$-calculus, which lives comfortably in the class of essentially first-order higher-order rewrite systems, but for the *whole* class.

Proposition 8.2 Let $\mathcal{R}$ be a left-linear essentially first-order $SERS_{db}$. Every standard derivation $v : M \twoheadrightarrow N$ in $fo(\mathcal{R})_\sigma$ with N in σ-normal form is projected onto a standard derivation $\sigma(v) : \sigma(M) \twoheadrightarrow N$ in $\mathcal{R}$.

The resulting standardization procedure for standardizing a $SERS_{db}$-derivation Υ consists in:

1. 'Implement' the $SERS_{db}$-derivation Υ as a *FExERS*-derivation v (see Figure 8.1) by means of the Simulation Proposition (Proposition 7.46).

2. Apply first-order standardization on v [Bou85] yielding a standard (first-order) derivation ϕ.

[1] See Section 8.3 for the definition of the projection of a derivation.

3. Project the derivation ϕ by means of the Projection Proposition (Proposition 7.56) obtaining $\sigma(\phi)$. Use Proposition 8.2 to conclude that $\sigma(\phi)$ is a standard derivation in the higher-order framework.

In fact, we shall take a small step further and prove that Υ and $\sigma(\phi)$ are Lévy permutation equivalent, when v and ϕ are.

$$\begin{tikzcd}
\Upsilon \arrow[d, "\text{Simulation}"'] & \sigma(\phi) \\
v \arrow[r, squiggly, "\text{first-order standardization}"'] & \phi \arrow[u, "\text{Projection}"']
\end{tikzcd}$$

Figure 8.1: Standardization Procedure

Structure of the chapter

We begin by formalizing the notion of descendant and residual in $SERS_{db}$. Residuals allow redexes to be traced along derivations. Thus if R and U are redexes in a term M and $M \overset{U}{\rightarrow} N$, then the residuals of R via the U-rewrite step in N may be defined with the aid of the notion of descendant. In general, R and U are required to be non-overlapping since otherwise there seems to be no general way of defining the notion of residual. However, given a *FExERS* $\mathcal{R}_{\mathcal{W}}$ it shall make sense for us to trace $\mathcal{R}$-redexes via $\mathcal{W}$-derivations, even though $\mathcal{R}$-redexes may overlap $\mathcal{W}$-redexes. Thus a different notion of tracing, which we call *correspondence*, shall be introduced.

After defining what it means for a derivation to be standard we recall the definition of the projection of a $fo(\mathcal{R})_{\mathcal{W}}$-derivation ϕ into its higher-order derivation $\mathcal{W}(\phi)$.

The notions of uncontributable symbol (intuitively, those that verify the property that any rewrite step below them cannot create redexes above them) and correspondent allow P-A.Melliès' proof technique to be applied to the case of arbitrary $fo(\mathcal{R})_{\mathcal{W}}$-derivations, obtaining Proposition 8.2.

The final section of this chapter proves a strong standardization theorem. This is achieved by showing that if a first-order derivation ϕ standardizes to another first-order derivation ψ, with ϕ Lévy permutation equivalent to ψ, then their projections $\mathcal{W}(\phi)$ and $\mathcal{W}(\psi)$, are also Lévy permutation equivalent.

8.1 Preliminaries

This section presents a more detailed survey on standardization in rewriting, and then introduces some notation. The reader already familiar with standardization may safely skip to the subsection on notation.

8.1.1 A Brief Survey

In 1958 the first standardization theorem was proved by H.B.Curry and R.Feys [CF58] for the $\lambda\beta$-calculus: for every β-derivation from a term M to a term N there exists a *standard* β-derivation from M to N. Before defining standard β-derivations we illustrate by means of an example the concept of *descendant* of a symbol (which we shall use for defining standard derivations) via some β-derivation, a formal development of this notion may be found in Section 8.2.

Example 8.3 Let $M = \Delta((\lambda x.x)z)$. The descendants of the rightmost λ-symbol in M via the derivation $M = \Delta((\lambda x.x)z) \rightarrow_\beta ((\lambda^* x.x)z)((\lambda^* x.x)z) = N$ are all the λ-symbols marked with an asterisk in N. The leftmost λ-symbol in M has no descendants in N.

By marking the head λ-symbol of a β-redex we may trace β-redexes. A standard β-derivation is defined as follows (we present an equivalent definition due to J.W.Klop [Klo80]):

Definition 8.4 (Standard β-derivation) Let $v : M_0 \xrightarrow{r_0} M_1 \xrightarrow{r_1} \dots$ be a finite or infinite β-derivation. We decorate v with markers '*' as follows: suppose up to M_{n-1} the markers have already been attached and consider the step $M_n \xrightarrow{r_n} M_{n+1}$. Mark

1. every λ in M_n which descends from a λ^* in M_{n-1}, and

2. every λ of a β-redex in M_n whose head λ is to the left of that of r_n, if not yet marked.

We say v is *standard* if no marked redex is contracted.

Note that a redex that is marked may be considered frozen in the sense that it shall never be reduced. Moreover, by observing that contracting a non-marked redex in a term M does not erase the marked redexes in M, we may conclude that not only are marked redexes never reduced but also they are never erased. Let us consider examples of standard and non-standard derivations.

Example 8.5 Let $I = \lambda x.x$. Then the derivation: $\Delta(Iz) \rightarrow \Delta z \rightarrow zz$ is not standard: indeed by applying Def.8.4 we obtain the following derivation decorated with markers: $(\lambda^* x.xx)(Iz) \rightarrow (\lambda^* x.xx)z \rightarrow zz$; note that the last redex contacted is a marked redex since its λ is marked with an asterisk. The derivation $\Delta(Iz) \rightarrow (Iz)(Iz) \rightarrow (Iz)z \rightarrow zz$ is not standard either. However, the β-derivation: $\Delta(Iz) \rightarrow (Iz)(Iz) \rightarrow z(Iz) \rightarrow zz$ is standard.

We may observe that in a standard derivation computation takes place from left to right. As a consequence normalization of the leftmost rewrite strategy in the $\lambda\beta$-calculus is obtained. Indeed, a standard derivation from a term M to a term N in β-normal form must be a leftmost derivation for otherwise there would be some marked redex which is present in N.

In 1978 J-J.Lévy [Lév78] strengthens this result and proves, with the aid of a notion of equivalence on β-derivations (*Lévy permutation equivalence*), that there exists a *unique* standard rewrite derivation in each equivalence class of β-derivations. This result is sometimes referred to as *strong standardization* in the literature.

In 1980 J.W.Klop [Klo80] provides two new proofs of strong standardization for $\lambda\beta$. The first proof consists of computing a standard derivation by repeatedly extracting the so called leftmost contracted redex. The second proof [Klo80, Section I.10] is based on rewriting β-derivations: a so called *anti-standard pair* is replaced by a *standard pair* in a β-derivation. By proving strong normalization and confluence of this notion of 2-dimensional rewriting he establishes strong standardization. He also shows that the equivalence of β-derivations induced by this notion of 2-dimensional rewriting coincides with Lévy permutation equivalence.

Perhaps the most important step in standardization was its extension to first-order rewriting. The task of freeing oneself from the left-to-right bias introduced by standardization in the context of the $\lambda\beta$-calculus was perhaps the challenging task, as remarked in [Mel01]. Let us restrict attention to the class of orthogonal first-order rewrite systems (*OTRS*). Consider the following adaptation of Def. 8.4 to first-order rewriting:

Definition 8.6 (K-standard derivation) Let $\mathcal{R}$ be an *OTRS* and $v : M_0 \xrightarrow{r_0} M_1 \xrightarrow{r_1} \dots$ be a finite or infinite $\mathcal{R}$-derivation. We decorate v with markers '*' as follows: suppose up to M_{n-1} the markers have already been attached and consider the step $M_n \xrightarrow{r_n} M_{n+1}$. Mark

1. every symbol s in M_n which descends from a s^* in M_{n-1}, and

2. every $\mathcal{R}$-redex in M_n whose head symbol is to the left of that of r_n, if not yet marked.

We say v is *standard* if no marked redex is contracted.

Let us consider some of the problems presented by *OTRS* as regards standardization.

Example 8.7 The leftmost-outermost strategy no longer normalizes. Indeed, consider the following *OTRS* $\mathcal{R} = \{f(X,a) \rightarrow b,\ c \rightarrow c,\ d \rightarrow a\}$ taken from [HL91]. There is a standard derivation from the term $f(c,d)$ to its normal form: $f(c,d) \rightarrow f(c,a) \rightarrow b$ yet the leftmost-outermost strategy leads to an infinite derivation: $f(c,d) \rightarrow f(c,d) \rightarrow \dots$.

Thus the leftmost-outermost strategy is no longer normalizing for *OTRS*. But things may get worse for the standardization theorem itself may fail.

Example 8.8 Consider the *OTRS* $\mathcal{S} = \{f(X, a) \to g(X, X),\ c \to d,\ i(X) \to X\}$ taken from [Klo80] and consider also the derivation $v : f(c, i(a)) \to f(c, a) \to g(c, c) \to g(d, c)$. Then recalling the definition of K-standard derivation (Def. 8.6) and inspecting the derivation graph of the term $f(c, i(a))$ in Figure 8.2 the reader may note that there is no standard derivation from $f(c, i(a))$ to $g(d, c)$.

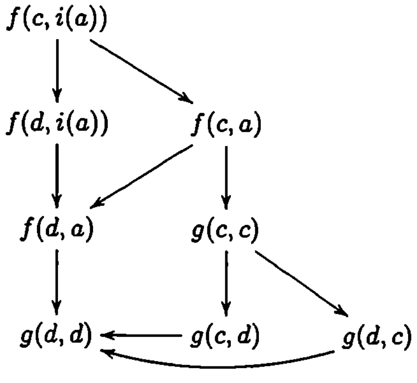


Figure 8.2: $\mathcal{S}$-graph of the term $f(c, i(a))$

In 1979 G.Huet and J-J.Lévy [HL79] (later published as [HL91]) proposed the following notion of standard derivation for *OTRS*:

Definition 8.9 (HL-standard derivation) Let $\mathcal{R}$ be an *OTRS* and let $v : M_0 \xrightarrow{r_0} M_1 \xrightarrow{r_1} \dots$ be a finite or infinite $\mathcal{R}$-derivation. We say v is *standard* if for every rewrite step r_n with $M_n = C[\rho L]_p$ and $M_{n+1} = C[\rho R]_p$:

1. either all r_j with $j > n$ are not above[2] p,

2. or, if $M_m = C'[\rho' L']_q$ and $M_{m+1} = C'[\rho' R']_q$ is the first rewrite step with $m > n$ in v above p then $p = q.p'$ with p' a non-variable position of L'.

Thus in a HL-standard derivation computation proceeds in an outside-in fashion. The only situation where a redex r_j occurring at a position q may be contracted *after and above* a redex r_i (with $i < j$) at position p with q above p, is when r_i 'contributes' to the redex r_j. The derivation $f(c, d) \to f(c, a) \to b$ of Example 8.7 is a HL-standard derivation. The derivation v of Example 8.8 is also a HL-standard derivation.

It is also shown in [HL91] that if $\mathcal{R}$ is an orthogonal term rewrite system and $v : M \twoheadrightarrow_{\mathcal{R}} N$ then there is a unique (up to permutation of disjoint redexes) HL-standard derivation $\phi : M \twoheadrightarrow_{\mathcal{R}} N$ with ϕ Lévy permutation equivalent to v.

In 1985 G.Boudol [Bou85] extended the work of G.Huet and J-J.Lévy to the case of left-linear ambiguous term rewrite systems. This extension to term rewrite systems allowing the presence of critical pairs is a non-trivial extension. First-order conditional rewriting has been dealt with by T.Suzuki [Suz96].

In 1992 G.Gonthier, J-J.Lévy and P-A.Melliès [GLM92] published an axiomatic standardization theorem. Abstract Reduction Systems [Klo92] are equipped with a primitive residual relation for tracing redexes along derivations and also a primitive nesting relation between redexes yielding *Axiomatic Reduction Systems*. For those Axiomatic Reduction Systems satisfying some axioms an abstract proof of standardization was obtained. Thus by an appropriate instantiation of the axioms they obtain a proof of standardization covering, among others, the $\lambda\beta$-calculus, *OTRS*, orthogonal *CRS*, and some graph based systems (e.g. dags, interaction networks). The proof is based on ideas taken from J.W.Klop's first proof of standardization for $\lambda\beta$ [Klo80].

In his 1996 PhD thesis P-A.Melliès [Mel96] continues the study of axiomatic rewrite systems, and in particular, axiomatic standardization. Based on a modified axiomatics for his Axiomatic Reduction Systems he presents a new proof of strong standardization. Although the construction of a standard derivation proceeds by extracting external redexes as in [GLM92], this process of extraction is computed by a notion of 2-dimensional rewriting, as in J.W.Klop's second proof (although, in contrast to J.W.Klop, a notion of 2-dimensional rewriting *modulo* certain permutations is put to work). Finally, the full power of the 2-dimensional approach to standardization is explored in [Mel01]: *Axiomatic Rewrite Systems* (*AxRS*) are introduced as a pair consisting of a graph

[2] Def. 8.10.

and a binary relation between the paths in this graph. The axioms on *AxRS* impose conditions on this relation between paths and no longer speak in terms of residual relations, nesting relations and compatibility [GLM92].

As regards graph-rewriting, in 1995 D.Clark and R.Kennaway [CK96] extended the work of G.Huet, J-J.Lévy and G.Boudol to graph rewriting systems.

In the realm of higher-order rewriting in his PhD thesis [Klo80] J.W.Klop proves strong standardization for the class of ***left-normal*** orthogonal (called regular in [Klo80]) *CRS*. The notion of standard derivation he uses is K(lop)-standard derivation of Def. 8.6. The problems mentioned in examples 8.7 and 8.8 are avoided by restricting attention to the subclass of left-normal orthogonal *CRS*: a *CRS* $\mathcal{R}$ is left-normal if all function symbols occur to the left of all metavariables in every *LHS* of a rule in $\mathcal{R}$. Examples of left-normal orthogonal *CRS* are $\lambda\beta$ and Combinatory Logic.

As already mentioned [GLM92, Mel96] also cover higher-order rewriting. Moreover, the abstract standardization theorem in [Mel96] applies to left-linear *CRS* admitting critical pairs (see also [WM00]). In [Oos96] van Oostrom provides a proof of standardization based on 2-dimensional rewriting for orthogonal *HRS* thus extending[3] the work of J.W.Klop. The results by G.Gonthier et al, P-A.Melliès and van Oostrom do not insist on left-normality hence the standard derivations obtained are unique *up to* disjoint permutation of redexes, in contrast to the uniqueness of standard derivations obtained by J.W.Klop for left-linear orthogonal *CRS*. However, as a bonus J.W.Klop obtains normalization of the leftmost rewrite strategy for left-normal orthogonal *CRS* [Klo80, Remark 6.2.8.10].

8.1.2 Notation

We shall use letters $r, u, v, \ldots$ for *FExERS*-redexes and $R, U, V, \ldots$ for $SERS_{db}$-redexes; $v, \phi, \psi, \omega, \ldots$ for *FExERS*-derivations and $\Upsilon, \Phi, \Psi, \Omega, \ldots$ for $SERS_{db}$-derivations; and $\mathcal{R}, \mathcal{S}$ for $SERS_{db}$. We say that a derivation of the form $M_1 \xrightarrow{u_1} M_2 \xrightarrow{u_2} M_3 \ldots M_n \xrightarrow{u_n} M_{n+1}$ has length n; this derivation is also written $u_1; \ldots; u_n$. Every term M induces an empty derivation e_M which we abbreviate e that satisfies $e; \phi = \phi; e = \phi$ (where ϕ is a *FExERS* or $SERS_{db}$-derivation).

Recall that $fo(\mathcal{R})_{\mathcal{W}}$ denotes the *FExERS* which is the first-order version of the $SERS_{db}$ $\mathcal{R}$ (Def. 7.35) and where the substitution calculus has been fixed to be the $\mathcal{W}$-calculus, for some basic substitution calculus $\mathcal{W}$. The rewrite rules in $fo(\mathcal{R})_{\mathcal{W}}$ are $fo(\mathcal{R}) \cup \mathcal{W}$ (Def. 7.15). In the sequel we shall fix σ as basic substitution calculus unless stated otherwise.

We recall that a $SERS_{db}$-metaterm M is ***linear*** if it contains at most one occurrence of an X-based metavariable (Def. 7.1). We say $\mathcal{R}$ is ***left-linear*** if the *LHS* of each rewrite rule in $\mathcal{R}$ is linear.

8.2 Descendants and Residuals

This section introduces the notions of *descendants* and *residuals* with the primary objective of fixing notation. Descendants shall allow us to *trace* subterms and to define *residuals*. In full precision, we shall study the descendants of positions along rewrite derivations. We then study some specific properties of descendants in the substitution calculus σ.

8.2.1 Definitions

A position is a sequence of natural numbers (i.e. elements of $\mathbb{N}^*$); ϵ denotes the empty sequence. We use letters $p, q, \ldots$ for positions. The set of positions of a $SERS_{db}$-pre-metaterm (Def. 6.27) M is denoted $Pos(M)$, and the subterm of a pre-metaterm M at a position $p \in Pos(M)$ is denoted $M|_p$; both notions are defined simultaneously as follows:

- if M is a de Bruijn index or a metavariable then: $Pos(M) \stackrel{\text{def}}{=} \{\epsilon\}$ and $M|_\epsilon \stackrel{\text{def}}{=} M$.
- if $M = f(M_1, \ldots, M_n)$ for f a function symbol, or $M = \xi(M_1, \ldots, M_n)$ for ξ a binder symbol then: $Pos(M) \stackrel{\text{def}}{=} \{\epsilon\} \cup \{i.p \mid p \in Pos(M_i),\ 1 \leq i \leq n\}$ and $M|_\epsilon \stackrel{\text{def}}{=} M$. Also, $M|_{i.p'} \stackrel{\text{def}}{=} M_i|_{p'}$ where $1 \leq i \leq n$.
- if $M = M_1[\![M_2]\!]$ then: $Pos(M) \stackrel{\text{def}}{=} \{\epsilon\} \cup \{1.p \mid p \in Pos(M_1)\} \cup \{2.p \mid p \in Pos(M_2)\}$ and $M|_\epsilon \stackrel{\text{def}}{=} M$. Also, $M|_{i.p'} \stackrel{\text{def}}{=} M_i|_{p'}$ where $1 \leq i \leq 2$.

[3] And also correcting (see [Mel96, Section 6.2.2]).

Note that this definition covers de Bruijn terms (since they are de Bruijn pre-metaterms) and *FExERS* terms (since they are first-order terms, in particular the substitution operator $\bullet[\bullet]$ is regarded as a binary function symbol $[](\bullet,\bullet)$). Also, this definition of positions is different (in the case of the de Bruijn metasubstitution operator) from the one we dealt with in Chapter 6 (just after Def. 6.27), where positions were defined on the basis of trees which were associated to de Bruijn pre-metaterms. The latter definition introduced a dummy function symbol *sub* when associating a tree to a pre-metaterm containing a de Bruijn metasubstitution operator (Figure 6.2), and proved appropriate for studying parameter paths. In this chapter we revert to the classical definition of position where the de Bruijn metasubstitution operator is considered as a binary function symbol (see [KOO01a]).

Definition 8.10 (On positions and symbol positions) We write $\leq$ for the *prefix ordering* on positions ($p \leq q$ iff $q = p.p'$) and $<$ for the strict prefix ordering on positions. When $p \leq q$ we say p is 'above' or is a 'prefix of' q, and when $p < q$ we say p is 'strictly above' or is a 'strict prefix of' q. If $p \not\leq q$ and $q \not\leq p$ then p and q are said to be *disjoint*, written $p \parallel q$. A position p in M is called a *symbol position* if $M|_p = f(M_1,\ldots,M_n)$ for some function symbol f, or $M|_p = \xi(M_1,\ldots,M_n)$ for some binder symbol ξ. In that case we write $p \in SPos(M)$. Furthermore, $SPos(M,f)$ stands for the set of positions of the symbol f in the term M (be it a binder or a function symbol). The *position of a redex* in a term is the position of its head symbol in that term. A position is above (resp. strictly above) a redex if it is above (resp. strictly above) the position of its head symbol.

Definition 8.11 (Preservation of a position) A derivation $u_1;\ldots;u_n$ *preserves* a position p when none of the redexes u_i is above p.

If $p = p'.i$ is a position in M where i is some natural number then we define $\mathbf{prev}(p,M) \stackrel{\text{def}}{=} p'$.

Definition 8.12 (Patterns, arguments, and nested redexes) Subterms of a redex $M = \rho(L)$ that correspond to metavariables in L are the *arguments* of M, and the *pattern* of M is the pattern of L (Def. 6.31). If V and U are redexes in a term M then we say V *nests* U in M if U occurs in an argument of V in M.

Note that in order to determine if V nests U not only do we need to know that V is a redex but we also have to know what rewrite rule it is an instance of. For example if $\mathcal{R} = \{a \rightarrow_a b,\ f(X_\epsilon) \rightarrow_{f1} b,\ f(a) \rightarrow_{f2} b\}$ then both a and $f(a)$ are redexes, however to determine if $f(a)$ nests a we need to know what redex $f(a)$ is an instace of.

Example 8.13 Consider the term $M = app(\lambda(\lambda(app(\lambda 1, 1))), 2)$. Note that M is a β-redex. The subterms $\lambda(app(\lambda 1, 1))$ and 2 are arguments of M, the remaining symbols in M conform the pattern of M. If we replace the arguments of M with some distinguished constant $\Box$ then the pattern of M may be written $app(\lambda(\Box), \Box)$. The β-redex in $\lambda(app(\lambda 1, 1))$ is nested by M.

In the literature on higher-order rewriting it is not uncommon to decompose each rewrite step into two parts (we shall follow [KOO01a]): a *TRS* part in which the *LHS* is replaced by the *RHS* without evaluating the metasubstitution operators, and a substitution part where the delayed metasubstitution operators are evaluated. The reason for introducing this refinement of a higher-order rewrite step (initially due to J.W.Klop [Klo80]) is that the *RHS*s of rewrite rules may contain nested metasubstitution operators rendering the task of tracing terms via the rewrite step non-trivial. The descendant relation of a reduction step can be obtained by composing the descendant relation on the *TRS* part and the descendant relation of the metasubstitution evaluation part.

Definition 8.14 Let $(\Sigma,\mathcal{R})$ be a $SERS_{db}$ and let Σ' be the $SERS_{db}$-signature resulting from augmenting Σ with a fresh function symbol f and binder symbol ξ. The *substitution-frozen variant* of $(\Sigma,\mathcal{R})$, written $(\Sigma', SFV(\mathcal{R}))$ (or simply $SFV(\mathcal{R})$ is no confusion may arise), is obtained by replacing every occurrence of the metasubstitution operator $M[N]$ on the *RHS* of a rewrite rule with the metaterm $f(\xi(M), N)$. The *substitution-evaluation variant* of $(\Sigma,\mathcal{R})$ is defined as the $SERS_{db}$ consisting of the alphabet Σ' and the sole rewrite rule:

$$f(\xi(X_\alpha), Y_\epsilon) \rightarrow X_\alpha[Y_\epsilon]$$

We write $(\Sigma', SEV(\mathcal{R}))$ (or, simply $SEV(\mathcal{R})$) for the substitution-evaluation variant of $(\Sigma,\mathcal{R})$.

A $SERS_{db}$ whose *RHS*s do not contain occurrences of the metasubstitution operator $\bullet[\bullet]$ is called *simple*. The substitution-frozen variant of a $SERS_{db}$ is always simple. We shall define the *descendant relation* on a simple $SERS_{db}$ and on the substitution-evaluation variant of $SERS_{db}$ separately.

Definition 8.15 (Descendants of positions in a *TRS* step) Let $\mathcal{R}$ be a simple $SERS_{db}$ and let U be an (L, R)-redex for $(L, R) \in \mathcal{R}$ occurring at a position p in a term M, let $M \xrightarrow{U} N$, and let q be a position in M.

1. If p and q are disjoint, then the descendant of q is the same[4] position q in N.
2. If $q \leq p$, then again the descendant of q is the position q in N.
3. If $q = p.p'$ where p' is a nonvariable position of L, then the descendant of q is the position p in N (i.e. the position of the contractum of U) [5].
4. If $q = p.p_i.p'$ where p_i is the position of the ith-from-the-left metavariable occurrence in L, then the descendants (possibly none at all) of q are all the positions in N of the form $q_j = p.p_i^j.p'$ for $1 \leq j \leq n_i$ where $p_i^1, \ldots, p_i^{n_i}$ are the positions of all occurrences of that same metavariable in R.

We write $p[\![U]\!]q$ if the the position q in N is a descendant of the position p in M.

Subterms may be traced by their positions. If P is a subterm of M occurring at a position p and $p[\![U]\!]q$, then the subterm of N at position q is the descendant of P via U. Note that every subterm except erased ones have one or more descendants.

Example 8.16 Consider the step $M = app(c, \lambda(app(\lambda P, Q)[id])) \rightarrow_{Func_{app}} app(c, \lambda(app((\lambda P)[id], Q[id]))) = N$. Here are some examples of descendants of this *TRS* step. The term N is a descendant of the term M by case 2 of Def. 8.15. The term $app((\lambda P)[id], Q[id])$ is a descendant of $app(\lambda P, Q)[id]$ and $app(\lambda P, Q)$ by case 3 of Def. 8.15; the subterm λP of N is a descendant of the same subterm in M by case 4 of Def. 8.15; both occurrences of the subterm id in N are descendants of id in M (i.e. it has two descendants). The subterm c in N is a descendant of c in M by case 1 of Def. 8.15.

If S is a set of positions we use the notation $S[\![U]\!]$ for $\{q \mid p[\![U]\!]q \text{ for some } p \in S\}$. Let us consider now how to trace terms via a substitution step.

Definition 8.17 (Descendants of positions in a substitution step) Let $U = f(\xi(O_1), O_2)$ be a $SEV(\mathcal{R})$-redex in M at a position p, let $M \xrightarrow{U} N$, and let q be a position q in M.

1. If p and q are disjoint, then the descendant of q is the same position q in N.
2. If $q \leq p$, then again the descendant of q is the position q in N.
3. If $q = p.1.1.p'$ (i.e. $M|_q \subseteq O_1$), then the descendant of q is the position $p.p'$ in N.
4. If $q = p.2.p'$ (i.e. $M|_q \subseteq O_2$), then the descendants (possibly none at all) of q are the positions in N of the form $q_i = p.p_i.p'$ with $1 \leq i \leq n$ where $p_1, \ldots, p_n$ are the positions of all occurrences of the constant symbol $*$ in $O_1\{\!\{1 \leftarrow *\}\!\}$[6].

The descendants of arbitrary derivations of a $SERS_{db}$ may be obtained by decomposing each step into a *TRS* step and (zero or more) substitution steps, applying the descendant concept as defined above, and taking the transitive closure and noting that relation composition is associative. If Υ is an $\mathcal{R}$-derivation then Υ-descendants are defined to be the descendants under the decomposition of Υ. Also, we shall speak of the Υ-descendants of a *position* p for $p \in SPos(M)$ meaning the Υ-descendants of the subterm at position p in M. The *ancestor* relation is the inverse of the descendant relation.

Residuals shall be defined for redexes only whereas descendants are defined for all subterms. Contracted redexes shall not have residuals (however, they do have descendants).

Definition 8.18 ($SERS_{db}$-residual) Let $M \xrightarrow{U} N$ in a $SERS_{db}$ $\mathcal{R}$ with U an (L, R)-redex, let $V \subseteq M$ be an (L', R')-redex, and let $P \subseteq N$ be a U-descendant of V. We call P a *U-residual of* V, written $V[\![U]\!]P$, if

[4] There is a slight abuse of notation here since positions are relative to terms.

[5] This definition coincides with that of G.Boudol [Bou85] where the term *trace* is used instead of descendant and that of Z.Khasidashvili [Kha93] where a related notion called *essentiallity* is studied.

[6] Since $*$ is a constant note that for the updating functions we have $\mathcal{U}_i^n(*) = *$. Also, recall that $\bullet\{\!\{\bullet \leftarrow \bullet\}\!\}$ is the de Bruijn substitution on terms (Def. 6.34)

1. the patterns of U and V do not overlap, and
2. P is an (L', R')-redex.

A redex in N is said to be *created* if it is not the residual of a redex in M. When considering the class of left-linear $SERS_{db}$ the second condition in Def. 8.18 may be dropped, as in the first-order case. Observe that since first-order rewrite systems are a strict subclass of $SERS_{db}$ the above definition applies to them too.

Example 8.19 Consider the following *OTRS* from Example 8.7 $\mathcal{R} = \{f(X,a) \rightarrow_f b,\ c \rightarrow_c c,\ d \rightarrow_d a\}$ and the rewrite step $M = f(c,d) \rightarrow_d f(c,a) = N$. The redex $f(c,a)$ in N is created since it is not a residual of a redex in M. However, the occurrence of c in N is a residual of the c-redex in M.

We would now like to introduce the *correspondence* relation, a notion belonging to the first-order rewrite systems *FExERS*. In a *FExERS* $\mathcal{R}_\mathcal{W}$ we shall sometimes be interested in tracing $\mathcal{R}$-redexes through $\mathcal{W}$-reductions, for $\mathcal{W}$ some basic substitution calculus. Although first-order rewrite systems are particular $SERS_{db}$s, Def. 8.18 is too general for our purposes. The problem is that $\mathcal{R}$-redexes may overlap $\mathcal{W}$-redexes since $\mathcal{W}$ is in charge of propagating the substitution operator. This entails that an $\mathcal{R}$-redex may be 'lost' when traversed by a substitution operator.

Example 8.20 Consider the basic substitution calculus σ. The *Beta*-redex in M is lost in the following σ-reduction-step:

$$M = app(\lambda P, Q)[id] \rightarrow_{Func_{app}} app((\lambda P)[id], Q[id])$$

A notion of residual for the particular case of the $\lambda\sigma$-calculus relating to that of the λ-calculus has been studied in [Ber92]. In order to regain lost $\mathcal{R}$-redexes through $\mathcal{W}$-derivations we shall introduce the correspondence relation. Owing to the fact that a basic substitution calculus is a first-order rewrite system (Def. 7.10) this notion makes use of the descendant concept for the *TRS* part of higher-order rewrite steps (Def. 8.15).

Definition 8.21 (*FExERS*-**correspondence**) Let $\mathcal{R}_\mathcal{W}$ be a *FExERS*. Let r be an $(L,R) \in \mathcal{R}$-redex occurring in M. Let $v : M \twoheadrightarrow_\mathcal{W} N$ be a $\mathcal{W}$-derivation. Suppose

1. $r \subseteq M$ v-descends to a subterm $u \subseteq N$ at position q, i.e. $p[\![v]\!]q$ where p is the position of the head symbol of r, and
2. $u = N|_q$ is an (L,R)-redex.

then u is a *v-correspondent* of r in N, written $r\langle\!\langle v\rangle\!\rangle u$.

Example 8.22 Assume P and Q are pure terms (they contain no occurrences of the substitution operator) and consider the derivation: $M = app(\lambda P, Q)[id] \rightarrow_{Func_{app}} app((\lambda P)[id], Q[id]) \twoheadrightarrow_\sigma app(\lambda(P), Q) = N$ Then the *Beta*-redex at position 1 in M has the *Beta* (or β)-redex at position ϵ in N as correspondent.

As remarked before there seems to be no general way of defining the residual of a redex under the contraction of some other *overlapping* redex. However, for the particular case of substitution calculi (in particular we shall use it in the case of σ only; see Remark 8.46) Def. 8.21 shall accomplish its duty, and it is only in the context of such calculi that the latter definition must be considered. However, this Def. 8.21 seems to make sense for any basic calculi of substitutions satisfying 'similar properties as those of σ which we shall study in the next subsection, of which three important ones are: σ may not create function or binder symbols, the ancestor relation induced by σ rewriting over the function and binder symbols is a total function, and any two σ-derivations to σ-normal form induce the same descendant relation on positions.

8.2.2 Tracing Terms in the Substitution Calculus

We now study some basic properties concerning σ-residuals and σ-correspondents of $\mathcal{R}$-redexes in a *FExERS* $\mathcal{R}_\sigma$. This entails the study of descendants of positions of function and binder symbols via σ-derivations. Figure 8.3 recalls the rewrite rules of the basic substitution calculus σ over some signature Γ. We begin with a remark.

Remark 8.23 If $p \in SPos(M,f)$ for some function or binder symbol f, $v : M \twoheadrightarrow_\sigma N$ and $p[\![v]\!]q$ then $q \in SPos(N,f)$. That is to say, σ-descendants of function or binder symbols are once again function or binder symbols, and moreover, they have the same 'name'. This may be verified by inspecting the rewrite rules of the σ-calculus.

$$
\begin{array}{lll}
f(P_1,\dots,P_n)[s] & \rightarrow_{Func_f} & f(P_1[s],\dots,P_n[s]) \\
\xi(P_1,\dots,P_n)[s] & \rightarrow_{Bind_\xi} & \xi(P_1[1\cdot(s\circ\uparrow)],\dots,P_n[1\cdot(s\circ\uparrow)]) \\
P[s][t] & \rightarrow_{Clos} & P[s\circ t] \\
1[P\cdot s] & \rightarrow_{VarCons} & P \\
1[id] & \rightarrow_{VarId} & 1 \\
 & & \\
(P\cdot s)\circ t & \rightarrow_{Map} & P[t]\cdot(s\circ t) \\
id\circ s & \rightarrow_{IdL} & s \\
(s_1\circ s_2)\circ s_3 & \rightarrow_{Ass} & s_1\circ(s_2\circ s_3) \\
\uparrow\circ(P\cdot s) & \rightarrow_{ShiftCons} & s \\
\uparrow\circ id & \rightarrow_{ShiftId} & \uparrow
\end{array}
$$

Figure 8.3: The σ-calculus

The next property we look at is *parametricity* [Oos94] of σ over function and binder symbols. The intuition is that the names of function and binder symbols which are traced is of no importance in a σ-derivation to normal form. As mentioned in [Oos94], an example of a parametric calculus is the λ-calculus: it is parametric over 'symbols' [Klo80, I.10.1.2] (any two $\lambda\beta$-derivations from M to a β-normal form N are Lévy permutation equivalent). Let us write $M \rightarrowtail_\sigma N$ when M σ-rewrites to σ-normal form N.

Lemma 8.24 (Parametricity of σ) All derivations $\upsilon : M \rightarrowtail_\sigma N$ (i.e. N in σ-normal form) induce the same descendant relation $\llbracket\upsilon\rrbracket$ over $SPos(M) \times SPos(N)$.

Proof. We use a proof technique due to van Oostrom [Oos94]. We must prove that if $\upsilon, \phi : M \rightarrowtail_\sigma N$ then $\llbracket\upsilon\rrbracket = \llbracket\phi\rrbracket$. Before proceeding two observations:

Observation 1. The σ-calculus does not create function or binder symbols, i.e. if $S \subseteq SPos(M)$ is the set of all positions in M of some (binder or function) symbol f and $M \xrightarrow{r}_\sigma N'$ then $S\llbracket r\rrbracket$ is the set of all symbol positions of f in N' .

Observation 2. If we replace some (function or binder) symbol f in M occurring at a position p with a fresh symbol g obtaining M', then the derivation υ is transformed into a new σ-derivation υ', and

$$p\llbracket\upsilon\rrbracket q \Longleftrightarrow p\llbracket\upsilon'\rrbracket q$$

This may be verified by induction on the length of υ.

Now let $p \in SPos(M, f)$ (i.e. p is a position of the symbol f in the term M) and let g be a fresh symbol. Then replacing f with g in M yields a term M' and two new σ-derivations υ' and ϕ' from M' to N_1 and N_2, respectively, such that:

$$p\llbracket\upsilon\rrbracket q \Longleftrightarrow p\llbracket\upsilon'\rrbracket q \text{ and } p\llbracket\phi\rrbracket q \Longleftrightarrow p\llbracket\phi'\rrbracket q \tag{8.1}$$

Since g is a fresh symbol then:

$$p\llbracket\upsilon'\rrbracket q \Longleftrightarrow \text{ the head symbol of } N_1|_q \text{ is } g \tag{8.2}$$

Indeed, the left-to-right direction follows from Remark 8.23. For the right-to-left direction suppose the head symbol of $N_1|_q$ is g and that $p \in SPos(M, g)$ then by the first observation it follows that p must be an ancestor of q, that is, $p\llbracket\upsilon'\rrbracket q$.

So, completing the equivalences 8.1 with the equivalence 8.2 we obtain:

$$p\llbracket\upsilon\rrbracket q \Longleftrightarrow p\llbracket\upsilon'\rrbracket q \Longleftrightarrow \text{ the head symbol of } N_1|_q \text{ is } g$$

and

$$p\llbracket\phi\rrbracket q \Longleftrightarrow p\llbracket\phi'\rrbracket q \Longleftrightarrow \text{ the head symbol of } N_2|_q \text{ is } g$$

Finally, the result follows from noting that $N_1 = N_2$ from the confluence of the σ-calculus.

It is not clear that Lemma 8.24 holds for arbitrary σ-derivations due to the possibility of "syntactic coincidences" [HL91]. The following example illustrates this phenomenon when arbitrary symbols are traced over arbitrary σ-derivations.

Example 8.25 Let $P = 1[id][id]$ and $Q = 1[id]$. Then $v : \underline{1[id][id]} \rightarrow_{Clos} 1[\underline{id \circ id}] \rightarrow_{IdL} 1[id]$ and $\phi : \underline{1[id]}[id] \rightarrow_{VarId} 1[id]$. If we trace the subterm $1[id]$ in P occurring at position 1 we obtain $1[\![v]\!]\epsilon$ and $1[\![\phi]\!]1$.

Given a *FExERS* $\mathcal{R}_\sigma$ we shall be tracing $\mathcal{R}$-redexes. Thus the following result shall be useful. Its proof relies on Lemma 8.24.

Corollary 8.26 (Parametricity of σ over $\mathcal{R}$-redexes) Let $\mathcal{R}_\sigma$ be a *FExERS*. All derivations $v : M \rightarrowtail_\sigma N$ (i.e. N in σ-normal form) induce the same correspondence relation $\langle\!\langle v \rangle\!\rangle$ over $\mathcal{R}$-redexes.

Corollary 8.26 allows us to speak of *the* correspondents of some $\mathcal{R}$-redex in M, in $\sigma(M)$. A related notion of equivalence of derivations is strong equivalence as defined by Hindley in [Hin78] (see also [Klo80]): two derivations $v : M \twoheadrightarrow N$ and $\phi : M \twoheadrightarrow N$ are *Hindley-equivalent* if for every redex $r \in M$ the residuals of r via v coincide with those of r via ϕ.

The following notion of lined-up symbol positions shall be used when dealing with strong standardization. It was introduced by the author in an early proof of the Projection Proposition, and independently in [Mel96] as part of a "dynamical order" on $\lambda\sigma$-terms (see Section 8.3.2 for further details).

Definition 8.27 (Lined-up symbol positions) Let $p, q \in SPos(M)$. We say p is *lined-up* with q in M iff $p = o.1.p'$ and $q = o.2.q'$ and one of the following two conditions hold:

- either, the head-symbol of $M|_o$ is $\bullet[\bullet]$,
- or, the head-symbol of $M|_o$ is $\bullet \circ \bullet$.

A set S of disjoint symbol positions in a term M is lined-up if for every $p, q \in S$ either p is lined-up with q in M or q is lined-up with p in M. Also, if r and u are redexes in M, we say r is lined-up with u in M iff the positions of their head symbols are lined-up.

As an example of lined-up symbol positions consider the term $M = f(c)[g(1) \cdot ((f(2) \cdot id) \circ (g(2) \cdot id))]$. Then the position 1 is lined-up with position 2.1 (and with 2.2.1.1, and with 2.2.2.1), however 2.1 is not lined-up with 1. Also 2.1 is not lined-up with 2.2.1.1. The position 2.2.1.1 is lined-up with 2.2.2.1.

Note that the relation 'is lined-up with' is not symmetrical since if p is lined-up with q in M then p is to the left of q in M. The intuition behind Def. 8.27 is that lined-up $\mathcal{R}$-redexes may potentially yield nested residuals via σ-reduction.

Definition 8.28 (Conflict-free set of symbol positions) A set S of symbol positions in M is called *a conflict-free set of positions in M* if S is a set of disjoint non-lined-up symbol positions in M.

Lemma 8.29 (Descendants of a conflict-free set) Let M be a term and S be a conflict-free set of symbol positions in M. Suppose $M \xrightarrow{u}_\sigma N$. Let S' be the set of u-descendants of positions in S in N. Then S' is a conflict-free set in N, i.e. all u-descendants of positions in S are once again disjoint and, moreover, they are non-lined-up.

Proof. Let S be a conflict-free set in M and suppose $M \xrightarrow{u}_\sigma N$. The proof is by induction on the (length of the) position where the rewrite step takes place. We shall consider only the cases where the rewrite takes place at the root, for the other cases follow by applying the induction hypothesis.

1. $f(P_1, \ldots, P_n)[s] \rightarrow f(P_1[s], \ldots, P_n[s])$. If $S = \{p\}$ then the u-descendants of p are a conflict free-set in N (note that if $p \in s$ then p shall have n descendants). Otherwise, consider any two positions $p, q \in S$ with $p \neq q$. Then

   - either, there are indices i, j with $1 \leq i, j \leq n$ such that $p \in P_i$ and $q \in P_j$,
   - or, $p, q \in s$.

In both cases their u-descendants are disjoint and non-lined up. Note that it is not possible that $p \in P_i$ for some $1 \leq i \leq n$ and $q \in s$ since S is non-lined-up.

2. $\xi(P_1,\ldots,P_n)[s] \to \xi(P_1[1 \cdot (s \circ \uparrow)],\ldots,P_n[1 \cdot (s \circ \uparrow)])$. As in the previous case.

3. $P[s][t] \to P[s \circ t]$. If $S = \{p\}$ then either $p \in P$, or $p \in s$ or $p \in t$. These cases are seen to hold. Otherwise, consider any two positions $p, q \in S$ with $p \neq q$. Then it must be that $p, q \in P$, or $p, q \in s$ or $p, q \in t$, in all cases S' is seen to be a conflict-free set in N.

4. $1[id] \to 1$ or $\uparrow \circ id \to \uparrow$. These cases hold trivially since $S = \emptyset$ and there are no u-descendants to consider.

5. $1[P \cdot s] \to P$. If $S = \{p\}$ then either $p \in P$, or $p \in s$ and the result follows since p has at most one u-descendant. Otherwise, consider any two positions $p, q \in S$ with $p \neq q$. Then either $p, q \in P$ or $p, q \in s$ or $p \in P$ and $q \in s$, and the result holds as above.

6. $id \circ s \to s$, $\uparrow \circ (P \cdot s) \to s$, $(s1 \circ s2) \circ s3 \to s1 \circ (s2 \circ s3)$, and $(P \cdot t) \circ s \to P[s] \cdot (t \circ s)$ are analogous to the previous case.

We recall the reader that $M \rightarrowtail_\sigma N$ indicates that M σ-rewrites to the σ-normal form N.

Corollary 8.30 (Correspondents of a conflict-free set) Let $\mathcal{R}_\sigma$ be a *FExERS*. Consider a term M and a conflict-free set S of $\mathcal{R}$-redex positions in M. Suppose $v : M \rightarrowtail_\sigma N$. Let S' be the set of v-correspondents of redexes in S in N. Then S' is a conflict-free set in N, i.e. all v-correspondents of redexes in S are once again disjoint and, moreover, they are non-lined-up.

We have seen how descendants of non-lined up positions behave, now we shall look at how descendants of lined-up positions behave. This is not required for the standardization theorem (Proposition 8.2 anounced in the introduction of the chapter), it shall be required when considering strong standardization (Proposition 8.58). We seek to prove the following two propositions:

Proposition 8.31 (Correspondents of lined-up redexes) Let $\mathcal{R}_\sigma$ be a *FExERS*. Let r, u be $\mathcal{R}$-redexes in M such that r is lined-up with u in M. Let $v : M \rightarrowtail_\sigma N$ and let r' be a v-correspondent of r and u' a v-correspondent of u in N. Then one of the following holds:

1. either, r' nests u',

2. or, r' is disjoint with u'.

Proposition 8.32 (Correspondents of nested redexes) Let $\mathcal{R}_\sigma$ be a *FExERS*. Let r, u be $\mathcal{R}$-redexes in M such that r nests u. Let $v : M \rightarrowtail_\sigma N$ and let r' be a v-correspondent of r and u' a v-correspondent of u in N. Then one of the following holds:

1. either, r' nests u',

2. or, r' is disjoint with u'.

In order to prove Proposition 8.31 and 8.32 we need to prove more general statements. This we set out to do below.

Lemma 8.33 (One-step descendants of symbol positions which are one above the other) Let $p, q \in SPos(M)$ with $p < q$. Let $M \xrightarrow{u}_\sigma N$ and let p' be a u-descendant of p and q' a u-descendant of q in N. Then either $p' < q'$ or $\{p', q'\}$ is a conflict-free set in N.

Proof. By a close inspection of the rewrite rules in the σ-calculus. We proceed by induction on the (length of the) position where the rewrite step takes place. Also, we shall assume M is either a term of sort **T** or **S**.

- rewrite-step at the root position.

1. $f(M_1, \ldots, M_n)[s] \rightarrow f(M_1[s], \ldots, M_n[s])$. If $p, q \in M_i$ for some $1 \leq i \leq n$ then p', q' are unique and $p' < q'$. If $p = 1$ and $q \in M_i$ with $1 \leq i \leq n$ then $p' = \epsilon < q'$. Finally, if $p, q \in s$ then
$$\begin{cases} p' < q' & \text{if } i < p' \text{ and } i < q' \text{ for some } 1 \leq i \leq n \\ \{p', q'\} \text{ conflict-free} & \text{otherwise} \end{cases}$$
2. $\xi(M_1, \ldots, M_n)[s] \rightarrow \xi(P_1[1 \cdot (s \circ \uparrow)], \ldots, P_n[1 \cdot (s \circ \uparrow)])$. As in the previous case.
3. $P[s][t] \rightarrow P[s \circ t]$. Then either $p, q \in P$, or $p, q \in s$, or $p, q \in t$. In each case p', q' are unique and $p' < q'$.
4. $1[id] \rightarrow 1$ or $\uparrow \circ id \rightarrow \uparrow$. These cases hold trivially since there are no symbol positions in M.
5. $1[P \cdot s] \rightarrow P$. Then $p, q \in P$ and p', q' are unique, and $p' < q'$.
6. $id \circ s \rightarrow s$. Then $p, q \in s$ and p', q' are unique, and $p' < q'$.
7. $\uparrow \circ (P \cdot s) \rightarrow s$. Analogous to the previous case.
8. $(s1 \circ s2) \circ s3 \rightarrow s1 \circ (s2 \circ s3)$. Then either $p, q \in s_1$, or $p, q \in s_2$ or $p, q \in s_3$. In all cases p', q' are unique and $p' < q'$.
9. $(P \cdot s) \circ t \rightarrow P[t] \cdot (s \circ t)$. If $p, q \in P$ or $p, q \in s$ then p', q' are unique and $p' < q'$. Otherwise $p, q \in t$ and we reason as follows:
$$\begin{cases} p' < q' & \text{if } 1.2 < p' \text{ and } 1.2 < q' \\ p' < q' & \text{if } 2.2 < p' \text{ and } 2.2 < q' \\ \{p', q'\} \text{ conflict-free} & \text{otherwise} \end{cases}$$

- rewrite-step at an internal position.

1. $M = f(M_1, \ldots, M_n)$ and $M_i \xrightarrow{u}_{\sigma} M_i'$ for some $i \in 1..n$. If $p, q \in M_j$ with $i \neq j$ then the result is direct, if $i = j$ then we use the induction hypothesis. Otherwise $p = \epsilon$ in which case p' is unique and $p' < q'$ for all $q' \in q\llbracket u \rrbracket$.
2. $M = \xi(M_1, \ldots, M_n)$. As in the previous case.
3. $M = P[s]$. Then either $p, q \in P$ or $p, q \in s$. In both cases we apply the induction hypothesis.
4. $M = P \cdot s$ or $M = s \circ t$. As in the previous case.

•

The reader should note that Lemma 8.33 no longer holds if *arbitrary* positions in M are considered. Indeed, recall that the substitution operator traverses function and binder symbols.

Corollary 8.34 (Descendants of symbol positions which are one above the other) Let $p, q \in SPos(M)$ with $p < q$. Let $v : M \twoheadrightarrow_{\sigma} N$ and let p' be a v-descendant of p and q' a v-descendant of q in N. Then either $p' < q'$ or $\{p', q'\}$ is a conflict-free set in N.

Proof. By induction on the length n of v.

- $n = 0$. The result holds directly.
- $n > 0$. Suppose $v = \phi; u$ and $M \twoheadrightarrow_{\sigma}^{\phi} M' \xrightarrow{u}_{\sigma} N$. Since $p\llbracket v \rrbracket p'$ and $q\llbracket v \rrbracket q'$ there exist (unique) positions $p_1, q_1 \in SPos(M')$ such that $p\llbracket \phi \rrbracket p_1$ and $p_1\llbracket u \rrbracket p'$, and $q\llbracket \phi \rrbracket q_1$ and $q_1\llbracket u \rrbracket q'$.

  By the induction hypothesis two cases may arise:

  1. Either, $p_1 < q_1$. Then by Lemma 8.33 we are done.
  2. Or, $\{p_1, q_1\}$ is a conflict-free set in M'. Then by Lemma 8.29 the set $\{p', q'\}$ is conflict-free in M and we are done.

•

Lemma 8.35 (One-step descendants of lined-up symbol positions) Let $p, q \in SPos(M)$ with p lined-up with q. Let $M \xrightarrow{u}_{\sigma} N$ and let p' be a u-descendant of p and q' a u-descendant of q in N. Then one of the following holds:

1. either, $p' < q'$,
2. or, $\{p', q'\}$ is a conflict-free set in N,
3. or, p' is lined-up with q' in N.

Proof. By a close inspection of the rewrite rules in the σ-calculus. We proceed by induction on the (length of the) position where the rewrite step takes place. Also, we shall assume M is either a term of sort **T** or **S**.

- rewrite-step at the root position.
  1. $f(M_1, \dots, M_n)[s] \rightarrow f(M_1[s], \dots, M_n[s])$. Then
     - If $p, q \in M_i$ with $1 \leq i \leq n$ then p', q' are unique and p' is lined-up with q'.
     - If $p = 1$ and $q \in s$ then $p' = \epsilon < q'$ for all $q' \in q[\![u]\!]$.
     - If $p, q \in s$ then one of the following cases holds:
       $\begin{cases} p' \text{ lined-up with } q' & \text{if } i < p' \text{ and } i < q' \text{ for some } q \leq i \leq n \\ \{p', q'\} \text{ conflict-free} & \text{otherwise} \end{cases}$
     - $p \in M_i$ for some $1 \leq i \leq n$ and $q \in s$.
       $\begin{cases} p' \text{ lined-up with } q' & \text{if } i < q' \\ \{p', q'\} \text{ conflict-free} & \text{otherwise} \end{cases}$
  2. $\xi(M_1, \dots, M_n)[s] \rightarrow \xi(M_1[\mathit{lift}(s)], \dots, M_n[\mathit{lift}(s)])$. As in the previous case.
  3. $P[s][t] \rightarrow P[s \circ t]$. If $p, q \in P$, or $p, q \in s$, or $p, q \in t$ then p', q' are unique and p' is lined-up with q'. If $p \in M$ and $q \in s$ or $q \in t$ then the same holds. If $p \in s$ and $q \in t$ then also p', q' are unique and p' is lined-up with q'. No other cases are possible.
  4. $1[id] \rightarrow 1$ or $\uparrow \circ id \rightarrow \uparrow$. These cases hold trivially since there are no symbol positions in M.
  5. $1[P \cdot s] \rightarrow P$. Then $p, q \in P$ and p', q' are unique, and p' is lined-up with q'.
  6. $id \circ s \rightarrow s$. Then $p, q \in s$ and p', q' are unique, and p' lined-up with q'.
  7. $\uparrow \circ (P \cdot s) \rightarrow s$. Analogous to the previous case.
  8. $(s1 \circ s2) \circ s3 \rightarrow s1 \circ (s2 \circ s3)$. Then p', q' are unique and p' is lined-up with q'.
  9. $(P \cdot s) \circ t \rightarrow P[t] \cdot (s \circ t)$. Then
     - If $p, q \in P$ or $p, q \in s$ then p', q' are unique and p' is lined-up with q'.
     - If $p \in P$ and $q \in t$ (the case $p \in s$ and $q \in t$ is analogous) then we reason as follows:
       $\begin{cases} p' \text{ lined-up with } q' & \text{if } 1.2 < q' \\ \{p', q'\} \text{ conflict-free} & \text{otherwise} \end{cases}$
     - If $p, q \in t$ then we reason as follows:
       $\begin{cases} p' \text{ lined-up with } q' & \text{if } 1.2 < p' \text{ and } 1.2 < q' \\ p' \text{ lined-up with } q' & \text{if } 2.2 < p' \text{ and } 2.2 < q' \\ \{p', q'\} \text{ conflict-free} & \text{otherwise} \end{cases}$
- rewrite-step at an internal position.
  1. $M = f(M_1, \dots, M_n)$ and $M_i \xrightarrow{u}_\sigma M_i'$ for some $i \in 1..n$. Then either $p, q \in M_j$ with $j \neq i$ and the result is direct, or $p, q \in M_i$ and we may use the induction hypothesis.
  2. $M = \xi(M_1, \dots, M_n)$. As in the previous case.
  3. $M = P[s]$. Then if $p, q \in P$ or $p, q \in s$ we apply the induction hypothesis. Otherwise $p \in P$ and $q \in s$ in which case p' is lined-up with q' for all $p' \in p[\![u]\!]$ and $q' \in q[\![u]\!]$.
  4. $M = P \cdot s$. We use the induction hypothesis.
  5. $M = s \circ t$. Analogous to the case $M = P[s]$.

$\bullet$

Corollary 8.36 (Descendants of lined-up symbol positions) Let $p, q \in \mathit{SPos}(M)$ with p lined-up with q. Let $v : M \twoheadrightarrow_\sigma N$ and let p' be a v-descendant of p and q' a v-descendant of q in N. Then one of the following holds:

1. $p' < q'$, or
2. $\{p', q'\}$ is a conflict-free set in N, or
3. p' is lined-up with q' in N.

Proof. The proof makes use of Lemma 8.35, Lemma 8.33 and Lemma 8.29. We proceed by induction on the length n of v.

- $n = 0$. The result holds directly.
- $n > 0$. Suppose $v = \phi; u$ and $M \twoheadrightarrow^{\phi}_{\sigma} M' \xrightarrow{u}_{\sigma} N$. Since $p\llbracket v\rrbracket p'$ and $q\llbracket v\rrbracket q'$ there exist (unique) positions $p_1, q_1 \in SPos(M')$ such that $p\llbracket \phi\rrbracket p_1$ and $p_1\llbracket u\rrbracket p'$, and $q\llbracket \phi\rrbracket q_1$ and $q_1\llbracket u\rrbracket q'$.

  By the induction hypothesis three cases may arise:

  1. Either, $p_1 < q_1$. Then by Lemma 8.33 we are done.
  2. Or, p_1 is lined-up with q_1 in M'. Then by Lemma 8.35 we are done.
  3. Or, $\{p_1, q_1\}$ is a conflict-free set in M'. Then by Lemma 8.29 the set $\{p', q'\}$ is conflict-free in M and we are done.

•

Note that in Corollary 8.36 if v is a σ-derivation to σ-normal form then the case p' lined-up with q' in N is not possible. The proof of Proposition 8.31, however, is not a direct consequence of this fact since a priori it is not clear that $p' < q'$ in N, where p' (resp. q') is the v-descendant of the position of the head symbol of r (resp. u), assures us that r' nests u'. We shall see that indeed r' nests u'.

Proof.[of Proposition 8.31]
Let p (resp. q) be the position of the head symbol of r (resp. u) in M. First an observation.

Observation: Note that $r\langle\langle v\rangle\rangle r'$ implies the following:

- Let $p\llbracket v\rrbracket p'$. For every position o in the pattern of r there is a unique o' with $o\llbracket v\rrbracket o'$ such that $p' \leq o'$.
- Let o and $\texttt{prev}(o, M)$[7] be positions in the pattern of r. Let o' and o'' be the unique (by the previous item) symbol positions such that $o\llbracket v\rrbracket o'$ and $\texttt{prev}(o, M)\llbracket v\rrbracket o''$ and $p' \leq o'$ and $p' \leq o''$. Then $o'' = \texttt{prev}(o', N)$. Moreover, if the term at position o is in the i-th argument of the function or binder symbol at position $\texttt{prev}(o, M)$ then also the term at position o' is in the i-the argument of the function or binder symbol at position o''.

This observation simply states that the pattern of r may be reconstructed in r'.

Suppose $p\llbracket v\rrbracket p'$ and $q\llbracket v\rrbracket q'$. By Corollary 8.36 either $p' < q'$, or $\{p', q'\}$ is a conflict-free set in N, or p' is lined-up with q' in N. Since N is a σ-normal form, the last case is not possible. If $\{p', q'\}$ is a conflict-free set in N, then r' and u' are disjoint and we are done. Otherwise, suppose $p' < q'$. We are left to verify that r' and u' do not overlap.

Suppose that u' overlaps r', that is, there is a position o' in the pattern of r' such that o' is also a position in the pattern of u'. We have $p' < q' \leq o'$, as Figure 8.4 illustrates. By the previous observation applied to r there must be a position o_1 in the pattern of r such that $o_1\llbracket v\rrbracket o'$. Likewise, applying the observation to u there must be a position o_2 in the pattern of u such that $o_2\llbracket v\rrbracket o'$. But this contradicts the fact that the ancestor relation is a (partial) function. Therefore, r' nests u'.

•

Proposition 8.32 may be proved in a similar manner.

[7] Recall that $\texttt{prev}(p.n, M) \overset{\text{def}}{=} p$, where $n \in \mathbb{N}$ and $p.n \in Pos(M)$.

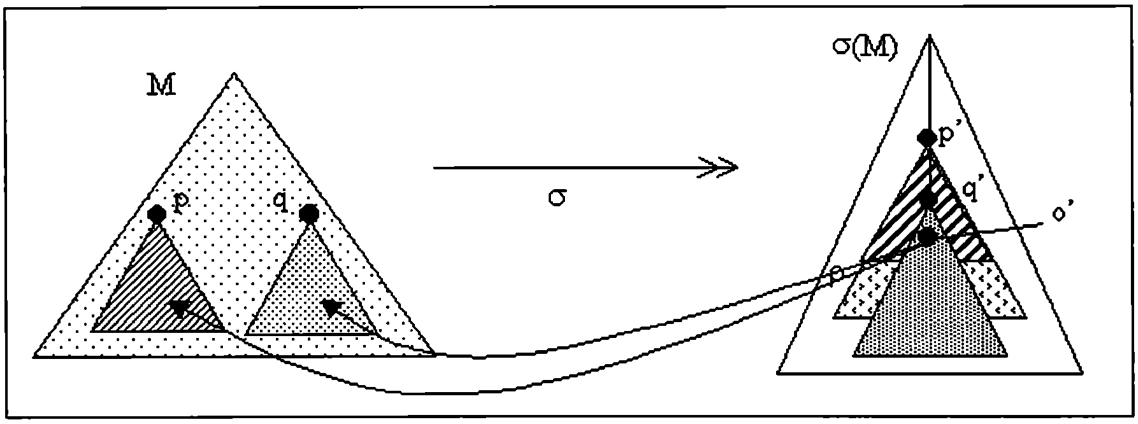


Figure 8.4: Non-overlapping redexes may not have overlapping correspondents.

8.3 From Standard $fo(\mathcal{R})_\sigma$-derivations to Standard $\mathcal{R}$-derivations

We shall now show that standard $fo(\mathcal{R})_\sigma$-derivations project onto standard $\mathcal{R}$-derivations via σ-interpretation (Hardin interpretation). This requires recalling the notion of finite developments and standard derivations.

Definition 8.37 (Developments) A *development* of a set of non-overlapping redexes S in M, in a $SERS_{db}$ $\mathcal{R}$ is an $\mathcal{R}$-derivation $\Upsilon : M_0 \xrightarrow{u_0} M_1 \xrightarrow{u_1} M_2 \xrightarrow{u_2} \dots$ which only contracts residuals of redexes in S. Υ is a *complete development* if no residuals of redexes in S remain.

Thus in a development of a set of redexes S of a term M no newly created redexes may be contracted, only residuals of those redexes in S are allowed to be contracted. The property of *Finite Developments (FD)* states that all developments of a set of non-overlapping redexes are finite. *FD* is usually proved in the setting of orthogonal (first or higher-order systems) in which case any set of redexes in any term is always non-overlapping. However, in the case of left-linear ambiguous systems *FD* still makes sense provided a set S of non-overlapping redexes is considered, and that the residuals of two non-overlapping redexes is non-overlapping. The latter property may be verified in first-order term rewrite systems and $SERS_{db}$, among other higher-order rewrite formalisms, in the style of Proposition 8.31.

Proposition 8.38 (Finite Developments) All developments of redexes in a term M in a $SERS_{db}$ $\mathcal{R}$ are finite and cofinal (they end in the same term).

For a proof of *FD* in the setting of (orthogonal) *SERS* see [Kha01]. *FD* for regular *CRS* may be found in [Klo80], for the more general *HRS* (and correcting [Klo80]) see [Oos97]. In fact, the following strengthening of *FD* is seen to hold [Oos94]:

Proposition 8.39 (Strong Finite Developments) All developments of a finite set of non-overlapping redexes in a term M in a $SERS_{db}$ $\mathcal{R}$ are finite and cofinal. Moreover, all complete developments of the same set of redexes induce the same residual relation.

Our interest in strong *FD* is that it shall be used for defining standard derivations following the Klop-Melliès [Mel01] presentation based on rewriting derivations. More precisely, we shall require strong *FD* in order to define *irreversible permutations*. It is also required for 'sequentializing' parallel rewrite steps.

Definition 8.40 Let $\mathcal{R}$ be a $SERS_{db}$. Let S be a set of disjoint $\mathcal{R}$-redexes in M. Let $M \rightrightarrows_{\mathcal{R}} N$ be a parallel $\mathcal{R}$-rewrite step (Def. 7.50) contracting only redexes in S. A *sequentialization* of the rewrite step $M \rightrightarrows_{\mathcal{R}} N$ is any complete development of S in M.

Note that if $M \rightrightarrows_{\mathcal{R}} N$ then any sequentialization of S in M is a derivation ending in N. This follows from the Strong Finite Developments proposition.

Definition 8.41 (Reversible and irreversible permutations) Let $\mathcal{R}$ be a $SERS_{db}$.

1. Let R and U be $\mathcal{R}$-redexes in M occurring at disjoint positions. A *reversible permutation* is a pair of derivations $R; U' \Diamond U; R'$ such that U' is the unique R-residual of U, and R' is the unique U-residual of R.
2. Let R and U be $\mathcal{R}$-redexes in M such that R occurs in an argument of U. An *irreversible permutation* is a pair of derivations $R; U' \rhd U; \Upsilon$ such that U' is the unique R-residual of U, and the derivation Υ is a complete development of the (possibly empty set of) U-residuals of R.

The reversible permutation relation is symmetric. It relates any two complete developments of a fixed pair of disjoint redexes in a term.

Definition 8.42 (Standardization preorder, Lévy permutation equivalence) Let $\mathcal{R}$ be a $SERS_{db}$.

1. A *reversible (resp. irreversible) standardization step* is a pair of $\mathcal{R}$-derivations $\Upsilon \overset{r}{\Rightarrow} \Phi$ (resp. $\Upsilon \overset{i}{\Rightarrow} \Phi$) such that $\Upsilon = \Upsilon_1; \Psi; \Upsilon_2$, $\Phi = \Upsilon_1; \Omega; \Upsilon_2$, and $\Psi \Diamond \Omega$ (resp. $\Psi \rhd \Omega$). In either case we write $\Upsilon \overset{1}{\Rightarrow} \Phi$. The *standardization preorder* $\Rightarrow$ is the least reflexive and transitive relation containing $\overset{1}{\Rightarrow}$.
2. The *Lévy permutation equivalence* $\equiv$ is defined as the least equivalence relation containing $\Rightarrow$.
3. The *reversible permutation equivalence* $\simeq$ is defined as the least equivalence relation containing $\overset{r}{\Rightarrow}$.

Since first-order rewrite systems are a particular case of $SERS_{db}$ the following definition of standard derivations apply to them too.

Definition 8.43 (Standard derivation) Let $\mathcal{R}$ be a $SERS_{db}$. A derivation $\Upsilon : M \twoheadrightarrow_{\mathcal{R}} N$ is *standard* in $\mathcal{R}$ when there is no sequence:

$$\Upsilon = \Upsilon_0 \overset{r}{\Rightarrow} \Upsilon_1 \overset{r}{\Rightarrow} \ldots \overset{r}{\Rightarrow} \Upsilon_{k-1} \overset{i}{\Rightarrow} \Upsilon_k$$

of $k-1$ reversible steps $\overset{r}{\Rightarrow}$ followed by an irreversible step $\overset{i}{\Rightarrow}$, for $k \geq 1$. Or equivalently, if Υ is in normal form with respect to $\overset{i}{\Rightarrow}$-rewriting modulo $\simeq$.

The reversible standardization steps are necessary for the nested and nesting redexes of an irreversible standardization step to 'meet' each other.

Example 8.44 Let $\mathcal{R} = \{f(X,a) \to b,\ c \to c,\ d \to a\}$ be the *OTRS* of Example 8.7 and consider the derivation (contracted redexes have been underlined): $f(d, f(d, \underline{d})) \to f(d, f(\underline{d}, a)) \to f(\underline{d}, f(a,a)) \to f(a, \underline{f(a,a)}) \to f(a,b)$. There is no irreversible standardization step applicable, however we may reorganize it via reversible standardization steps as follows: $f(d, f(d, \underline{d})) \to f(\underline{d}, f(d,a)) \to f(a, f(\underline{d}, a)) \to f(a, \underline{f(a,a)}) \to f(a,b)$. The final two rewrite steps constitute an irreversible standardization redex.

Before ending the subsection we shall define how to project $fo(\mathcal{R})_\mathcal{W}$ derivations onto $\mathcal{R}$-derivations, for $\mathcal{R}$ an essentially first-order $SERS_{db}$ (Def. 7.35). We recall the *Projection Proposition* (Proposition 7.56) for it shall be used for projecting first-order derivations to the $SERS_{db}$ framework. Also, we shall seize the opportunity to recall the other main result of Chapter 7, namely, the *Simulation Proposition* (Proposition 7.46). The latter, although not used for defining the projection of a derivation, shall be used when defining our standardization procedure, as observed in the introduction to the chapter.

Proposition 7.56 (Projection Proposition) Let $\mathcal{R}$ be a $SERS_{db}$ and let $fo(\mathcal{R})_\mathcal{W}$ be its first-order version, where $\mathcal{W}$ is a basic substitution calculus satisfying the scheme. If $M \to_{fo(\mathcal{R})_\mathcal{W}} N$ then $\mathcal{W}(M) \rightrightarrows_{\mathcal{R}} \mathcal{W}(N)$.

That is to say, a $fo(\mathcal{R})_\mathcal{W}$-step u may be simulated by a *parallel* $\mathcal{R}$-step (Def. 7.50). If u is a $\mathcal{W}$-step then $\mathcal{W}(M) = \mathcal{W}(N)$, otherwise $\mathcal{W}(M)$ rewrites to $\mathcal{W}(N)$ by rewriting the disjoint correspondents of u in $\mathcal{W}(M)$[8]. We recall the reader that this parallel rewrite relation is reflexive.

Proposition 7.46 (Simulation Proposition) Let $\mathcal{R}$ be a $SERS_{db}$ and let $fo(\mathcal{R})_\mathcal{W}$ be its first-order version. Suppose $M \to_{\mathcal{R}} N$ then

1. if $fo(\mathcal{R})_\mathcal{W}$ is an *ExERS* then $M \to_{fo(\mathcal{R})/\mathcal{W}} N$.

[8] The notion of correspondent has been defined for the basic substitution calculus σ only, and we shall make use of this result in the latter case only.

2. if $fo(\mathcal{R})_{\mathcal{W}}$ is a *FExERS* then $M \rightarrow_{fo(\mathcal{R})} \circ \twoheadrightarrow_{\mathcal{W}} N$ where $\circ$ denotes relation composition.

The second item of the Simulation Proposition, the item of our interest in this chapter, may be interpreted as: all derivations in a $SERS_{db}$ $\mathcal{R}$ may be 'implemented' by derivations in the first-order setting via its first-order version. So now we consider what it means to project a derivation from the *FExERS* setting to the $SERS_{db}$ setting.

Definition 8.45 (Projection of $fo(\mathcal{R})_{\mathcal{W}}$-derivations) Let ϕ be a $fo(\mathcal{R})_{\mathcal{W}}$-derivation. We define $\mathcal{W}(\phi)$ by induction on the length of ϕ:

$$
\begin{array}{lcl}
\mathcal{W}(e_M) & \stackrel{\mathrm{def}}{=} & e_{\mathcal{W}(M)} \\
\mathcal{W}(u;\psi) & \stackrel{\mathrm{def}}{=} & \left\{ \begin{array}{ll} e_{\mathcal{W}(M)}; \mathcal{W}(\psi) & \text{if } M \xrightarrow{u}_{\mathcal{W}} N \\ v_1; \ldots; v_n; \mathcal{W}(\psi) & \text{if } M \xrightarrow{u}_{fo(\mathcal{R})} N \end{array} \right.
\end{array}
$$

where $v_1; \ldots; v_n$ is any sequentialization of the parallel rewrite step $\mathcal{W}(M) \rightrightarrows_{\mathcal{R}} \mathcal{W}(N)$, the latter of which results from applying the Projection Proposition to the rewrite step $M \xrightarrow{u}_{fo(\mathcal{R})} N$.

In full precision, $\mathcal{W}$ applied to a $fo(\mathcal{R})_{\mathcal{W}}$-derivation yields an $\simeq$-equivalence class of $\mathcal{R}$-derivations. In this section we shall only deal with the case in which $\mathcal{W}$ is the σ-calculus.

Remark 8.46 It seems appropriate to shed some light on the motivation behind introducing both the usual first-order residual relation for *FExERS* ($[\bullet]$) and the new correspondence relation for *FExERS* ($\langle\langle\bullet\rangle\rangle$). Recall that we shall 'implement' a $SERS_{db}$-derivation Υ as a *FExERS*-derivation v (see Figure 8.5) by means of the Simulation Proposition. We then shall apply first-order standardization on v making use of the usual first-order residual relation yielding a standard (in the first-order framework) derivation ϕ. Finally, we are left to verify that the projection of ϕ, namely $\sigma(\phi)$, is a standard derivation in the higher-order framework. This is where the correspondence relation comes in, since it shall allow us to trace the destination of $fo(\mathcal{R})$-redexes appearing in ϕ, in the derivation $\sigma(\phi)$.

$$\begin{tikzcd}
\Upsilon \arrow[d, "\text{Simulation}"'] & & \sigma(\phi) \\
v \arrow[rr, squiggly, "\begin{array}{c}\text{first-order}\\ \text{standardization}\\ \text{(residual relation } [\bullet]\text{)}\end{array}"'] & & \phi \arrow[u, "\text{Projection (correspondence relation } \langle\langle\bullet\rangle\rangle\text{)}"']
\end{tikzcd}$$

Figure 8.5: Standardization Procedure

8.3.1 Standardization

Definition 8.47 ((Un)contributable symbol) Let $\mathcal{R}_{\mathcal{W}} = (\Gamma, \mathcal{R}, \mathcal{W})$ be a left-linear *FExERS*. A function or binder symbol $g \in \Gamma$ of arity n is called *uncontributable* in $\mathcal{R}_{\mathcal{W}}$ if

1. either, g does not occur on the *LHS* of any rule in $\mathcal{R}_{\mathcal{W}}$,
2. or, g occurs on the *LHS* of a rule in $\mathcal{R}_{\mathcal{W}}$ only under the form $g(X_1, .., X_n)$ (i.e. it occurs applied to metavariables).

A symbol in Γ which is not uncontributable is called *contributable*.

The idea behind uncontributable symbols is that redexes strictly below them cannot create redexes above them (Lemma 8.50).

Example 8.48 The λ-symbol is an example of an uncontributable symbol in the $\lambda\sigma$-calculus, i.e. in $fo(\beta)_\sigma$. Whereas, the application symbol is a contributable symbol in $\lambda\sigma$ due to the *Beta*-rule. As a further example, for any essentially first-order $SERS_{db}$ $\mathcal{R}$ the cons-symbol $\bullet \cdot \bullet$ is uncontributable in $\mathcal{R}_\sigma$.

Although the definition of uncontributable symbol as it stands shall suffice for our purposes a more general formulation is possible. One may define the notion of i-uncontributable symbol for i a non-zero natural number. A symbol g of arity n is i-uncontributable for $i \leq n$ if either the first case of Def. 8.47 holds, or g occurs on the *LHS* or a rule in $\mathcal{R}_\mathcal{W}$ only under the form $g(M_1, \ldots, M_i, X, M_{i+1}, \ldots, M_n)$. That is to say, it may only occur on the *LHS* of a rewrite rule with its ith argument applied to a metavariable. We may then define a symbol of arity n as uncontributable if it is i-uncontributable for all $1 \leq i \leq n$.

Also note that the notion of uncontributable symbol does not coincide with that of constructor symbol in a constructor TRS [Klo92]: given a constructor TRS there may be constructor symbols which are contributable (e.g. 's' in $f(s(s(x))) \to x$) and likewise there may be uncontributable symbols that are not constructor symbols (e.g. f in $f(x) \to a$).

The proof of the following lemma [Mel00, Lemma 6.3] originally formulated for the $\lambda\sigma$-calculus holds without further ado for an arbitrary *FExERS* $\mathcal{R}_\mathcal{W}$, and is included here for the sake of completeness.

Lemma 8.49 Let $\mathcal{R}_\mathcal{W}$ be a *FExERS*. Suppose that a position p is strictly above a redex $P \xrightarrow{r} Q$. For every derivation $\phi = r; \psi$:

1. either, ϕ preserves p,
2. or, ϕ is equal modulo $\simeq$ to a derivation $\phi_1; u; v; \phi_2$ such that ϕ_1 preserves the position p, the position p is strictly above the redex u, and the redex v is above p.

Proof. We begin by proving a claim: if $u_1; ..; u_j : P \twoheadrightarrow R$ is a $\mathcal{R}_\mathcal{W}$-derivation which preserves the position p, and contracts a redex strictly below p then it may be reorganized into a derivation $v_1; \ldots; v_j$ which is $\simeq$-equivalent to $u_1; ..; u_j$ such that v_j is strictly below p. Let $u_1; ..; u_j : P \twoheadrightarrow R$ be a $\mathcal{R}_\mathcal{W}$-derivation which preserves the position p, and contracts a redex strictly below p. Let u_k be the last redex strictly below p. If $k < j$, the redex u_{k+1} which is not strictly below p may be permuted *reversibly* before u_k. Observe that the resulting derivation $u_1; \ldots; u'_k; u'_{k+1}; \ldots; u_j$ preserves p, and contracts the redex u'_{k+1} strictly below p. Repeating the process $j - k$ times, one constructs a derivation $v_1; \ldots; v_j \simeq u_1; \ldots; u_j$ whose last redex v_j is strictly below p.

Let $\phi = r_1; \ldots; r_n$ be a derivation and p an occurrence which strictly above r_1. We suppose that ϕ does not preserve p. Let r_{j+1} be the first redex in ϕ which is above p. By the above claim, there exists a derivation $v_1; \ldots; v_j$ equal to $r_1; \ldots; r_j$ modulo $\simeq$ whose last redex $u = v_j$ is strictly below p. We conclude.

•

Lemma 8.50 Let $\mathcal{R}_\sigma$ be a left-linear *FExERS*. Suppose that a position p is strictly above a redex $P \xrightarrow{r} Q$. Every standard derivation $\phi = r; \psi$ preserves p when:

1. either, p is a g-node for g an uncontributable symbol in $\mathcal{R}_\sigma$,
2. or, p is a g-node for g a function or binder symbol in $\mathcal{R}_\sigma$ and ψ is a σ-derivation.

Proof. Given the standard derivation $\phi = r; \psi$ and the occurrence p strictly above r we apply Lemma 8.49. If the first case holds then we are done, we shall see that given the hypothesis the second case cannot hold.

Suppose that v is above the occurrence p and that p is strictly above u in a pair $M \xrightarrow{u} N \xrightarrow{v} P$. The derivation $u; v$ cannot be standard, unless u creates v. Now, in at least the following two cases creation is not possible:

1. When the occurrence p is the occurrence of an uncontributable symbol in $\mathcal{R}_\sigma$. Note that this includes the cons-node $\bullet \cdot \bullet$.
2. When the occurrence p is a function or binder symbol node and u is a σ-redex the only possible pattern of creation is when u is a $(Func)_f$-redex for some function symbol f or a $(Bind)_\xi$-redex for some binder symbol ξ and v is an $fo(\mathcal{R})$-redex. This may be seen to hold due to the fact that function and binder symbols are uncontributable in the σ-calculus, or stated equivalently: substitutions are created "downwards". For example, the pair $M = g(h(c)[id]) \to_{func_h} g(h(c[id])) \to c$ where $\mathcal{R} \overset{\text{def}}{=} \{g(h(X)) \to c\}$, $p = \epsilon$ in M, the position at which v occurs in N is ϵ and the position at which u occurs in M is 1.

∎

The restriction to left-linear *FExERS* in Lemma 8.50 is essential for otherwise a redex strictly below p could create a redex above p.

Lemma 8.51 Let $\mathcal{R}_\sigma$ be a left-linear *FExERS*. Let $v : M \twoheadrightarrow N$ be a standard $\mathcal{R}_\sigma$-derivation with N in σ-normal form. Then no $\mathcal{R}_\sigma$-redex ever appears below a cons-node $\bullet \cdot \bullet$.

Proof. By contradiction. Suppose there exists an r_i contracted in $\psi = r_1; \ldots; r_n$ strictly below the occurrence p of a cons-node. Then by Lemma 8.50(1) the derivation $r_i; \ldots; r_n$ preserves p. Since N is a pure term we arrive at a contradiction.

∎

A remark before proceeding to the main proposition.

Remark 8.52 Let $\mathcal{R}$ be an essentially first-order $SERS_{db}$ and let $fo(\mathcal{R})_\sigma$ be its first-order version. Let r be a $fo(\mathcal{R})$-redex in M occurring at a position p with r not in the body of a substitution. Let R be its (unique) corresponding $\mathcal{R}$-redex in $\sigma(M)$ occurring at a position p', i.e. $r\langle\langle v\rangle\rangle R$ for $v : M \rightarrowtail_\sigma \sigma(M)$ any σ-derivation from M to σ-normal form. Then for every $q' \in SPos(\sigma(M))$ with $q' < p'$, we have $q < p$ where $q \in SPos(M)$ is the (unique) ancestor of q' (see Figure 8.6). Note that although the ancestor relation, in general, is not a (partial) function, in the case of the σ-calculus and when dealing with positions of (function or binder) symbols, we do indeed obtain a unique ancestor (see the rewrite rules in Figure 8.3). For example, in the rewrite-step $M = f(M_1, \ldots, M_n)[s] \rightarrow_{Func_f} f(M_1[s], \ldots, M_1[s]) = N$ both ϵ and 1 in M are ancestors of ϵ in N, yet when considering solely the positions of f-symbols it is just 1 in M we are interested in.

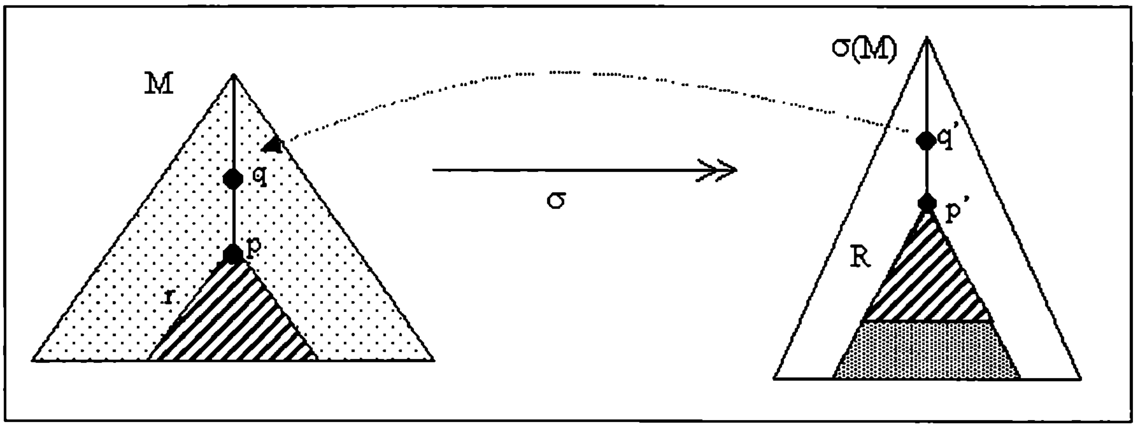


Figure 8.6: Ancestors of symbol positions.

We are now ready to prove the main proposition.

Proposition (8.2. Let $\mathcal{R}$ be a left-linear essentially first-order $SERS_{db}$. Every standard derivation v : $M \twoheadrightarrow N$ in $fo(\mathcal{R})_\sigma$ with N in σ-normal form is projected onto a standard derivation $\sigma(v) : \sigma(M) \twoheadrightarrow N$ in $\mathcal{R}$.

Proof. First note that by Lemma 8.51 no $fo(\mathcal{R})$-redex contracted in $v : M \twoheadrightarrow N$ ever occurs inside a substitution s, since $fo(\mathcal{R})$-redexes are of sort T and thus would have to take place inside a cons-node. As remarked in [Mel00] this does not hold for σ-redexes as the following example $\Upsilon : M \twoheadrightarrow_\sigma N$ illustrates[9]:

$$\Upsilon : M = 1[(1 \cdot id) \circ (N \cdot id)] \rightarrow_{Map} 1[1[N \cdot id] \cdot (id \circ (N \cdot id))] \rightarrow_{VarCons} 1[N \cdot id] \rightarrow_{VarCons} N$$

This property implies that every $fo(\mathcal{R})$-redex contracted in v has a unique correspondent $\mathcal{R}$-redex in $\sigma(v)$. Let $\sigma(v) = R_1; \ldots; R_o$ and let $\rho : \{1, \ldots, o\} \rightarrow \{1, \ldots, n\}$ be the function which associates to any $\mathcal{R}$-redex R_k in $\sigma(v)$ the unique $fo(\mathcal{R})$-redex $r_{\rho(k)}$ in $v = r_1; \ldots; r_n$ to which it corresponds. Let R_k and R_{k+1} be two consecutive $\mathcal{R}$-redexes in $\sigma(v)$. Note that the $fo(\mathcal{R})_\sigma$-derivation $r_i; \ldots; r_j = r_{\rho(k)+1}; \ldots; r_{\rho(k+1)-1}$ between $r_{\rho(k)}$ and $r_{\rho(k+1)}$ contracts only σ-redexes.

We shall now show that:

[9] Although brackets are kings in $\lambda\mathbf{x}$ [KOO01b], they are 'almost' kings in $\lambda\sigma$.

Case 1. every reversible standardization step $\sigma(v) \overset{r}{\Rrightarrow} \Phi$ in $\mathcal{R}$ may be mirrored as a (non-empty) series of reversible standardization steps $v \overset{r}{\Rrightarrow} \dots \overset{r}{\Rrightarrow} \phi$ in $fo(\mathcal{R})_\sigma$, where $\sigma(\phi) = \Phi$, and

Case 2. every irreversible standardization step $\sigma(v) \overset{i}{\Rrightarrow} \Phi$ in $\mathcal{R}$ may be mirrored as a (non-empty) series of standardization steps $v \overset{r}{\Rrightarrow} \dots \overset{r}{\Rrightarrow} \phi' \overset{i}{\Rrightarrow} \dots \overset{i}{\Rrightarrow} \phi$ in $fo(\mathcal{R})_\sigma$, where $\sigma(\phi) = \Phi$.

Hence the result follows by reasoning by contradiction since every standardisation step acting on the projected higher-order rewrite derivation may be mimicked by projection-related standardisation steps of the same nature (reversible/irreversible) over first-order derivations. So we shall now focus on these two items.

Case 1. Suppose two $\mathcal{R}$-redexes R_k and R_{k+1} can be permuted using a reversible permutation, that is, $R_k; R_{k+1} \diamond R'_k; R'_{k+1}$. We construct a $fo(\mathcal{R})_\sigma$-derivation ϕ such that $v \simeq \phi$ and $\sigma(\phi) = R_1; \dots; R'_k; R'_{k+1}; \dots; R_p$. By Lemma 8.50, the derivation $r_i; \dots; r_j$ preserves the occurrence of any function or binder symbol strictly above $r_{\rho(k)}$. And, in particular, the lowest function or binder symbol g appearing above $R_k : \sigma(P) \to \sigma(Q)$ and R_{k+1} in the term $\sigma(P)$ which, by Remark 8.52, is strictly above $r_{\rho(k)}$ in P. Then the derivation $\psi = r_{\rho(k)}; r_i; \dots; r_j; r_{\rho(k+1)}$ may be reorganized modulo $\simeq$ into a derivation ψ' such that $\sigma(\psi') = R'_k; R'_{k+1}$ as follows: let p be the occurrence of g in P and assume $P|_p = g(N_1, \dots, N_m)$ and suppose $r_{\rho(k)}$ occurs in N_{l_1} and the head symbol of $r_{\rho(k+1)}$ occurs in N_{l_2} for $l_1, l_2 \in 1..m$ and $l_1 \neq l_2$:

1. First contract all the redexes in ψ prefixed by $p.l_2$
2. Second contract $r_{\rho(k+1)}$,
3. Third contract the (unique) residual of $r_{\rho(k)}$,
4. Finally contract the remaining redexes of ψ.

Case 2. Suppose two $\mathcal{R}$-redexes R_k and R_{k+1} can be permuted using an irreversible permutation $R_k; R_{k+1} \rhd R'_k; \Psi$. Observe the following:

Observation: by Remark 8.52 all the symbols in the redex pattern of (the ancestor of) R_{k+1} strictly above R_k in $\sigma(P)$ are present in P above the occurrence of $r_{\rho(k)}$. Moreover, none of these symbols occurs embraced by a substitution. This follows from two facts:

1. on the one hand, by Lemma 8.50, the derivation $r_i; \dots; r_j$ preserves all these symbols (in particular the lowest one), and
2. on the other, $r_{\rho(k+1)}$ is an $fo(\mathcal{R})$-redex hence its *LHS* contains no occurrences of the substitution operator $\bullet[\bullet]$.

Having concluded with our observation we proceed with the proof of Case 2. We consider two subcases, reasoning by contradiction in each one:

- The redex $r_{\rho(k)}$ in P occurs under an uncontributable symbol g belonging to the pattern of R_{k+1}. By Lemma 8.50 the path $r_i; \dots; r_n$ preserves every uncontributable symbol strictly above $r_{\rho(k)}$. Among these symbols is the symbol g involved in the pattern of R_{k+1}. The redex $r_{\rho(k+1)}$ is above the position of this symbol. We reach a contradiction.
- All symbols above $r_{\rho(k)}$ in P belonging to the pattern of R_{k+1} are contributable. Suppose that the two $\mathcal{R}$-redexes R_k and R_{k+1} can be permuted using an irreversible permutation $R_k; R_{k+1} \rhd R'_k; \Psi$; we shall arrive at a contradiction. Let p be the occurrence of the unique ancestor of the head symbol g of R_{k+1} in P, and $P|_p = g(M_1, .., M_m)$. By Lemma 8.50 the derivation $r_i; ...; r_j$ preserves p. Let $l \in 1..m$ such that $r_{\rho(k)}$ occurs in M_l. We may then reorganize modulo $\simeq$ the derivation $\psi = r_{\rho(k)}; r_i; \dots; r_j; r_{\rho(k+1)}$ obtaining ψ', as follows:
  1. First rewrite all redexes in $r_i; \dots; r_j$ prefixed by $p.1$,$p.2$,...,$p.l - 1$,$p.l + 1$,..,$p.m$ in turn (i.e. first all those prefixed by $p.1$, then those by $p.2$, and so on) and those disjoint to p.
  2. Second, rewrite all redexes prefixed by $p.l$ but disjoint to the (unique) residual of $r_{\rho(k)}$. At this moment the redex $r_{\rho(k+1)}$ must have emerged since

  - by the Observation there are no substitution symbols in M_l between p and the position of $r_{\rho(k)}$, and
  - $r_{j+1} = r_{\rho(k+1)}$, i.e. it is an $fo(\mathcal{R})$-redex and hence its *LHS* contains no occurrences of the substitution operator $\bullet[\bullet]$.

3. Thirdly, rewrite the (unique) residual of $r_{\rho(k)}$, say $r'_{\rho(k)}$, followed by the redexes in $r_i; \ldots; r_j$ prefixed by the occurrence of $r_{\rho(k)}$, i.e. those not rewritten in Step 1 or Step 2.
4. Finally, rewrite $r_{\rho(k+1)}$.

Note that $\psi' = \psi'_1; \psi'_2$ where ψ'_1 consists solely of σ-rewrite steps and $\psi'_2 = r'_{\rho(k)}; r_{k_1}; \ldots; r_{k_m}; r_{\rho(k+1)}$ for some $m \leq j - i$. Applying $m + 1$ irreversible standardization steps starting from ψ'_2 we may obtain $\psi'_3 = r'_{\rho(k+1)}; \psi_{r_{\rho(k)}}; \psi_{r_{k_1}}; \ldots; \psi_{r_{k_m}}$. Finally, setting $\psi = \psi'_1; \psi'_3$ we may conclude.

Note that Proposition 8.2 fails for *arbitrary* standard derivations $\upsilon : M \twoheadrightarrow N$ in $fo(\mathcal{R})_\sigma$.

Example 8.53 Let $\mathcal{R}$ be the λ-calculus, and let $\lambda\sigma = fo(\mathcal{R})_\sigma$ be its first-order version. Then the derivation starting from $M = ((\lambda(11))1)[(\lambda 1)c \cdot id]$, ϕ is a standard first-order derivation:

$$\phi : ((\lambda(11))1)[\underline{(\lambda 1)c} \cdot id] \rightarrow_{Beta} \underline{((\lambda(11))1)}[1[c \cdot id] \cdot id] \rightarrow_{Beta} (11)[1 \cdot id][1[c \cdot id] \cdot id]$$

However, $\sigma(\phi)$ is not standard in the $SERS_{db}$ framework.

$$\sigma(\phi) : (\lambda(11))\underline{((\lambda 1)c)} \rightarrow_\beta \underline{(\lambda(11))c} \rightarrow_\beta cc$$

The reader may have noticed that although the (only) two β-redexes in M are disjoint, they are lined-up (see Proposition 8.31). Thus although they occur at disjoint positions in M, when we project via σ their correspondents become nested. A solution proposed by Melliès [Mel01] is to divide the class of reversible permutations (used for standardization in the first-order framework) into two subclasses: one subclass which remains reversible, and another which is transformed into irreversible. So now we have two classes of irreversible permutations, the usual (say nesting) ones, and the new lined-up (lu) irreversible permutations. The latter are defined as follows. Let $\mathcal{R}_\sigma$ be a *FExERS*. Then $R; U' \rhd_{lu} U; R'$ where the $\mathcal{R}$-redex U is lined-up with the $\mathcal{R}$-redex R in M and U' is the unique R-residual of U, and R' is the unique U-residual of R. Under this extended definition of permutation (and induced notion of standardization) Proposition 8.2 seems (verifying the details is left to future work) to apply to arbitrary standard derivations $\upsilon : M \twoheadrightarrow N$ in $fo(\mathcal{R})_\sigma$. Note, however, that when implementing a higher-order derivation via the Simulation Proposition as illustrated in Figure 8.1 of the introduction to the chapter, the resulting derivation shall always end in a pure term.

The full standardization procedure takes the following form.

Definition 8.54 (Standardization Procedure) The standardization procedure applied to a derivation $\Upsilon : M \twoheadrightarrow_{\mathcal{R}} N$ of a left-linear essentially first-order $SERS_{db}$ $\mathcal{R}$ consists in the following steps (see Figure 8.1):

Step 1 (Simulation). Apply the Conversion Procedure to $\mathcal{R}$ obtaining a full first-order rewrite system $fo(\mathcal{R})_\sigma$ (Def. 7.35). Note that $fo(\mathcal{R})_\sigma$ shall always be ambiguous[10] even if $\mathcal{R}$ is orthogonal. The Simulation Proposition yields as output a $fo(\mathcal{R})_\sigma$-derivation $\upsilon : M \twoheadrightarrow_{fo(\mathcal{R})_\sigma} N$ implementing the $\mathcal{R}$-derivation Υ.

Step 2 (Standardization). Use the standardization procedure described in [Bou85] applicable to first-order left-linear ambiguous rewrite systems. The output is a $fo(\mathcal{R})_\sigma$-derivation ϕ such that $\phi \equiv \upsilon$ and ϕ is standard in $fo(\mathcal{R})_\sigma$.

Step 3 (Projection). Project the standard derivation. Define the $\mathcal{R}$-derivation Φ as $\sigma(\phi)$. Proposition 8.2 guarantees that Φ is standard in $\mathcal{R}$.

Note that the standardization procedure yields a unique standard derivation modulo reversible permutations. This procedure allows us to conclude with the following result:

Theorem 8.55 (Standardization for $SERS_{db}$) Let $\mathcal{R}$ be a left-linear essentially first-order $SERS_{db}$. Assume $\Upsilon : M \twoheadrightarrow_{\mathcal{R}} N$. Then there is a standard derivation $\Phi : M \twoheadrightarrow_{\mathcal{R}} N$.

The restriction to essentially first-order $SERS_{db}$ is necessary since it is this class of systems that may be converted to full first-order rewrite systems. We now consider a strong version of standardization.

[10] Assuming the alphabet contains some symbol of strictly positive arity.

8.3.2 Strong Standardization

This subsection deals with strong standardization of $SERS_{db}$-derivations. Given a derivation Υ in some left-linear essentially first-order $SERS_{db}$ $\mathcal{R}$, the Standardization Procedure provides us with a standard derivation Φ in $\mathcal{R}$. However, we would like Φ to be not just any standard derivation coinitial and cofinal with Υ, but also Lévy permutation equivalent to Φ, in other words, $\Phi \equiv \Upsilon$. We shall see that this transfer of standardization not only provides us with a standardization result but also yields strong standardization.

Proposition 8.56 Let v and ϕ be $fo(\mathcal{R})_\sigma$-derivations. If $v \overset{1}{\Rightarrow} \phi$ then $\sigma(v) \equiv \sigma(\phi)$.

Before providing a proof we would like to draw the readers attention to the statement of this proposition. It does not hold that $v \overset{1}{\Rightarrow} \phi$ implies $\sigma(v) \Rightarrow \sigma(\phi)$.

Example 8.57 Let $\mathcal{R}$ be the λ-calculus, and let $\lambda\sigma = fo(\mathcal{R})_\sigma$ be its first-order version. Then $v \overset{1}{\Rightarrow} \phi$ (more precisely, $v\Diamond\phi$) where (the redexes contracted at each step have been underlined in order to ease readability)

$$\begin{array}{l}
v : (\underline{(\lambda(11))1})[(\lambda 1)c \cdot id] \rightarrow_{Beta} (11)[1 \cdot id][\underline{(\lambda 1)c} \cdot id] \rightarrow_{Beta} (11)[1 \cdot id][1[c \cdot id] \cdot id] \\
\phi : ((\lambda(11))1)[\underline{(\lambda 1)c} \cdot id] \rightarrow_{Beta} \underline{(\lambda(11))1}[1[c \cdot id] \cdot id] \rightarrow_{Beta} (11)[1 \cdot id][1[c \cdot id] \cdot id]
\end{array}$$

However, $\sigma(\phi) \overset{1}{\Rightarrow} \sigma(v)$ (and $\sigma(v) \overset{1}{\not\Rightarrow} \sigma(\phi)$), since

$$\begin{array}{l}
\sigma(v) : \underline{(\lambda(11))((\lambda 1)c)} \rightarrow_\beta \underline{((\lambda(1))c)}((\lambda 1)c) \rightarrow_\beta c\underline{((\lambda 1)c)} \rightarrow_\beta cc \\
\sigma(\phi) : (\lambda(11))\underline{((\lambda 1)c)} \rightarrow_\beta \underline{(\lambda(11))c} \rightarrow_\beta cc
\end{array}$$

The reason for this is that $v\Diamond\phi$ by permuting two disjoint but lined-up redexes as illustrated in Example8.53.

Proof.[of Proposition 8.56] Let $v' : M \xrightarrow{r} N_1 \xrightarrow{u'} N$ for $\{r, u\}$ $fo(\mathcal{R})_\sigma$-redexes in M such that r does not nest u, and u' is the (unique) r-residual of u. It suffices to show that the claim holds for the following two cases:

Case 1. if $v'\Diamond\phi'$ for $\phi' = u; r'$ then $\sigma(v') \equiv \sigma(\phi')$, and

Case 2. if $v' \rhd \phi'$ for $\phi' = u; \phi''$ then $\sigma(v') \equiv \sigma(\phi')$.

Our analysis depends on whether r and u are σ or $fo(\mathcal{R})$-redexes in M and shall distinguish cases 1 and 2 as needed.

1. In either case, if r and u are σ-redexes then the result holds trivially.

2. Suppose u is a σ-redex and r a $fo(\mathcal{R})$-redex (the viceversa case is analogous). Then

$$\sigma(v') : \sigma(M) \rightrightarrows_{\mathcal{R}} \sigma(N_1) = \sigma(N) \text{ and } \sigma(\phi') : \sigma(M) = \sigma(N_2) \rightrightarrows_{\mathcal{R}} \sigma(N')$$

   By parametricity of σ over $fo(\mathcal{R})$-redexes (Corollary 8.26) the correspondents of $r \in M$ in $\sigma(N_2)$ are the same as those in $\sigma(M)$. Then any two sequentializations of the parallel $\mathcal{R}$-step shall yield equivalent derivations modulo $\simeq$.

3. Suppose both u and r are $fo(\mathcal{R})$-redexes in M. Here we distinguish the two subcases:

   - Reversible permutation (Case 1). Suppose $\{r, u\}$ are disjoint redexes in M. Then if the correspondents of $\{r, u\}$ via σ are disjoint in $\sigma(M)$ we may simply sequentialize the derivation of the corresponding redexes. Otherwise, by Corollary 8.30 we may assume that r is lined-up with u. Let $S = \{r_1, \ldots, r_n\}$ be the set of (pairwise disjoint by Corollary 8.30) correspondents of r in $\sigma(M)$. Then by Proposition 8.31 each correspondent u' of u is either disjoint with all redexes in S or is nested by some (one) redex in S. Finally, note that set of correspondents of u in $\sigma(M)$ are pairwise disjoint too by Corollary 8.30.

     Thus we may construct the standardization $\sigma(v') \Leftarrow \sigma(\phi')$ where $\sigma(v')$ rewrites all r_is in some order and then rewrites all the correspondents of u's in some order, and $\sigma(\phi')$ rewrites all (correspondents of) u in some order, and then all the (unique) correspondents of the r_is in some order.

- Irreversible permutation (Case 2). Suppose u nests r. Let $S = \{u_1, \ldots, u_n\}$ be the set of (disjoint by Corollary 8.30) correspondents of u in $\sigma(M)$. Then by Proposition 8.32 each correspondent of r in $\sigma(M)$ is either disjoint with all redexes in S or is nested by some (one) redex in S. Finally, note that the set of correspondents of r in $\sigma(M)$ is pairwise disjoint too by Corollary 8.30.

  Thus we may construct the standardization $\sigma(\upsilon') \Rightarrow \sigma(\phi')$ where $\sigma(\upsilon')$ rewrites all the correspondents of r in some order and then rewrites all the (unique correspondents of the) u_is in some order, and $\sigma(\phi')$ rewrites all the u_is in some order and then all the (correspondents of) r in some order.

▪

As a consequence of Proposition 8.56 if $\upsilon \equiv \phi$ then $\sigma(\upsilon) \equiv \sigma(\phi)$. We may now formulate the strong version of the standardization theorem.

Proposition 8.58 (Strong standardization for $SERS_{db}$**)** Suppose $\mathcal{R}$ is a left-linear essentially first-order $SERS_{db}$ and $\Upsilon : M \twoheadrightarrow_{\mathcal{R}} N$. Then Υ may be standardized to a unique (modulo reversible permutation equivalence) standard derivation $\Phi : M \twoheadrightarrow_{\mathcal{R}} N$ which is Lévy permutation equivalent to Υ.

Conclusions

This thesis is concerned with term rewriting and, in particular, calculi of explicit substitutions. We have considered perpetual reductions in calculi of explicit substitutions in Part I, we have dealt with the $\lambda\mathtt{x}$-calculus and then considered the more involved $\lambda\mathtt{ws}$-calculus. Part II augmented M.Abadi and L.Cardelli's object calculus ς with explicit substitutions and analyzed some difficulties arising when simulating the lambda calculus (more precisely, λv). Part III studied the encoding of higher-order rewriting in first-order rewriting and then considered the transfer of the Standardization Theorem from the first-order case to the higher-order one. A brief synopsis of the contents of the thesis together with hints at future research directions follows.

Part I: Perpetuality in Calculi of Explicit Substitutions

Calculi of explicit substitutions are non-orthogonal by nature. This state of affairs may be witnessed by considering how the class of derivations between terms is affected when a calculus is augmented with explicit substitutions. In particular, any two pure terms M and N such that there is a derivation from M to N are furnished with a rich supply of alternative derivations between them. Preservation of strong normalization (PSN) is in charge of verifying that we enrich with caution.

Let us expound further on this issue fixing the lambda calculus as our setting in order to simplify matters. Define a pair of pure λ-terms (M, N) as *bounded* if there exists $n \geq 0$ such that for all derivations $v : M \twoheadrightarrow_{\beta} N$ we have $|v| < n$. Then an appropriate condition for avoiding indiscriminated enrichment of derivations could be the following notion of *preservation of boundedness* (PB): let λe be a calculus of explicit substitutions for the λ-calculus; we say λe satisfies *preservation of boundedness* if for every bounded pair (M, N) the following holds: $\exists n \geq 0.\forall v : M \twoheadrightarrow_{\lambda e} N'$ with $e(N') = N$ we have $|v| < n$. Although PSN is not strictly equivalent with this notion, the more general techniques required to prove PSN are enough to prove preservation of boundedness. However, in some settings PB could be preferred over PSN. An example is the infinitary λ-calculus [KKSdV95], where potentially infinite normal forms are of interest. PSN would not be of much use in infinitary lambda calculus with explicit substitutions, however PB is applicable.

The techniques developed in order to prove PSN have not been fully exploited. Part I shows how these techniques may fruitfully be applied in order to yield perpetual rewrite strategies and inductive characterizations of strongly normalizing terms in calculi of explicit substitutions. In the case of $\lambda\mathtt{x}$ it is worth noting that these results have been applied with success in order to verify, via the Tait-Martin Löf-Girard proof method, strong normalization of a polymorphic lambda calculus with explicit substitutions. The latter calculus is defined and studied in Chapter 3. As for the $\lambda\mathtt{ws}$-calculus the presence of substitution composition makes the inductive characterization of its strongly normalizing terms a pleasant newcomer. Future lines of research are:

- An open problem is that of finding a maximal strategy for $\lambda\mathtt{x}$ (Remark 3.18). Although the strategy $\mathcal{F}_{\infty}(\bullet)$ is a candidate, we have not succeeded in verifying this. Note that this has also been left pending in [BH98] (together with the task of providing an inductive characterization of the set of strongly $\lambda\mathtt{x}$-normalizing terms which we have addressed in this thesis).

- It is rather unfortunate that nothing on strong normalization of typed $\lambda\mathtt{ws}$ has been said. The reducibility technique should be applicable with the aid of the characterization of the strongly $\lambda\mathtt{ws}$-normalizing terms.

- From a more general standpoint, the plethora of methods for proving PSN, as described in the introduction of the thesis, suggests the lack of a sufficiently general proof technique for dealing with this property.

Part II: Explicit Substitutions for a Calculus of Objects

The origins of calculi of explicit substitution stem in category theory and lambda calculus, however current widespread use of the object-oriented programming paradigm gives rise in a natural way to the question of whether the virtues of promoting metalevel substitution to the object-level in the context of foundational calculi for functional programming, make themselves present in analogous calculi for the object-oriented paradigm. In an attempt to answer this question Part II of this thesis puts Abadi and Cardelli's ς-calculus [AC96] under the microscope; it is presented in a de Bruijn indices setting and is augmented with explicit substitutions. Although the minimum properties of ς, such as confluence, are shown to be retained subtle issues concerning simulation of functional calculi have driven us to introduce a new substitution operator (invoke substitution) together with fields into the language. Compliance with the property of preservation of strong normalization in the presence of interaction between the new substitution operator and the usual explicit substitution operator is the issue of the last section of this part. As regards future research directions we mention:

- Type systems for the ς-calculus are thoroughly studied in [AC96], including subtypes and polymorphism. It would be interesting to extend our work to these settings.
- As already mentioned, existing formalisms for implementing higher-order rewriting via explicit substitutions, such as *XRS* and explicit *CRS*, are not able to cope with our augmented ς-calculus due to the presence of two distinct notions of substitution. Therefore, an issue which deserves further attention is how to extend these frameworks for higher-order rewriting in order to revert this situation.

Part III: From Higher-Order to First-Order Rewriting

Among the virtues of explicit substitutions is the fact that it allows higher-order calculi to be reduced to first-order ones, the λ-calculus being the prime witness of this fact. Part III inquires into the following fundamental issues:

1. What other calculi besides the λ-calculus are witnesses of this reduction to first-order?
2. What benefits result from it (hence justifying our calling it a virtue)?

As regards the first of these issues, we consider the general case of reducing higher-order rewrite systems to a first-order setting by presenting an encoding of the former into the latter. A distinctive advantage of our approach is that a well-established higher-order rewrite formalism is used as the departing formalism, namely a simplified variant of Khasidashvili's Expression Reduction Systems [Kha90]. Explicit Expression Reduction Systems is the first-order formalism defined as the destination formalism. A translation, called the Conversion Procedure, to first-order rewriting modulo an equational theory is considered, followed by a simple syntactic criterion to determine if a system may be reduced to a first-order rewrite system where the equational theory is empty (systems we have dubbed essentially first-order higher-order rewrite systems). Moreover, this translation commits to no particular calculus of explicit substitutions but rather relies on a generic macro-based substitution calculus encompassing many existing calculi, of the kind, in the literature. Also, it is argued that relating higher-order rewriting to first-order rewriting is not only appealing from an expressive-power point of view but also from the possibility of transferring results from the vast body of properties of the first-order framework to the higher-order one. This is the approach we take for shedding some light on the second issue. We argue that the class of essentially first-order $SERS_{db}$ is appropriate for such a task. Of course, this program is worthy of no serious consideration unless the class of essentially first-order systems includes 'interesting' systems, however the λ-calculus, among others, lives comfortably inside this class. The last chapter of Part III is devoted to the transfer of standardization. We list some interesting research directions:

- The Conversion Procedure amounts to incorporating, into the first-order notion of reduction, not only the computation of substitutions but also the higher-order (pattern) matching process. Indeed, so called pattern substitutions [DHKP98], arise naturally in the setting of the Conversion Procedure. It would be interesting to distinguish by means of different substitution operators and calculi, substitutions for matching and for 'usual' substitution in the rewrite system resulting from the Conversion Procedure. This would yield calculi of explicit substitutions and explicit matching.

- We have not considered preservation of strong normalization for the translation of higher-order rewrite systems. In the case of Explicit *CRS* some work has been done [BR96, Blo97]. In a first approach it seems convenient to fix some calculus of explicit substitutions. However, in order to maintain the parametricity achieved by using a general macro-based substitution calculus (basic substitution calculi) it would no doubt be of interest to identify additional conditions on basic substitution calculi which would guarantee PSN. This is related to the above mentioned research line on sufficiently general proof techniques for proving PSN.

- The transfer of other properties such as completion are left to future work.

- Chapter 8 raises the question of whether not only $\lambda\sigma$ but the whole class of essentially first-order $SERS_{db}$ enjoy finite normalization cones. From this one would be able to conclude that external derivations are normalizing in the explicit substitution counterpart of *any* orthogonal essentially first-order $SERS_{db}$.

- A precise comparison in the lines of [OR93] but between *SERS* and the *HRS* formalism would be interesting. Also one could compare $SERS^{*}$ and *HRS*. As already mentioned the metasubstitution operator may not occur on the *LHS* of a *SERS* rewrite rule. However, it seems that our results on simulation and projection can be extended to the case where they may occur on the *LHS* of a rewrite rule (the lemmata required are the same as those already developed in Part III). Let us denote this variant of *SERS* by $SERS^{*}$. Observe that $SERS^{*}$ has more 'matching power' than *SERS*. However (and in relation to the previous item), note that this would not be 'equivalent' to lifting the pattern condition on *HRS* since matching is computed by 'developments' in $SERS^{*}$.

- We have not considered preservation of strong normalization for the translation of higher-order rewrite systems. In the case of Explicit *CRS* some work has been done [BR96, Blo97]. In a first approach it seems convenient to fix some calculus of explicit substitutions. However, in order to maintain the parametricity achieved by using a general macro-based substitution calculus (basic substitution calculi) it would no doubt be of interest to identify additional conditions on basic substitution calculi which would guarantee PSN. This is related to the above mentioned research line on sufficiently general proof techniques for proving PSN.

- The transfer of other properties such as completion are left to future work.

- Chapter 8 raises the question of whether not only $\lambda\sigma$ but the whole class of essentially first-order $SERS_{db}$ enjoy finite normalization cones. From this one would be able to conclude that external derivations are normalizing in the explicit substitution counterpart of *any* orthogonal essentially first-order $SERS_{db}$.

- A precise comparison in the lines of [OR93] but between *SERS* and the *HRS* formalism would be interesting. Also one could compare $SERS^*$ and *HRS*. As already mentioned the metasubstitution operator may not occur on the *LHS* of a *SERS* rewrite rule. However, it seems that our results on simulation and projection can be extended to the case where they may occur on the *LHS* of a rewrite rule (the lemmata required are the same as those already developed in Part III). Let us denote this variant of *SERS* by $SERS^*$. Observe that $SERS^*$ has more 'matching power' than *SERS*. However (and in relation to the previous item), note that this would not be 'equivalent' to lifting the pattern condition on *HRS* since matching is computed by 'developments' in $SERS^*$.

Appendix A

Appendix

A.1 Perpetuality in Calculi of Explicit Substitutions

Definition A.1 (RPO) Let $s = f(s_1, .., s_m)$ and $t = g(t_1, .., t_n)$ be terms in $\mathcal{T}_l$. Then $s \succ_{\mathcal{T}_l} t$ if and only if one of the following holds:

1. (subterm) $s_i \succ_{\mathcal{T}_l} t$ or $s_i = t$ for some $i \in 1..m$
2. (decreasing heads) $f \gg g$ and $s \succ_{\mathcal{T}_l} t_1, .., s \succ_{\mathcal{T}_l} t_n$
3. (equal heads) $f = g$ and $\langle s_1, .., s_m \rangle \succ'_{\mathcal{T}_l} \langle t_1, .., t_n \rangle$

where $\succ'_{\mathcal{T}_l}$ is the extension of $\succ_{\mathcal{T}_l}$ to multisets.

Lemma A.2 (Subject reduction) Let $\Gamma \rhd M : \sigma$ be a derivable type judgement and suppose $M \rightarrow_{F_{es}} N$. Then $\Gamma \rhd N : \sigma$ is a derivable type judgement.

Proof. By induction on the reduction $M \rightarrow_r N$ with $r \in F_{es}$. Thus we consider the cases when the reduction takes place at the root and when the reduction is internal.

Suppose the reduction takes place at the root. Then we have the following cases to analyse:

- $r = Beta2$. Then $M = (\lambda x : \tau.P)Q$ and $N = P\langle x := Q\rangle$ and the derivation runs

$$\frac{\dfrac{\Gamma, x : \tau \rhd P : \sigma}{\Gamma \rhd \lambda x : \tau.P : ZT(\tau) \rightarrow \sigma}\,abs \qquad \Gamma \rhd Q : ZT(\tau)}{\Gamma \rhd (\lambda x : \tau.P)Q : \sigma}\,app$$

  and we obtain

$$\frac{\Gamma, x : \tau \rhd P : \sigma \quad \Gamma \rhd Q : ZT(\tau)}{\Gamma \rhd P\langle x := Q\rangle : \sigma}\,subs$$

- $r = \tau Beta$. Then $M = (\Lambda t.P)\tau$ and $N = P[t := \tau]$ and the derivation runs

$$\frac{\dfrac{\Gamma \rhd P : \sigma'}{\Gamma \rhd \Lambda t.P : \forall t.\sigma'}\,tabs}{\Gamma \rhd (\Lambda t.P)\tau : \sigma'\{t \leftarrow ZT(\tau)\}}\,tapp$$

  where $t \notin FTV(\Gamma)$. We may obtain

$$\frac{\Gamma \rhd P : \sigma'}{\Gamma \rhd P[t := \tau] : \sigma'\{t \leftarrow ZT(\tau)\}}\,tsubs$$

Note that $\Gamma \in \{\ \{x_1 : \sigma'_1, \ldots, x_n : \sigma'_n\} \mid \sigma'_i =_{ZT} \sigma_i[t := \tau]\ \mathit{for\ all}\ i \in 1..n\}$ where $\Gamma = \{\sigma_1, \ldots, \sigma_n\}$, since $t \notin FTV(\Gamma)$ and we may use the rule *ztgc*.

- $r = \mathit{zapp}$. Then $M = (PQ)\langle x := R\rangle$ and $R = P\langle x := R\rangle Q\langle x := R\rangle$ and the derivation runs as follows

$$\dfrac{\dfrac{\Gamma, x : \tau \rhd P : \sigma_1 \to \sigma \quad \Gamma, x : \tau \rhd Q : \sigma_1}{\Gamma, x : \tau \rhd PQ : \sigma}\,\mathit{app} \qquad \Gamma \rhd R : ZT(\tau)}{\Gamma \rhd (PQ)\langle x := R\rangle : \sigma}\,\mathit{subs}$$

and we may obtain

$$\dfrac{\dfrac{\Gamma, x : \tau \rhd P : \sigma_1 \to \sigma \quad \Gamma \rhd R : ZT(\tau)}{\Gamma \rhd P\langle x := R\rangle : \sigma_1 \to \sigma}\,\mathit{subs} \qquad \dfrac{\Gamma, x : \tau \rhd Q : \sigma_1 \quad \Gamma \rhd R : ZT(\tau)}{\Gamma \rhd Q\langle x := R\rangle : \sigma_1}\,\mathit{subs}}{\Gamma \rhd P\langle x := R\rangle Q\langle x := R\rangle : \sigma}\,\mathit{app}$$

- $r = \mathit{zlam}$. Then $M = (\lambda y : \sigma_1.(P))\langle x := Q\rangle$ and $N = \lambda y : \sigma_1.(P\langle x := Q\rangle)$. The derivation runs

$$\dfrac{\dfrac{\Gamma, x : \tau, y : \sigma_1 \rhd P : \sigma_2}{\Gamma, x : \tau \rhd \lambda y : \sigma_1.(P : ZT(\sigma_1) \to \sigma_2)}\,\mathit{abs} \qquad \Gamma \rhd Q : ZT(\tau)}{\Gamma \rhd (\lambda y : \sigma_1.(P))\langle x := Q\rangle : ZT(\sigma_1) \to \sigma_2}\,\mathit{subs}$$

Now by the variable convention we may assume that all bound type variables in Q do not occur free in σ_1 and that y does not occur bound in Q. Then by Lemma 3.43 we have that $\Gamma, y : \sigma_1 \rhd Q : \tau$ is a derivable type judgement. This allows us to construct the following derivation.

$$\dfrac{\dfrac{\Gamma, x : \tau, y : \sigma_1 \rhd P : \sigma_2 \quad \Gamma, y : \sigma_1 \rhd Q : ZT(\tau)}{\Gamma, y : \sigma_1 \rhd P\langle x := Q\rangle : \sigma_2}\,\mathit{subs}}{\Gamma \rhd \lambda y : \sigma_1.(P\langle x := Q\rangle) : ZT(\sigma_1) \to \sigma_2}\,\mathit{abs}$$

- $r = \mathit{zvar}$. Then $M = x\langle x := P\rangle$, $N = P$, $\sigma = ZT(\sigma')$ and the derivation runs

$$\dfrac{\Gamma, x : \sigma' \rhd x : ZT(\sigma') \quad \Gamma \rhd P : ZT(\sigma')}{\Gamma \rhd x\langle x := P\rangle : ZT(\sigma')}\,\mathit{subs}$$

And the subderivation ending in the $\Gamma \rhd P : ZT(\sigma')$ suffices.

- $r = \mathit{zgc}$. Then $M = Q\langle x := P\rangle$ and $N = Q$ with $x \notin FV(Q)$, and the derivation runs

$$\dfrac{\Gamma, x : \tau \rhd Q : ZT(\sigma') \quad \Gamma \rhd P : ZT(\tau)}{\Gamma \rhd Q\langle x := P\rangle : ZT(\sigma')}\,\mathit{subs}$$

But one may verify by induction that if $\Gamma, x : \tau \rhd Q : \sigma$ is a derivable type judgement such that $x \notin FV(Q)$, then $\Gamma \rhd Q : \sigma$ is also a derivable type judgement. This concludes de case.

- $r = \mathit{zappt}$. Then $M = (P\tau')\langle x := Q\rangle$ and $N = P\langle x := Q\rangle\tau'$ and $\sigma = \sigma'\{t \leftarrow ZT(\tau')\}$ where

$$\dfrac{\dfrac{\Gamma, x : \tau \rhd P : \forall t.\sigma'}{\Gamma, x : \tau \rhd P\tau' : \sigma'\{t \leftarrow ZT(\tau')\}}\,\mathit{tapp} \qquad \Gamma \rhd Q : ZT(\tau)}{\Gamma \rhd (P\tau')\langle x := Q\rangle : \sigma'\{t \leftarrow ZT(\tau')\}}\,\mathit{subs}$$

Then we may commute the application of rules *tapp* and *subs* as follows

$$\dfrac{\dfrac{\Gamma, x : \tau \rhd P : \forall t.\sigma' \quad \Gamma \rhd Q : ZT(\tau)}{\Gamma \rhd P\langle x := Q\rangle : \forall t.\sigma'}\,\mathit{subs}}{\Gamma \rhd P\langle x := Q\rangle\tau' : \sigma'\{t \leftarrow ZT(\tau')\}}\,\mathit{tapp}$$

- $r = \mathit{zlamt}$. Then $M = (\Lambda t.P)\langle x := Q\rangle$ and $N = \Lambda t.P\langle x := Q\rangle$ and the derivation runs

$$\dfrac{\dfrac{\Gamma, x : \tau \rhd P : \rho}{\Gamma, x : \tau \rhd \Lambda t.P : \forall t.\rho}\,\mathit{tabs} \qquad \Gamma \rhd Q : ZT(\tau)}{\Gamma \rhd (\Lambda t.P)\langle x := Q\rangle : \forall t.\rho}\,\mathit{subs}$$

  where $t \notin FTV(\Gamma, x : \tau)$. And we may obtain

$$\dfrac{\dfrac{\Gamma, x : \tau \rhd P : \rho \quad \Gamma \rhd Q : ZT(\tau)}{\Gamma \rhd P\langle x := Q\rangle : \rho}\,\mathit{subs}}{\Gamma \rhd \Lambda t.P\langle x := Q\rangle : \forall t.\rho}\,\mathit{tabs}$$

- $r = \mathit{ztapp}$. Then $M = (PQ)[t := \tau]$ and $N = P[t := \tau]Q[t := \tau]$ and the derivation runs

$$\dfrac{\dfrac{\Delta \rhd P : \sigma_1 \to \sigma_2 \quad \Delta \rhd Q : \sigma_1}{\Delta \rhd PQ : \sigma_2}\,\mathit{app}}{\Delta_{[t:=\tau]} \rhd (PQ)[t := \tau] : \sigma_2\{t \leftarrow ZT(\tau)\}}\,\mathit{tsubs}$$

  And we may obtain

$$\dfrac{\dfrac{\Delta \rhd P : \sigma_1 \to \sigma_2}{\Delta_{[t:=\tau]} \rhd P[t := \tau] : (\sigma_1 \to \sigma_2)\{t \leftarrow ZT(\tau)\}}\,\mathit{tsubs} \qquad \dfrac{\Delta \rhd Q : \sigma_1}{\Delta_{[t:=\tau]} \rhd Q[t := \tau] : \sigma_1\{t \leftarrow ZT(\tau)\}}\,\mathit{tsubs}}{\Delta_{[t:=\tau]} \rhd P[t := \tau]Q[t := \tau] : \sigma_2\{t \leftarrow ZT(\tau)\}}\,\mathit{app}$$

- $r = \mathit{ztlam}$. Then $M = (\lambda y : \sigma'.(P))[t := \tau]$ and $N = \lambda y : \sigma'[t := \tau].(P[t := \tau])$ and the derivation runs

$$\dfrac{\dfrac{\Delta, y : \sigma' \rhd P : \rho}{\Delta \rhd \lambda y : \sigma'.P : ZT(\sigma') \to \rho}\,\mathit{abs}}{\Delta_{[t:=\tau]} \rhd (\lambda y : \sigma'.P)[t := \tau] : (ZT(\sigma') \to \rho)\{t \leftarrow ZT(\tau)\}}\,\mathit{tsubs}$$

  and we may obtain

$$\dfrac{\dfrac{\Delta, y : \sigma' \rhd P : \rho}{\Delta_{[t:=\tau]}, y : \sigma'[t := \tau] \rhd P[t := \tau] : \rho\{t \leftarrow ZT(\tau)\}}\,\mathit{tsubs}}{\Delta_{[t:=\tau]} \rhd \lambda y : \sigma'[t := \tau].P[t := \tau] : ZT(\sigma'[t := \tau]) \to \rho\{t \leftarrow ZT(\tau)\}}\,\mathit{tsubs}$$

  Then by Lemma 3.29(2) we are done.

- $r = \mathit{ztvar2}$. Then $M = x[t := \tau]$ and $N = x$ and the derivation runs

$$\dfrac{\Delta \rhd x : ZT(\rho)}{\Delta_{[t:=\tau]} \rhd x[t := \tau] : ZT(\rho)\{t \leftarrow ZT(\tau)\}}\,\mathit{tsubs}$$

  where $x : \rho \in \Delta$. Let $x : \rho' \in \Delta_{[t:=\tau]}$ and therefore $\rho' =_{ZT} \rho[t := \tau]$. Then $ZT(\rho') = ZT(\rho)\{t \leftarrow ZT(\tau)\}$ using Lemma 3.29(2) and the fact that ZT is complete. Then $\Delta_{[t:=\tau]} \rhd x : ZT(\rho') = ZT(\rho)\{t \leftarrow ZT(\tau)\}$ is a derivable type judgement (using *var*).

  Note that if the *ztvar2*-rule were replaced by the *ztgc2*-rule then the following result, which may be proved by induction on the size of the derivation, would be required: if $\Delta \rhd M : \rho$ is derivable and $t \notin FTV(M)$ then $\Delta_{[t:=\tau]} \rhd M : \rho\{t \leftarrow ZT(\tau)\}$ is derivable for any type τ.

- $r = \mathit{ztappt}$. Then $M = (P\tau)[t := \tau']$ and $N = P[t := \tau']\tau[t := \tau']$ and the derivation runs

$$\cfrac{\cfrac{\Delta \rhd P : \forall u.\rho}{\Delta \rhd P\tau : \rho\{u \leftarrow ZT(\tau)\}}\, tapp}{\Delta_{[t:=\tau']} \rhd (P\tau)[t := \tau'] : \rho\{u \leftarrow ZT(\tau)\}\{t \leftarrow ZT(\tau')\}}\, tsubs$$

and we may obtain

$$\cfrac{\cfrac{\Delta \rhd P : \forall u.\rho}{\Delta_{[t:=\tau']} \rhd P[t := \tau'] : (\forall u.\rho)\{t \leftarrow ZT(\tau')\} = \forall u.\rho\{\tau \leftarrow ZT(\tau')\}}\, tsubs}{\Delta_{[t:=\tau']} \rhd P[t := \tau']\tau[t := \tau'] : \rho\{t \leftarrow ZT(\tau')\}\{u \leftarrow ZT(\tau[t := \tau'])\}}\, tapp$$

Then using Lemma 3.29(1) and (2) we are done.

- $r = ztlamt$. Then $M = (\Lambda u.P)[t := \tau]$ and $N = \Lambda u.P[t := \tau]$ and we assume by the variable convention that $u \notin FTV(\tau)$. Then the derivation runs

$$\cfrac{\cfrac{\Delta \rhd P : \rho}{\Delta \rhd \Lambda u.P : \forall u.\rho}\, tabs}{\Delta_{[t:=\tau]} \rhd (\Lambda u.P)[t := \tau] : (\forall u.\rho)\{t \leftarrow ZT(\tau)\}}\, tsubs$$

where $u \notin FTV(\Delta)$. We may assume by the variable convention that $u \notin FTV(\Delta_{[t:=\tau]})$ and thus obtain the derivation

$$\cfrac{\cfrac{\Delta \rhd P : \rho}{\Delta_{[t:=\tau]} \rhd P[t := \tau] : \rho\{t \leftarrow ZT(\tau)\}}\, tsubs}{\Delta_{[t:=\tau]} \rhd \Lambda u.P[t := \tau] : \forall u.\rho\{t \leftarrow ZT(\tau)\}}\, tabs$$

Suppose now that the reduction is internal. Then we consider the following cases according to each possible context C

- $C = \lambda x : \tau.P$. We must consider two possible cases:
  - $P \rightarrow_{F_{es}} P'$ and therefore $N = \lambda x : \tau.P'$. This case is handled by the induction hypothesis.
  - $\tau \rightarrow_{ZT} \tau'$ and therefore $N = \lambda x : \tau'.P$. Then the derivation runs

  $$\cfrac{\Gamma, x : \tau \rhd P : \rho}{\Gamma \rhd \lambda x : \tau.P : ZT(\tau) \rightarrow \rho}\, abs$$

  And we may conclude by using the context reduction Lemma 3.44 on the derivation ending in the premise of *abs* followed by a new application of *abs*, and noting that $ZT(\tau) = ZT(\tau')$.

- $C = \Lambda t.P$. Then $P \rightarrow_{F_{es}} P'$ and $N = \Lambda t.P'$ and we use the induction hypothesis.
- $C = PQ$. Then either $P \rightarrow_{F_{es}} P'$ and therefore $N = P'Q$, or $Q \rightarrow_{F_{es}} Q'$ and therefore $N = PQ'$. In both cases we use the induction hypothesis.
- $C = P\rho$. Then either $P \rightarrow_{F_{es}} P'$ and therefore $N = P'\rho$, or $\rho \rightarrow_{ZT} \rho'$ and therefore $N = P\rho'$. For the first case we use the induction hypothesis. The second case is resolved by a new application of *tapp* and noting that $ZT(\rho) = ZT(\rho')$.
- $C = P\langle x := Q\rangle$. Then either $P \rightarrow_{F_{es}} P'$ and therefore $N = P'\langle x := Q\rangle$, or $Q \rightarrow_{F_{es}} Q'$ and therefore $N = P\langle x := Q'\rangle$. In both cases we use the induction hypothesis.
- $C = P[t := \tau]$. Then the derivation runs

$$\cfrac{\Delta \rhd P : \rho}{\Delta_{[t:=\tau]} \rhd P[t := \tau] : \rho\{t \leftarrow ZT(\tau)\}}\, tsubs$$

And either $P \rightarrow_{F_{es}} P'$ and therefore $N = P'[t := \tau]$, or $\tau \rightarrow_{ZT} \tau'$ and therefore $N = P[t := \tau']$. For the first case we use the induction hypothesis. For the second we construct the following derivation.

$$\frac{\Delta \rhd P : \rho}{\Delta_{[t:=\tau]} \rhd P[t := \tau'] : \rho\{t \leftarrow ZT(\tau')\}}\ tsubs$$

We are left to verify that the assignment $\Delta_{[t:=\tau]}$ is in the set of type assignments $\{\ \{x_1 : \sigma'_1, \ldots, x_n : \sigma'_n\} \mid \sigma'_i =_{ZT} \sigma_i[t := \tau']\ \mathit{for\ all}\ i \in 1..n\} = \Delta_{[t:=\tau']}$, where $\Delta = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$, which is true since $\tau \rightarrow_{ZT} \tau'$.

A.2 Explicit Substitutions for a Calculus of Objects

A.2.1 Confluence of ς^f_{db}

We prove confluence of ς^f_{db} by adapting a proof technique presented in [Tak89], a variation of the Tait-and-Martin Löf technique. For this, a notion of parallel rewriting is required, analogous to that defined in [Tak89] for the λ-calculus but in the ς-calculus. But before that, some basic properties of ς^f_{db}-rewriting and preservation of ς^f_{db}-rewriting by updating and substitution, must be considered

As regards the behaviour of substitutions with respect to substitution, the updating functions and reduction we have the following results (similar properties in the framework of calculi of explicit substitution appeared in [KR95]):

Lemma A.3 1. Let $a, b \in \mathcal{T}_{\varsigma^f_{db}}$. Then $\forall i, j, k$ such that $i > 0, j \geq 0$ and $j < i \leq j + k$ we have $\mathcal{U}^{k+1}_j(a)\{\!\{i \leftarrow b\}\!\} = \mathcal{U}^k_j(a)$.

2. Let $a, b \in \mathcal{T}_{\varsigma^f_{db}}$ and $i \leq n - k$. Then $\mathcal{U}^i_k(a)\{\!\{n \leftarrow b\}\!\} = \mathcal{U}^i_k(a\{\!\{n - i + 1 \leftarrow b\}\!\})$.

3. Let $a \in \mathcal{T}_{\varsigma^f_{db}}$ and $p \leq k < p + j$. Then $\mathcal{U}^i_k(\mathcal{U}^j_p(a)) = \mathcal{U}^{j+i-1}_p(a)$.

4. Let $a, b \in \mathcal{T}_{\varsigma^f_{db}}$ and $n \leq k + 1$. Then $\mathcal{U}^j_k(a\{\!\{n \leftarrow b\}\!\}) = \mathcal{U}^j_{k+1}(a)\{\!\{n \leftarrow \mathcal{U}^j_{k-n+1}(b)\}\!\}$.

5. Let $a \in \mathcal{T}_{\varsigma^f_{db}}$ and $n \leq l + 1$. Then $\mathcal{U}^j_{l+p}(\mathcal{U}^n_p(a)) = \mathcal{U}^n_p(\mathcal{U}^j_{l+p+1-n}(a))$.

6. Let $a, c \in \mathcal{T}_{\varsigma^f_{db}}$. Then $\mathcal{U}^2_k(a)\{\!\{k + 1 \leftarrow c\}\!\} = a$.

Proof. All but the last item are proved by induction on a. The last item may be proved by using item (1) and the fact that for any $j \geq 0$ we have $\mathcal{U}^1_j(a) = a$.

•

The Substitution Lemma also holds for the ς^f_{db}-calculus. For the sake of completeness we have included its proof in full detail.

Lemma A.4 (Substitution Lemma) Let $a, b, c \in \mathcal{T}_{\varsigma^f_{db}}$. Then $\forall n > 0, \forall i,\ 1 \leq i \leq n, a\{\!\{i \leftarrow b\}\!\}\{\!\{n \leftarrow c\}\!\} = a\{\!\{n + 1 \leftarrow c\}\!\}\{\!\{i \leftarrow b\{\!\{n - i + 1 \leftarrow c\}\!\}\}\!\}$.

Proof. By induction on a using Lemma A.3, items (1) and (2). •

The reduction relation is preserved by substitution and the updating functions. Since the proofs of these results resemble those of the corresponding results in the ς_{db}-calculus we have omitted them.

Lemma A.5 Let $a, a', b \in \mathcal{T}_{\varsigma^f_{db}}$. Then if $a \rightarrow_{\varsigma^f_{db}} a'$ then

1. $a\{\!\{n \leftarrow b\}\!\} \rightarrow_{\varsigma^f_{db}} a'\{\!\{n \leftarrow b\}\!\}$

2. $b\{\!\{n \leftarrow a\}\!\} \twoheadrightarrow_{\varsigma^f_{db}} b\{\!\{n \leftarrow a'\}\!\}$

3. $\mathcal{U}^i_k(a) \rightarrow_{\varsigma^f_{db}} \mathcal{U}^i_k(a')$.

In fact the following, more precise, variant of the preceding lemma shall be used later. Its proof relies on Lemma A.5.

Corollary A.6 Let $a, a' \in \mathcal{T}_{\varsigma^f_{db}}$ and $k \geq 0$. Then if $a \xrightarrow{k}_{\varsigma^f_{db}} a'$ then

1. $a\{\!\{n \leftarrow b\}\!\} \xrightarrow{k}_{\varsigma^f_{db}} a'\{\!\{n \leftarrow b\}\!\}$

2. $b\{\!\{n \leftarrow a\}\!\} \twoheadrightarrow_{\varsigma^f_{db}} b\{\!\{n \leftarrow a'\}\!\}$

We now consider its confluence by adapting a proof technique presented in [Tak89], a variation of the Tait-and-Martin Löf technique. For this, a notion of parallel rewriting is required, analogous to that defined in [Tak89] for the λ-calculus but in the ς-calculus. Figure A.1 provides such a definition. Also, the results developed in the previous subsection shall be used.

$$\frac{}{n \Rightarrow n}\,Ind \qquad \frac{a \Rightarrow a'}{a.l \Rightarrow a'.l}\,I \qquad \frac{a \Rightarrow a'}{l := a \Rightarrow l := a'}\,Am \qquad \frac{g \Rightarrow g'}{l \doteq g \Rightarrow l \doteq g'}\,Mm$$

$$\frac{a \Rightarrow a'}{\varsigma(a) \Rightarrow \varsigma(a')}\,M \qquad \frac{m_i \Rightarrow m'_i \; i \in 1..n}{[m_i{}^{i\in 1..n}] \Rightarrow [m'_i{}^{i\in 1..n}]}\,Ob \qquad \frac{a \Rightarrow a' \quad m \Rightarrow m'}{a \lhd < m > \Rightarrow a' \lhd < m' >}\,Ov$$

$$\frac{b \Rightarrow b' \quad m_i \Rightarrow m'_i \; i \in 1..n, i \neq j}{[l_j \doteq \varsigma(b), m_i{}^{i\in 1..n, i\neq j}].l_j \Rightarrow b'\{\!\{1 \leftarrow [l_j \doteq \varsigma(b'), m'_i{}^{i\in 1..n, i\neq j}]\}\!\}}\,Im$$

$$\frac{a \Rightarrow a'}{[l_j := a, m_i^{i\in 1..n, i\neq j}].l_j \Rightarrow a'}\,Ia \qquad \frac{b \Rightarrow b' \quad m_i \Rightarrow m'_i \; i \in 1..n, i \neq j}{[m_i{}^{i\in 1..n}] \lhd < l_j \doteq \varsigma(b) > \Rightarrow [l_j \doteq \varsigma(b'), m'_i{}^{i\in 1..n}]}\,Om$$

$$\frac{b \Rightarrow b' \quad m_i \Rightarrow m'_i \; i \in 1..n, i \neq j}{[m_i{}^{i\in 1..n}] \lhd < l_j := b > \Rightarrow [l_j := b', m'_i{}^{i\in 1..n}]}\,Oa$$

Figure A.1: Parallel reduction in the ς^f_{db}-calculus

Definition A.7 (Maximal Complete Development) Let $a \in \mathcal{T}_{\varsigma^f_{db}}$ then we define a^* (the Maximal Complete Development of a) inductively as follows:

$$\begin{array}{lcl}
n^* & \stackrel{\text{def}}{=} & n \\
([l_j \doteq \varsigma(b), m_i{}^{i\in 1..n, i\neq j}].l_j)^* & \stackrel{\text{def}}{=} & b^*\{\!\{1 \leftarrow [l_j \doteq \varsigma(b^*), m^*_i \;\; {}^{i\in 1..n, i\neq j}]\}\!\} \\
([l_j := b, m_i{}^{i\in 1..n, i\neq j}].l_j)^* & \stackrel{\text{def}}{=} & b^* \\
(a.l)^* & \stackrel{\text{def}}{=} & a^*.l \;\; if\ a.l\ is\ not\ a\ redex \\
\varsigma(b)^* & \stackrel{\text{def}}{=} & \varsigma(b^*) \\
(l \doteq g)^* & \stackrel{\text{def}}{=} & l \doteq g^* \\
(l := b)^* & \stackrel{\text{def}}{=} & l := b^* \\
([m_i{}^{i\in 1..n}] \lhd < l_j := c >)^* & \stackrel{\text{def}}{=} & [l_j := c^*,\; m^*_i \;\; {}^{i\in 1..n, i\neq j}] \\
([m_i{}^{i\in 1..n}] \lhd < l_j \doteq \varsigma(b) >)^* & \stackrel{\text{def}}{=} & [l_j \doteq \varsigma(b^*),\; m^*_i \;\; {}^{i\in 1..n, i\neq j}] \\
(a \lhd < m >)^* & \stackrel{\text{def}}{=} & a^* \lhd < m^* > \;\; if\ a \lhd < m > \;\; is\ not\ a\ redex. \\
[m_i{}^{i\in 1..n}]^* & \stackrel{\text{def}}{=} & [m^*_i \;\; {}^{i\in 1..n}]
\end{array}$$

The result below considers the behaviour of the parallel reduction relation respect to the updating functions and substitution.

Lemma A.8 Let $a, b \in \mathcal{T}_{\varsigma'_{db}}$. If $a \Rightarrow b$ then $\forall k, n \geq 0$ we have $\mathcal{U}^n_k(a) \Rightarrow \mathcal{U}^n_k(b)$.

Proof. By induction on a.

- $a = p$. Then $b = p$ and we may use rule (Ind).
- $a = c.l_j$. Then there are three subcases we must consider:
  - $b = c'.l_j$ and $c \Rightarrow c'$. Then by the induction hypothesis we have $\mathcal{U}^n_k(c) \Rightarrow \mathcal{U}^n_k(c')$. Then using rule (I) we obtain $\mathcal{U}^n_k(c).l_j \Rightarrow \mathcal{U}^n_k(c').l_j$.
  - $c = [l_j \doteq \varsigma(d), m_i{}^{i \in 1..n, i \neq j}]$ and $b = d'\{\!\{1 \leftarrow [l_j \doteq \varsigma(d'), m'_i{}^{i \in 1..n, i \neq j}]\}\!\}$, where $d \Rightarrow d'$ and $m_i \Rightarrow m'_i$. Then by induction hypothesis we have $\mathcal{U}^n_{k+1}(d) \Rightarrow \mathcal{U}^n_{k+1}(d')$ and $\mathcal{U}^n_k(m_i) \Rightarrow \mathcal{U}^n_k(m'_i)$, and therefore,

$$\frac{\mathcal{U}^n_{k+1}(d) \Rightarrow \mathcal{U}^n_{k+1}(d') \quad \mathcal{U}^n_k(m_i) \Rightarrow \mathcal{U}^n_k(m'_i)}{[l_j \doteq \varsigma(\mathcal{U}^n_{k+1}(d)), \mathcal{U}^n_k(m_i)\ {}^{i \in 1..n, i \neq j}].l_j \Rightarrow \mathcal{U}^n_{k+1}(d')\{\!\{1 \leftarrow [l_j \doteq \varsigma(\mathcal{U}^n_{k+1}(d')), \mathcal{U}^n_k(m'_i)\ {}^{i \in 1..n, i \neq j}]\}\!\}}\ Im$$

    And by Lemma A.3(4) $(n = 1)$ we have that $\mathcal{U}^n_{k+1}(d')\{\!\{1 \leftarrow [l_j \doteq \varsigma(\mathcal{U}^n_{k+1}(d')), \mathcal{U}^n_k(m'_i)\ {}^{i \in 1..n, i \neq j}]\}\!\} = \mathcal{U}^n_k(d'\{\!\{1 \leftarrow [l_j \doteq \varsigma(d'), m'_i{}^{i \in 1..n, i \neq j}]\}\!\})$.
  - $c = [l_j := d, m_i{}^{i \in 1..n, i \neq j}]$ and $b = d'$, where $d \Rightarrow d'$. Then by the induction hypothesis we have $\mathcal{U}^n_k(d) \Rightarrow \mathcal{U}^n_k(d')$, and therefore by rule (Ia), we have $[l_j := \mathcal{U}^n_k(d), \mathcal{U}^n_k(m_i)\ {}^{i \in 1..n, i \neq j}].l_j \Rightarrow \mathcal{U}^n_k(d')$
- $a = c \lhd < m >$. Then we consider three cases, one for each of the possible rules that could have been applied (i.e. $(Ov), (Om), (Oa)$). They are dealt with as above.

The remaining cases are similar and are dealt with as above.

▪

Lemma A.9 Let $d, d', e, e' \in \mathcal{T}_{\varsigma'_{db}}$ and $n \geq 1$. If $d \Rightarrow d'$ and $e \Rightarrow e'$ then we have $d\{\!\{n \leftarrow e\}\!\} \Rightarrow d'\{\!\{n \leftarrow e'\}\!\}$.

Proof. We use induction on d.

- $d = p$. Then $d' = p$ and we have

$$p\{\!\{n \leftarrow e\}\!\} \quad = \quad \begin{cases} p - 1 = p\{\!\{n \leftarrow e'\}\!\} & if\ p > n \\ \mathcal{U}^n_0(e) \Rightarrow \mathcal{U}^n_0(e') = p\{\!\{n \leftarrow e'\}\!\} & if\ p = n \\ p = p\{\!\{n \leftarrow e'\}\!\} & if\ p < n \end{cases}$$

  Case $p = n$ is justified by using Lemma A.8.
- $d = c.l_j$. Then we must consider three subcases:
  - $d' = c'.l_j$ with $c \Rightarrow c'$. Then $c.l\{\!\{n \leftarrow e\}\!\} = c\{\!\{n \leftarrow e\}\!\}.l \Rightarrow c'\{\!\{n \leftarrow e'\}\!\}.l = c'.l\{\!\{n \leftarrow e'\}\!\}$ by the induction hypothesis.
  - $c = [l_j \doteq \varsigma(b), m_i{}^{i \in 1..n, i \neq j}]$ and $d' = b'\{\!\{1 \leftarrow [l_j \doteq \varsigma(b'), m'_i{}^{i \in 1..n, i \neq j}]\}\!\}$ with $b \Rightarrow b'$ and $m_i \Rightarrow m'_i$. Now $c.l_j\{\!\{n \leftarrow e\}\!\} = [l_j \doteq \varsigma(b\{\!\{n + 1 \leftarrow e\}\!\}), m_i\{\!\{n \leftarrow e\}\!\}\ {}^{i \in 1..n, i \neq j}].l_j$.
    By induction hypothesis $b\{\!\{n + 1 \leftarrow e\}\!\} \Rightarrow b'\{\!\{n + 1 \leftarrow e'\}\!\}$ and $m_i\{\!\{n \leftarrow e\}\!\} \Rightarrow m'_i\{\!\{n \leftarrow e'\}\!\}$, therefore by applying rule (Im) we obtain

$$\begin{array}{lll} c.l_j\{\!\{n \leftarrow e\}\!\} & = & [l_j \doteq \varsigma(b\{\!\{n + 1 \leftarrow e\}\!\}), m_i\{\!\{n \leftarrow e\}\!\}\ {}^{i \in 1..n, i \neq j}].l_j \\ & \Rightarrow & b'\{\!\{n + 1 \leftarrow e'\}\!\}\{\!\{1 \leftarrow [l_j \doteq \varsigma(b'\{\!\{n + 1 \leftarrow e'\}\!\}), m'_i\{\!\{n \leftarrow e'\}\!\}\ {}^{i \in 1..n, i \neq j}]\}\!\} \\ & = & b'\{\!\{n + 1 \leftarrow e'\}\!\}\{\!\{1 \leftarrow [l_j \doteq \varsigma(b'), m'_i{}^{i \in 1..n, i \neq j}]\{\!\{n \leftarrow e'\}\!\}\}\!\} \\ & =_{L\ A.4} & b'\{\!\{1 \leftarrow [l_j \doteq \varsigma(b'), m'_i{}^{i \in 1..n, i \neq j}]\}\!\}\{\!\{n \leftarrow e'\}\!\} \\ & = & d'\{\!\{n \leftarrow e'\}\!\} \end{array}$$

  - $c = [l_j := b, m_i{}^{i \in 1..n, i \neq j}]$ and $d' = b'$ with $b \Rightarrow b'$. Now by the induction hypothesis we have that $b\{\!\{n \leftarrow e\}\!\} \Rightarrow b'\{\!\{n \leftarrow e'\}\!\}$ thus resulting in

$$\frac{b\{\!\{n \leftarrow e\}\!\} \Rightarrow b'\{\!\{n \leftarrow e'\}\!\}}{[l_j := b\{\!\{n \leftarrow e\}\!\}, m_i{}^{i \in 1..n, i \neq j}].l_j \Rightarrow b'\{\!\{n \leftarrow e'\}\!\}}\ Ia$$

- $d = \varsigma(a)$. Then from the definition of parallel reduction it must be that $d' = \varsigma(a')$ with $a \Rightarrow a'$. By induction hypothesis $a\{\!\{n+1 \leftarrow e\}\!\} \Rightarrow a'\{\!\{n+1 \leftarrow e'\}\!\}$ thus resulting in

$$\frac{a\{\!\{n+1 \leftarrow e\}\!\} \Rightarrow a'\{\!\{n+1 \leftarrow e'\}\!\}}{\varsigma(a\{\!\{n+1 \leftarrow e\}\!\}) \Rightarrow \varsigma(a'\{\!\{n+1 \leftarrow e'\}\!\})}\ M$$

  and by the definition of substitution we are done.

- $d = (l \doteq g)$. Then by studying the inference rules defining the parallel reduction relation we conclude that $d' = (l \doteq g')$ with $g \Rightarrow g'$. We resolve using the induction hypothesis in a similar fashion to the previous case.

- $d = (l := a)$. Then by studying the inference rules defining the parallel reduction relation we conclude that $d' = (l := a')$ with $a \Rightarrow a'$. We resolve using the induction hypothesis in a similar fashion to the previous case.

- $d = a \lhd < m >$. There are three subcases we must consider:

  - $d' = a' \lhd < m' >$ with $a \Rightarrow a'$ and $m \Rightarrow m'$. Then we resolve using the induction hypothesis.
  - $d = [m_i{}^{i \in 1..n}] \lhd < l_j \doteq \varsigma(b) >$. Then $d' = [l_j \doteq \varsigma(b'), m_i'{}^{i \in 1..n, i \neq j}]$ where $b \Rightarrow b'$ and $m_i \Rightarrow m_i'$.
    Now by induction hypothesis $b\{\!\{n+1 \leftarrow e\}\!\} \Rightarrow b'\{\!\{n+1 \leftarrow e'\}\!\}$ and $m_i\{\!\{n \leftarrow e\}\!\} \Rightarrow m_i'\{\!\{n \leftarrow e'\}\!\}$ allowing us to conclude using the *Om*-rule:

$$\frac{b\{\!\{n+1 \leftarrow e\}\!\} \Rightarrow b'\{\!\{n+1 \leftarrow e'\}\!\} \quad m_i\{\!\{n \leftarrow e\}\!\} \Rightarrow m_i'\{\!\{n \leftarrow e'\}\!\}}{[m_i\{\!\{n \leftarrow e\}\!\}{}^{i \in 1..n}] \lhd < l_j \doteq \varsigma(b\{\!\{n+1 \leftarrow e\}\!\}) > \Rightarrow [l_j \doteq \varsigma(b'\{\!\{n+1 \leftarrow e'\}\!\}), m_i'\{\!\{n \leftarrow e'\}\!\}{}^{i \in 1..n, i \neq j}]}$$

    And by the definition of substitution we are done.
  - $d = [m_i{}^{i \in 1..n}] \lhd < l_j := b >$. Then $d' = [l_j := b', m_i'{}^{i \in 1..n, i \neq j}]$ where $b \Rightarrow b'$ and $m_i \Rightarrow m_i'$.
    Now by induction hypothesis we have $b\{\!\{n \leftarrow e\}\!\} \Rightarrow b'\{\!\{n \leftarrow e'\}\!\}$ and $m_i\{\!\{n \leftarrow e\}\!\} \Rightarrow m_i'\{\!\{n \leftarrow e'\}\!\}$ allowing us to conclude:

$$\frac{b\{\!\{n \leftarrow e\}\!\} \Rightarrow b'\{\!\{n \leftarrow e'\}\!\} \quad m_i\{\!\{n \leftarrow e\}\!\} \Rightarrow m_i'\{\!\{n \leftarrow e'\}\!\}}{[m_i\{\!\{n \leftarrow e\}\!\}{}^{i \in 1..n}] \lhd < l_j := b\{\!\{n \leftarrow e\}\!\} > \Rightarrow [l_j := b'\{\!\{n \leftarrow e'\}\!\}, m_i'\{\!\{n \leftarrow e'\}\!\}{}^{i \in 1..n, i \neq j}]}\ Oa$$

    And by the definition of substitution we are done.

- $d = [m_i{}^{i \in 1..n}]$ and therefore $d' = [m_i'{}^{i \in 1..n}]$ where $m_i \Rightarrow m_i'$. Then we resolve using the induction hypothesis.

Proposition A.10 Let $a, b \in \mathcal{T}_{\varsigma_{db}'}$. If $a \Rightarrow b$ then $b \Rightarrow a^*$.

Proof. We use induction on a and lemmas A.8 and A.9.

- $a = n$. Then $b = n$ and we are done.

- $a = c.l$ where $c.l$ is not a redex. Then $b = c'.l$ and $c \Rightarrow c'$. By induction hypothesis $c' \Rightarrow c^*$ then by rule (I) we may conclude $c'.l \Rightarrow c^*.l$ and since $(c.l)^* = c^*.l$ we are done.

- $a = [l_j \doteq \varsigma(d), m_i{}^{i \in 1..n, i \neq j}].l_j$. Then there are two subcases we must consider.

- $b = c'.l_j$ with $[l_j \doteq \varsigma(d), m_i{}^{i\in 1..n, i\neq j}] \Rightarrow c'$. Studying the rules we verify that $c' = [m'_j, m'_i{}^{i\in 1..n,i\neq j}]$ where $(l_j \doteq \varsigma(d)) \Rightarrow m'_j$ and $m_i \Rightarrow m'_i$. And we may further state that $m'_j = (l_j \doteq \varsigma(d'))$ with $d \Rightarrow d'$. Then using the induction hypothesis we may obtain

$$\frac{d' \Rightarrow d^* \quad m'_i \Rightarrow m^*_i}{[l_j \doteq \varsigma(d'), m'_i{}^{i\in 1..n,i\neq j}].l_j \Rightarrow d^*\{\!\{1 \leftarrow [l_j \doteq \varsigma(d^*), m^*_i{}^{\ i\in 1..n,i\neq j}]\}\!\}}\ Im$$

  Note that $[l_j \doteq \varsigma(d'), m'_i{}^{i\in 1..n,i\neq j}].l_j = c'.l_j$ and also $d^*\{\!\{1 \leftarrow [l_j \doteq \varsigma(d^*), m^*_i{}^{\ i\in 1..n,i\neq j}]\}\!\} = ([l_j \doteq \varsigma(d), m_i{}^{i\in 1..n,i\neq j}].l_j)^*$.

- $b = d'\{\!\{1 \leftarrow [l_j \doteq \varsigma(d'), m'_i{}^{i\in 1..n,i\neq j}]\}\!\}$. Then we must have $d \Rightarrow d'$ and $m_i \Rightarrow m'_i$ with $i \in 1..n, i \neq j$. Then by induction hypothesis $d' \Rightarrow d^*$ and $m'_i \Rightarrow m^*_i$. Thus

$$\frac{\dfrac{d' \Rightarrow d^*}{l \doteq \varsigma(d') \Rightarrow l \doteq \varsigma(d^*)}\ M, Mm \qquad m'_i \Rightarrow m^*_i \ \ i \in 1..n, i \neq j}{[l \doteq \varsigma(d'), m'_i{}^{i\in 1..n,i\neq j}] \Rightarrow [l \doteq \varsigma(d^*), m^*_i{}^{\ i\in 1..n,i\neq j}]}\ Ob$$

  Now since $d' \Rightarrow d^*$ by Lemma A.9 we may conclude
  $d'\{\!\{1 \leftarrow [l_j \doteq \varsigma(d'), m'_i{}^{i\in 1..n,i\neq j}]\}\!\} \Rightarrow d^*\{\!\{1 \leftarrow [l_j \doteq \varsigma(d^*), m^*_i{}^{\ i\in 1..n,i\neq j}]\}\!\}$

- $a = [l_j := d, m_i{}^{i\in 1..n,i\neq j}].l_j$. Then there are two subcases we must consider.

  - $b = c'.l_j$ with $[l_j := d, m_i{}^{i\in 1..n,i\neq j}] \Rightarrow c'$. Studying the rules one may verify $c' = [m'_j, m'_i{}^{i\in 1..n,i\neq j}]$ where $(l_j := d) \Rightarrow m'_j$ and $m_i \Rightarrow m'_i$. And we may further state that $m'_j = (l_j := d')$ with $d \Rightarrow d'$. Then using the induction hypothesis we may obtain

$$\frac{d' \Rightarrow d^*}{[l_j := d', m'_i{}^{i\in 1..n,i\neq j}].l_j \Rightarrow d^*}\ Ia$$

    Note that $[l_j := d', m'_i{}^{i\in 1..n,i\neq j}].l_j. = c'.l_j$ and $d^* = [l_j := d, m_i{}^{i\in 1..n,i\neq j}].l_j^*$

  - $b = d'$ with $d \Rightarrow d'$ and by induction hypothesis $d' \Rightarrow d^*$ and we are done.

- $a = [m_i{}^{i\in 1..n}]$. Then it must be that $b = [m'_i{}^{i\in 1..n}]$ where $m_i \Rightarrow m'_i$. By induction hypothesis we have that $m'_i \Rightarrow m^*_i$ and therefore applying rule (Ob) we get $[m'_i{}^{i\in 1..n}] \Rightarrow [m^*_i{}^{\ i\in 1..n}]$. Note that $[m_i{}^{i\in 1..n}]^* = [m^*_i{}^{\ i\in 1..n}]$.

- $a = d \lhd < m >$ and a is not a redex. Then $b = d' \lhd < m' >$ with $d \Rightarrow d'$ and $m \Rightarrow m'$. We conclude using the induction hypothesis.

- $a = [m_i{}^{i\in 1..n}] \lhd < l_j \doteq \varsigma(d) >$. Then we must consider two subcases:

  - $b = a' \lhd < m' >$ where $[m_i{}^{i\in 1..n}] \Rightarrow a'$ and $(l_j \doteq \varsigma(d)) \Rightarrow m'$. Then it must be that $a' = [m'_i{}^{i\in 1..n}]$ with $m_i \Rightarrow m'_i$ and $m' = (l_j \doteq \varsigma(d'))$ with $d \Rightarrow d'$. Then by induction hypothesis we obtain $m'_i \Rightarrow m^*_i$ and $d' \Rightarrow d^*$ and therefore

$$\frac{d' \Rightarrow d^* \quad m'_i \Rightarrow m^*_i}{[m'_i{}^{i\in 1..n}] \lhd < l_j \doteq \varsigma(d') > \Rightarrow [l_j \doteq \varsigma(d^*), m^*_i{}^{\ i\in 1..n,i\neq j}]}\ Om$$

  - $b = [l_j \doteq \varsigma(d'), m'_i{}^{i\in 1..n,i\neq j}]$ where $m_i \Rightarrow m'_i$ for $i \in 1..n, i \neq j$ and $d \Rightarrow d'$. Then by induction hypothesis we obtain $m'_i \Rightarrow m^*_i$ and $d' \Rightarrow d^*$ and obtain

$$\frac{d' \Rightarrow d^* \quad m'_i \Rightarrow m^*_i \ \ i \in 1..n, i \neq j}{[l_j \doteq \varsigma(d'), m'_i{}^{i\in 1..n,i\neq j}] \Rightarrow [l_j \doteq \varsigma(d^*), m^*_i{}^{\ i\in 1..n,i\neq j}]}\ Ob$$

- $a = [m_i{}^{i\in 1..n}] \lhd < l_j := d >$. Then we must consider two subcases:

  - $b = a' \lhd < m' >$ where $[m_i{}^{i\in 1..n}] \Rightarrow a'$ and $(l_j := d) \Rightarrow m'$. Then it must be that $a' = [m'_i{}^{i\in 1..n}]$ with $m_i \Rightarrow m'_i$ and $m' = (l_j := d')$ with $d \Rightarrow d'$. Then by induction hypothesis we obtain $m'_i \Rightarrow m^*_i$ and $d' \Rightarrow d^*$ and therefore

$$\frac{d' \Rightarrow d^* \quad m_i' \Rightarrow m_i^*}{[m_i'\ ^{i\in 1..n}] \lhd < l_j := d' > \Rightarrow [l_j := d^*, m_i^*\ \ ^{i\in 1..n, i\neq j}]}\ Oa$$

- $b = [l_j := d', m_i'\ ^{i\in 1..n, i\neq j}]$ where $m_i \Rightarrow m_i'$ for $i \in 1..n, i \neq j$ and $d \Rightarrow d'$. Then by induction hypothesis we obtain $m_i' \Rightarrow m_i^*$ and $d' \Rightarrow d^*$ and therefore

$$\frac{d' \Rightarrow d^* \quad m_i' \Rightarrow m_i^* \quad i \in 1..n, i \neq j}{[l_j := d', m_i'\ ^{i\in 1..n, i\neq j}] \Rightarrow [l_j := d^*, m_i^*\ \ ^{i\in 1..n, i\neq j}]}\ Ob$$

The cases for methods and fields are similar.

▪

As an immediate corollary of Lemma A.10 we may conclude that the parallel reduction relation satisfies the diamond property. This entails confluence of ς_{db}^{f}.

Proof.[of Lemma 5.15] Firstly, note that $\rightarrow_{\varsigma_{db}^f} \subseteq \Rightarrow \subseteq \twoheadrightarrow_{\varsigma_{db}^f}$. The first inclusion follows from the form of the rules defining the parallel reduction relation and the fact that for every $a \in \mathcal{T}_{\varsigma_{db}^f}$ we have $a \Rightarrow a$. The second inclusion may be proved by induction. Therefore we have that $\Rightarrow^* = \twoheadrightarrow_{\varsigma_{db}^f}$. The diamond property of $\Rightarrow$ concludes the proof.

▪

A.2.2 Strong Normalization of *BES*

We shall denote the rewrite system whose only rule is SW by the SW-rewriting system or SW for short.

Definition A.11 (External form) We define an *external form* as a term $a[\Uparrow^{i_1} (k_1)][\Uparrow^{i_2} (k_2)]...[\Uparrow^{i_n} (k_n)]$ where k_j may be $@l$, $b/$ or $\uparrow$ in $\mathcal{T}_{\varsigma_{dbes}}$ such that it verifies the following conditions:

1. a is not a closure.
2. $\forall j \in 1..n-1$ if $k_j = @l$ then $i_j \geq i_{j+1}$.

Note that in an external form $a[\Uparrow^{i_1} (k_1)][\Uparrow^{i_2} (k_2)]...[\Uparrow^{i_n} (k_n)]$ the only possible SW-reductions may occur in a or in k_i with $i \in 1..n$. We shall use the following lemmas for the proof of weak normalization.

Lemma A.12 If $a[s_1][s_2]...[s_n]$ is an external form then for every $s_{n+1} = \Uparrow^{i_{n+1}} (k_{n+1})$ there is an external form $b = a[s_{j_1}][s_{j_2}]...[s_{j_{n+1}}]$ such that $[j_1, ..., j_{n+1}]$ is a permutation of $[1, ..., n+1]$ of the form $[1, \ldots, i-1, n+1, i, \ldots, n]$ and $a[s_1][s_2]...[s_{n+1}] \twoheadrightarrow_{SW} b$.

Proof. By induction on n.

- Case $n = 0$. Since a is not a closure, $a[s_1]$ is an external form and $a[s_1] \twoheadrightarrow_{SW} a[s_1]$.
- Case $n > 0$. We may assume $s_n = \Uparrow^{i_n} (@l)$ and $i_{n+1} > i_n$ otherwise we are done. Now we consider the following cases:
  - If $k_{n+1} = @l'$ then $a[s_1][s_2]...[s_n][\Uparrow^{i_{n+1}} (@l')] \rightarrow_{SW} a[s_1][s_2]...[\Uparrow^{i_{n+1}} (@l')][s_n]$. Now we apply the induction hypothesis to $a[s_1][s_2]...[s_{n-1}]$ and $[\Uparrow^{i_{n+1}} (@l')]$ and obtain an external form e such that $a[s_1][s_2]...[s_{n-1}][\Uparrow^{i_{n\div 1}} (@l')] \rightarrow_{SW} e$. Now the resulting permutation of $[1, \ldots, n-1, n+1]$ can have one of two forms
    * $[1, \ldots, n-1, n+1]$. Then e ends in the substitution $[\Uparrow^{i_{n+1}} (@l')]$ in which case $e[s_n]$ is an external form. The resulting permutation of $[1, \ldots, n+1]$ is $[1, \ldots, n-1, n+1, n]$.
    * $[1, \ldots, i-1, n+1, i, \ldots, n-1]$. Then e ends in the substitution $[s_{n-1}] = [\Uparrow^{i_{n-1}} (@l'')]$ and therefore $e[s_n]$ is an external form since $a[s_1][s_2]...[s_n]$ is. The resulting permutation is $[1, \ldots, i-1, n+1, i, \ldots, n-1, n]$.

  Therefore we have $a[s_1][s_2]...[s_n][\Uparrow^{i_{n+1}} (@l')] \rightarrow_{SW} a[s_1][s_2]...[\Uparrow^{i_{n+1}} (@l')][s_n] \twoheadrightarrow_{SW} e[s_n]$ and $e[s_n]$ is an external form.

- If $k_{n+1} = c/$ then we may apply SW and obtain $a[s_1][s_2]...[\Uparrow^{i_{n+1}} (c/)][\Uparrow^{i_n} (@l)]$. Now by applying the induction hypothesis to the term $a[s_1][s_2]...[s_{n-1}]$ and $[\Uparrow^{i_{n+1}} (c/)]$ we obtain an external form e. Now the resulting permutation of $[1, \ldots, n-1, n+1]$ can have one of two forms
  * $[1, \ldots, n-1, n+1]$. Then e ends in the substitution $[\Uparrow^{i_{n+1}} (c/)]$ in which case $e[s_n]$ is an external form. The resulting permutation of $[1, \ldots, n+1]$ is $[1, \ldots, n-1, n+1, n]$.
  * $[1, \ldots, i-1, n+1, i, \ldots, n-1]$. Then e ends in the substitution $[s_{n-1}] = [\Uparrow^{i_{n-1}} (@l'')]$ and therefore $e[s_n]$ is an external form since $a[s_1][s_2]...[s_n]$ is. The resulting permutation is $[1, \ldots, i-1, n+1, i, \ldots, n-1, n]$.

The case where $k_{n+1} =\uparrow$ is similar.

•

Lemma A.13 If $a[s_1][s_2]...[s_n]$ is a term in $\mathcal{T}_{\varsigma dbes}$ such that a is not a closure then for every $s_{n+1} =\Uparrow^{i_{n+1}} (k_{n+1})$ there is an external form $b = a[s_{i_1}][s_{i_2}]...[s_{i_{n+1}}]$ such that $[i_1, ..., i_{n+1}]$ is a permutation of $[1, ..., n+1]$ and $a[s_1][s_2]...[s_{n+1}] \twoheadrightarrow_{SW} b$.

Proof. By induction on n and using Lemma A.12.

Lemma A.14 (Weak Normalization of SW) The SW-rewriting system is weakly normalizing.

Proof. Using the technique presented in [KR97]. Let a be any term in $\mathcal{T}_{\varsigma dbes}$. We shall use structural induction on a to prove that there exists $a' \in SW$-normal forms such that $a \twoheadrightarrow_{SW} a'$.

- If $a = n$ then we are done.
- The other cases where a is not a closure are straightforward since the normal form is computed by obtaining the normal forms of the subterms.
- So suppose $a = b[s_1][s_2]...[s_n]$ and b is not a closure. By the previous lemma, $a \twoheadrightarrow_{SW} b[s_{i_1}][s_{i_2}]...[s_{i_n}]$. Since b and every s_{i_j} is simpler than a we may apply the induction hypothesis and obtain normal forms b' and s'_{i_j} such that $b \twoheadrightarrow_{SW} b'$ and $s_{i_j} \twoheadrightarrow_{SW} s'_{i_j}$. Note that if $s_{i_j} =\Uparrow^{i_j} (k_j)$ with k_j being of the form $@l$ or $c/$ or $\uparrow$ then s'_{i_j} must be of the form $s'_{i_j} =\Uparrow^{i_j} (k'_j)$ where k'_j is $@l$ or $c'/$(with $c \twoheadrightarrow_{SW} c'$) or $\uparrow$. But then we may obtain a normal form for a, namely $b'[s'_{i_1}][s'_{i_2}]...[s'_{i_n}]$.

•

For the proof of strong normalization of SW we shall need the following lemma which we state without proof (see [Oos98] for a proof and some historical remarks) .

Lemma A.15 Let $A = (S, R)$ be an Abstract Reduction System such that:

- R is weakly normalizing.
- R locally confluent.
- there exists a function $f : S \rightarrow \mathbb{N}$ such that aRb implies $f(a) < f(b)$.

then R is strongly normalizing.

Lemma A.16 (Strong Normalization of SW) The SW-rewrite system is strongly normalizing.

Proof. We define the following function $f : \mathcal{T}_{\varsigma dbes} \mapsto \mathbb{N}_{\geq\infty}$

$$\begin{array}{lll@{\qquad\qquad}lll}
f(n) & \overset{\text{def}}{=} & 1 & f([m_i{}^{i\in 1..n}]) & \overset{\text{def}}{=} & \sum_{i=1}^{n} f(m_i) \\
f(a.l) & \overset{\text{def}}{=} & f(a) & f(a[s]) & \overset{\text{def}}{=} & 2f(a) + f(s) \\
f(a \lhd < m >) & \overset{\text{def}}{=} & f(a) + f(m) & f(\Uparrow (s)) & \overset{\text{def}}{=} & 2f(s) \\
f(\varsigma(a)) & \overset{\text{def}}{=} & f(a) & f(@l) & \overset{\text{def}}{=} & 1 \\
f(l \doteq g) & \overset{\text{def}}{=} & f(g) & f(b/) & \overset{\text{def}}{=} & 1 \\
f(l := b) & \overset{\text{def}}{=} & f(b) & f(\uparrow) & \overset{\text{def}}{=} & 1
\end{array}$$

We prove by induction on a that if $a \rightarrow_{SW} b$ then we have $f(a) < f(b)$. Below we consider the cases where the reduction takes place at the root, the other cases hold by the induction hypothesis.

In fact, since $k > i$ we have

$$f(a[\Uparrow^i (@l)][\Uparrow^k (s)]) = 4f(a) + 2^i + 2^i + 2^k f(s) < 4f(a) + 2^k f(s) + 2^k f(s) + 2^i = f(a[\Uparrow^k (s)][\Uparrow^i (@l)])$$

Since SW is locally confluent (the rule SW overlaps itself and the corresponding critical pair may be closed in SW) the strong normalization property follows from weak normalization and by applying Lemma A.15.

∎

For the proof of strong normalization of *ESDB* we shall need Lemma 2.6.

Lemma A.17 (SN of $ESDB \cup \{MO, FO, FI\}$**)** $ESDB \cup \{MO, FO, FI\}$ is strongly normalizing.

Proof. We define a lexicographic order based on a measure $f : \mathcal{T}_{\varsigma_{dbes}} \mapsto \mathbb{N}_{\geq 2} \times \mathbb{N}_{\geq 2}$ by providing two polynomial components $h : \mathcal{T}_{\varsigma_{dbes}} \mapsto \mathbb{N}_{\geq 2}$ and $k : \mathcal{T}_{\varsigma_{dbes}} \mapsto \mathbb{N}_{\geq 2}$ and then letting $f(x) \stackrel{\text{def}}{=} (h(x), k(x))$. The functions h and k are defined in Figure A.2. Note that for any substitution s, $h(s) \geq 2$.

$$\begin{array}{lll@{\qquad\qquad}lll}
h(n) & \stackrel{\text{def}}{=} & 2^n & k(n) & \stackrel{\text{def}}{=} & 2^n \\
h(a.l) & \stackrel{\text{def}}{=} & h(a) + h(l) & k(a.l) & \stackrel{\text{def}}{=} & k(a) + k(l) \\
h(a \lhd < m >) & \stackrel{\text{def}}{=} & h(a) + h(m) + 1 & k(a \lhd < m >) & \stackrel{\text{def}}{=} & k(a) + k(m) + 1 \\
h(\varsigma(a)) & \stackrel{\text{def}}{=} & h(a) + 1 & k(\varsigma(a)) & \stackrel{\text{def}}{=} & k(a) + 1 \\
h(l := a) & \stackrel{\text{def}}{=} & h(l) + h(a) & k(l := a) & \stackrel{\text{def}}{=} & k(l) + k(a) \\
h(l \doteq g) & \stackrel{\text{def}}{=} & h(l) + h(g) & k(l \doteq g) & \stackrel{\text{def}}{=} & k(l) + k(g) \\
h([m_i{}^{i \,\in\, 1..n}]) & \stackrel{\text{def}}{=} & \sum_{i=1}^n h(m_i) + n + 1 & k([m_i{}^{i \,\in\, 1..n}]) & \stackrel{\text{def}}{=} & \sum_{i=1}^n k(m_i) + n + 1 \\
h(a[s]) & \stackrel{\text{def}}{=} & h(a)h(s) & k(a[s]) & \stackrel{\text{def}}{=} & k(a)k(s) \\
h(\Uparrow (s)) & \stackrel{\text{def}}{=} & h(s) & k(\Uparrow (s)) & \stackrel{\text{def}}{=} & 2k(s) \\
h(b/) & \stackrel{\text{def}}{=} & h(b) & k(b/) & \stackrel{\text{def}}{=} & k(b) \\
h(@l) & \stackrel{\text{def}}{=} & 3 & k(@l) & \stackrel{\text{def}}{=} & 3 \\
h(\uparrow) & \stackrel{\text{def}}{=} & 2 & k(\uparrow) & \stackrel{\text{def}}{=} & 3 \\
 & & \text{where } h(l) = k(l) \stackrel{\text{def}}{=} 1 & & &
\end{array}$$

Figure A.2: Polynomial interpretation

Now we may show by structural induction on u that if $u \rightarrow_{BES \cup \{MO,FO,FI,CO\}} v$ then $f(u) > f(v)$, and if $u \rightarrow_{SW} v$ then $f(u) = f(v)$. Finally, we conclude by applying Lemma 2.6 and A.16.

We shall consider reductions at the root only since for internal reduction the property holds by the induction hypothesis.

- if $u = [l_j := a, m_i^{i\ \in 1..n, i \neq j}].l_j \rightarrow_{FI} a = v$ then $h(u) = h(l_j := a) + \sum_{i=1, i \neq j}^n h(m_i) + n + 1 > h(a) = h(v)$.
- if $u = [m_i{}^{i \in 1..n}] \lhd < l_j \doteq f > \rightarrow_{MO} [l_j \doteq f, m_i{}^{i \in 1..n, i \neq j}] = v$, then $h(u) = (\sum_{i=1}^n h(m_i) + n + 1) + h(l_j = f) + 1 > h(l_j = f) + \sum_{i=1}^{n, i \neq j} h(m_i) + n + 1$.
- if $u = [m_i{}^{i \in 1..n}] \lhd < l_j := a > \rightarrow_{FO} [l_j := a, m_i{}^{i \in 1..n, i \neq j}] = v$, then $h(u) = (\sum_{i=1}^n h(m_i) + n + 1) + h(l_j := a) + 1 > h(l_j := a) + \sum_{i=1}^{n, i \neq j} h(m_i) + n + 1$.
- if $u = (\varsigma(c))[s] \rightarrow_{SM} \varsigma(c[\Uparrow (s)]) = v$, then $h(u) = (h(c) + 1)h(s) > h(c)h(s) + 1 = h(v)$.
- if $u = [m_i{}^{i\ \in\ 1..n}][s] \rightarrow_{SO} [m_i[s]^{i\ \in\ 1..n}] = v$, then $h(u) = (\sum_{i=1}^n h(m_i) + n + 1)h(s) > \sum_{i=1}^n h(m_i)h(s) + n + 1 = h(v)$.

- if $u = (l := a)[s] \rightarrow_{SF} l := a[s] = v$, then $h(u) = (1 + h(a))h(s) > 1 + h(a)h(s) = h(v)$.
- if $u = (l \doteq g)[s] \rightarrow_{SB} l \doteq g[s] = v$, then $h(u) = (1 + h(g))h(s) > 1 + h(g)h(s) = h(v)$.
- if $u = a.l[s] \rightarrow_{SI} a[s].l = v$, then $h(u) = (h(a) + 1)h(s) > h(a)h(s) + 1 = h(v)$.
- if $u = a \lhd < m > [s] \rightarrow_{SU} a[s] \lhd < m[s] >= v$, then $h(u) = (h(a) + h(m) + 1)h(s) > h(a)h(s) + h(m)h(s) + 1 = h(v)$.
- if $u = 1[a/] \rightarrow_{FVar} a = v$, then $h(u) = 2h(a) > h(a) = h(v)$.
- if $u = n + 1[a/] \rightarrow_{RVar} n = v$, then $h(u) = 2^{n+1}h(a) > 2^n = h(v)$.
- if $u = 1[@l] \rightarrow_{FInv} 1.l = v$, then $h(u) = 6 > 3 = h(v)$.
- if $u = n + 1[@l] \rightarrow_{RInv} n + 1 = v$, then $h(u) = 2^{n+1}3 > 2^{n+1} = h(v)$.
- if $u = 1[\Uparrow (s)] \rightarrow_{FVarLift} 1 = v$, then $h(u) = 2h(s) > 2 = h(v)$.
- if $u = n + 1[\Uparrow (s)] \rightarrow_{RVarLift} n[s][\uparrow] = v$, then
  $f(u) = (2^{n+1}h(s), 2^{n+2}k(s)) > (2^{n+1}h(s), 2^n k(s)3) = f(v)$.
- if $u = n[\uparrow] \rightarrow_{VarShift} n + 1 = v$, then $f(u) = (2^{n+1}, 2^n 3) > (2^{n+1}, 2^{n+1}) = f(v)$.
- if $u = a[\Uparrow^i (@l_j)][\Uparrow^i ([l_j := b, {m_i}^{i \in 1..n, i \neq j}]/)] \rightarrow_{CO} a[\Uparrow^i (b/)] = v$, then $h(u) = h(a)3(h(l := b) + \sum_{i=1}^{n, i \neq j} h(m_i) + n + 1) > h(a)h(b) = h(v)$.
- if $u = a[\Uparrow^i (@l)][\Uparrow^k (s)] \rightarrow_{SW} a[\Uparrow^k (s)][\Uparrow^i (@l)]$ with $k > i$ then $f(u) = (h(a)3h(s), k(a)2^i.3.2^k.k(s)) = (h(a)3h(s), k(a)2^i.3.2^k.k(s)) = h(v)$.

A.3 A de Bruijn Notation for Higher-Order Rewriting

A.3.1 From de Bruijn Valuations to Correct Valuations

In this section we shall prove that the translation of a valid de Bruijn valuation (Definition 6.67) does not depend on the choice of the t-metavariable.

Lemma A.18 (Renaming and the $U_\bullet^\bullet(\bullet)$ translation) Let l be a label of variables, z and y be two variables, S be a set of variables and a be a de Bruijn term such that:

1. $z \in l$ and $z \notin S$,
2. y does not occur in $U_l^S(a)$, and
3. $\mathtt{Names}(FV(a) \setminus\!\!\setminus |l|) \subseteq S$.

Then we have $U_l^S(a)\{z \leftarrow y\} =_\alpha U_{l\{z \leftarrow y\}}^S(a)$.

Proof. The condition $\mathtt{Names}(FV(a) \setminus\!\!\setminus |l|) \subseteq S$ is required for $U_l^S(a)$ and $U_{l\{z \leftarrow y\}}^S(a)$ to be defined. The proof proceeds by induction on a. The case where a is of the form $f(a_1, \ldots, a_n)$ follows from the induction hypothesis so we consider the remaining ones.

- $a = n$. We have two further cases to consider:
  - $1 \leq n \leq |l|$. Then on one hand $U_l^S(n)\{z \leftarrow y\} = \mathtt{at}(l, n)\{z \leftarrow y\} = \mathtt{at}(l\{z \leftarrow y\}, n) = U_{l\{z \leftarrow y\}}^S(n)$.
  - $n > |l|$. Then since $z \notin S$ we have $U_l^S(n)\{z \leftarrow y\} = x_{n-|l|}\{z \leftarrow y\} = x_{n-|l|} = x_{n-|l\{z \leftarrow y\}|} = U_{l\{z \leftarrow y\}}^S(n)$.

- $a = \xi(a_1, \ldots, a_n)$. Then we reason as follows:

$$\begin{array}{lcll}
U^S_l(a)\{z \leftarrow y\} & = & \xi x.(U^S_{xl}(a_1), \ldots, U^S_{xl}(a_n))\{z \leftarrow y\} & \\
& = & \xi x.(U^S_{xl}(a_1)\{z \leftarrow y\}, \ldots, U^S_{xl}(a_n)\{z \leftarrow y\}) & (z \in l \text{ hence } z \neq x \\
& & & \text{and } y \text{ not in } U^S_l(a)) \\
& =_\alpha & \xi x.(U^S_{xl\{z\leftarrow y\}}(a_1), \ldots, U^S_{xl\{z\leftarrow y\}}(a_n)) & \text{(i.h.)} \\
& =_\alpha & \xi v.(U^S_{xl\{z\leftarrow y\}}(a_1)\{x \leftarrow v\}, \ldots, U^S_{xl\{z\leftarrow y\}}(a_n)\{x \leftarrow v\}) & (v \text{ fresh}) \\
& =_\alpha & \xi v.(U^S_{vl\{z\leftarrow y\}\{x\leftarrow v\}}(a_1), \ldots, U^S_{vl\{z\leftarrow y\}\{x\leftarrow v\}}(a_n)) & \text{(i.h.)} \\
& =_\alpha & \xi v.(U^S_{vl\{z\leftarrow y\}}(a_1), \ldots, U^S_{vl\{z\leftarrow y\}}(a_n)) & (x \neq y \text{ and } x \notin l) \\
& =_\alpha & \xi v.(U^S_{(wl\{z\leftarrow y\})\{w\leftarrow v\}}(a_1), \ldots, U^S_{(wl\{z\leftarrow y\})\{w\leftarrow v\}}(a_n)) & \\
& =_\alpha & \xi v.(U^S_{wl\{z\leftarrow y\}}(a_1)\{w \leftarrow v\}, \ldots, U^S_{wl\{z\leftarrow y\}}(a_n)\{w \leftarrow v\}) & \text{(i.h.)} \\
& =_\alpha & \xi w.(U^S_{wl\{z\leftarrow y\}}(a_1), \ldots, U^S_{wl\{z\leftarrow y\}}(a_n)) & (w \in l\{z \leftarrow y\} \cup S) \\
& = & U^S_{l\{z\leftarrow y\}}(a) &
\end{array}$$

Since the translation function on the *LHS* and *RHS* of the equation to prove may have chosen different variables for the ξ binder we relate them through a fresh variable v.

∎

Lemma A.19 Let a and b be de Bruijn terms, l and l' labels of binder indicators and α a binder indicator. Then for $j \geq 0$ we have: $\mathit{Value}^{j+1}(l, a) = \mathit{Value}^{j+1}(l', b)$ implies $\mathit{Value}^j(\alpha l, a) = \mathit{Value}^j(\alpha l', b)$.

Proof. By induction on a.

- $a = m$. Since $\mathit{Value}^{j+1}(l, m) = \mathit{Value}^{j+1}(l', b)$ we have $b = n$ for some index n. We proceed by cases:
  - $m \leq j + 1$. Then since $\mathit{Value}^{j+1}(l, m) = m = \mathit{Value}^{j+1}(l', n)$ by Definition 6.40 we have $n = m$ and therefore $\mathit{Value}^j(\alpha l, m) = \mathit{Value}^j(\alpha l', n)$.
  - $m > j + 1$. We have two different cases:
    * $m-(j+1) \leq |l|$. Then by hypothesis we have $\mathit{Value}^{j+1}(l, m) = \mathtt{at}(l, m-(j+1)) = \mathit{Value}^{j+1}(l', n)$, and hence $0 < n - (j + 1) \leq |l'|$ and $\mathtt{at}(l, m - (j + 1)) = \mathtt{at}(l', n - (j + 1))$. Therefore $\mathit{Value}^j(\alpha l, m) = \mathit{Value}^j(\alpha l', n)$ since $1 < m - j \leq |\alpha l|$ and $1 < n - j \leq |\alpha l'|$.
    * $m - (j + 1) > |l|$. Then by hypothesis we have $\mathit{Value}^{j+1}(l, m) = x_{m-(j+1)-|l|} = \mathit{Value}^{j+1}(l', n)$, and hence $n - (j + 1) > |l'|$ and $m - (j + 1) - |l| = n - (j + 1) - |l'|$. Therefore $\mathit{Value}^j(\alpha l, m) = x_{m-j-|\alpha l|} = \mathit{Value}^j(\alpha l', n)$.
- $a = f(a_1, \ldots, a_n)$. By Definition 6.40 and the hypothesis we have necessarily that $b = f(b_1, \ldots, b_n)$ and $\mathit{Value}^{j+1}(l, a_i) = \mathit{Value}^{j+1}(l', b_i)$ for $1 \leq i \leq n$ so that by induction hypothesis we can conclude $\mathit{Value}^j(\alpha l, a_i) = \mathit{Value}^j(\alpha l', b_i)$ and thus $\mathit{Value}^j(\alpha l, a) = \mathit{Value}^j(\alpha l', b)$.
- $a = \xi(a_1, \ldots, a_n)$. By Definition 6.40 and the hypothesis we have necessarily that $b = \xi(b_1, \ldots, b_n)$ and $\mathit{Value}^{j+2}(l, a_i) = \mathit{Value}^{j+2}(l', b_i)$ for $1 \leq i \leq n$ so that by induction hypothesis we can conclude $\mathit{Value}^{j+1}(\alpha l, a_i) = \mathit{Value}^{j+1}(\alpha l', b_i)$ and thus $\mathit{Value}^j(\alpha l, a) = \mathit{Value}^j(\alpha l', b)$.

∎

Note that the converse of Lemma A.19 does not hold (for α may already be present in l or l'). Indeed, $\mathit{Value}^0(\alpha\alpha, 2) = \mathit{Value}^0(\alpha\alpha, 1)$, yet $\mathit{Value}^1(\alpha, 2) \neq \mathit{Value}^1(\alpha, 1)$. The value function is used to determine when a de Bruijn valuation is valid or not. It is defined in the SERS_{db} formalism in order to describe reduction on de Bruijn terms. A natural question which arises is that of the relationship between value equivalent de Bruijn terms considered as named terms via de $U^\bullet_\bullet(\bullet)$ translation in the *SERS* formalism. The following lemma investigates this matter.

Lemma A.20 Let $a, b \in \mathcal{T}_{db}$, S be a set of variables, l, l' be labels of binder indicators, k a label of variables, and θ_v a variable assignment. If both $U^S_{\theta_v(l)k}(a)$ and $U^S_{\theta_v(l')k}(b)$ are defined, then $\mathit{Value}(l, a) = \mathit{Value}(l', b)$ implies $U^S_{\theta_v(l)k}(a) =_\alpha U^S_{\theta_v(l')k}(b)$.

Proof. By induction on a.

- $a = m$. Since $\mathit{Value}^0(l,m) = \mathit{Value}^0(l',b)$ we have $b = n$ for some index n. The left hand side reads

$$U^S_{\theta_v(l)k}(m) \;=\; \begin{cases} \mathtt{at}(\theta_v(l),m) & m \leq |l| \\ \mathtt{at}(k, m-|l|) & |l| < m \leq |kl| \\ x_{m-|lk|} & m > |kl| \text{ and } x_{m-|lk|} \in S \end{cases}$$

  We now consider the following cases:

  - $m \leq |l|$. Then since $\mathit{Value}^0(l,m) = \mathtt{at}(l,m) = \mathit{Value}^0(l',n)$ we have $n \leq |l'|$ and $\mathtt{at}(l,m) = \mathtt{at}(l',n)$. Then $U^S_{\theta_v(l')k}(n) = \mathtt{at}(\theta_v(l'),n) = \theta_v(\mathtt{at}(l',n)) = \theta_v(\mathtt{at}(l,m)) = U^S_{\theta_v(l)k}(m)$.
  - $|l| < m \leq |lk|$. Then since $\mathit{Value}^0(l,m) = x_{m-|l|} = \mathit{Value}^0(l',n)$ we have $n > |l'|$ and $x_{m-|l|} = x_{n-|l'|}$. Then $m - |l| = n - |l'|$. Thus $U^S_{\theta_v(l')k}(n) = \mathtt{at}(k, n-|l'|) = \mathtt{at}(k, m-|l|) = U^S_{\theta_v(l)k}(m)$.
  - $m > |lk|$. Then since $\mathit{Value}^0(l,m) = x_{m-|l|} = \mathit{Value}^0(l',n)$ we have $n > |l'|$ and $x_{m-|l|} = x_{n-|l'|}$. Then $m - |l| = n - |l'|$ and since $m > |lk|$ we also have $n > |l'k|$. Thus $U^S_{\theta_v(l')k}(n) = x_{n-|l'k|} = x_{m-|lk|} = U^S_{\theta_v(l)k}(m)$.

- $a = f(a_1,\ldots,a_n)$. Since $\mathit{Value}^0(l,a) = \mathit{Value}^0(l',b)$ we have $b = f(b_1,\ldots,b_n)$ and $\mathit{Value}^0(l,a_i) = \mathit{Value}^0(l',b_i)$ for $1 \leq i \leq n$. Then by the induction hypothesis we have $U^S_{\theta_v(l)k}(a_i) =_\alpha U^S_{\theta_v(l')k}(b_i)$ and hence $U^S_{\theta_v(l)k}(a) =_\alpha U^S_{\theta_v(l')k}(b)$.

- $a = \xi(a_1,\ldots,a_n)$. Since $\mathit{Value}^0(l,a) = \mathit{Value}^0(l',b)$ we have $b = \xi(b_1,\ldots,b_n)$ and $\mathit{Value}^1(l,a_i) = \mathit{Value}^1(l',b_i)$ for $1 \leq i \leq n$. Then $\mathit{Value}^0(\beta l,a_i) = \mathit{Value}^0(\beta l',b_i)$ holds by Lemma A.19, where in particular we can take β to be a fresh o-metavariable such that θ_v is undefined on β. Let us extend the function θ_v to β by defining $\theta_v(\beta) \overset{\text{def}}{=} z$, where z is a fresh variable such that $z \notin \theta_v(l)\theta_v(l')k \cup S$. Then since $U^S_{z\theta_v(l)k}(a_i)$ and $U^S_{z\theta_v(l')k}(b_i)$ are defined we can apply the induction hypothesis to get $U^S_{\theta_v(\beta l)k}(a_i) = U^S_{z\theta_v(l)k}(a_i) =_\alpha U^S_{z\theta_v(l')k}(b_i) = U^S_{\theta_v(\beta l')k}(b_i)$ for $1 \leq i \leq n$.

  We now reason as follows:

$$\begin{array}{lll} & U^S_{\theta_v(l)k}(\xi(a_1,\ldots,a_n)) & \\ = & \xi x.(U^S_{x\theta_v(l)k}(a_1),\ldots,U^S_{x\theta_v(l)k}(a_n)) & (x \notin \theta_v(l)k \cup S) \\ =_\alpha & \xi z.(U^S_{x\theta_v(l)k}(a_1)\{x \leftarrow z\},\ldots,U^S_{x\theta_v(l)k}(a_n)\{x \leftarrow z\}) & \\ = & \xi z.(U^S_{z\theta_v(l)k}(a_1),\ldots,U^S_{z\theta_v(l)k}(a_n)) & (\text{L.}A.18) \\ =_\alpha & \xi z.(U^S_{z\theta_v(l')k}(b_1),\ldots,U^S_{z\theta_v(l')k}(b_n)) & (\text{i.h.}) \\ = & \xi z.(U^S_{y\theta_v(l')k}(b_1)\{y \leftarrow z\},\ldots,U^S_{y\theta_v(l')k}(b_n)\{y \leftarrow z\}) & (\text{L.}A.18) \\ =_\alpha & \xi y.(U^S_{y\theta_v(l')k}(b_1),\ldots,U^S_{y\theta_v(l')k}(b_n)) & (y \notin \theta_v(l')k \cup S \text{ by Def. 6.63}) \\ = & U^S_{\theta_v(l')k}(\xi(b_1,\ldots,b_n)) & \end{array}$$

■

Note that, in general, the converse of Lemma A.20 does not hold. Indeed it suffices to consider $k = \epsilon$, $l = \alpha$, $l' = \beta$, $a = 1$, $b = 1$, $S = \emptyset$ and the variable assignment $\theta_v\alpha = \theta_v\beta = x$. Then $U^S_x(a) = x = U^S_x(b)$ but $\mathit{Value}(\alpha,1) = \alpha \neq \beta = \mathit{Value}(\beta,1)$.

We can now show that the translation of de Bruijn valuations is correct in the sense mentioned above. This is completed in Chapter 6 as Lemma 6.68.

A.3.2 From Valid de Bruijn Valuations to Admissible Valuations

This subsection shows that if we depart from a valid valuation κ in the de Bruijn indices setting and we translate this valuation as dictated by Definition 6.67 into a valuation in the *SERS* setting, then we obtain an admissible valuation. In other words, the resulting valuation is safe (Definition 6.20) and verifies the path condition (Definition 6.21).

A word on notation: we shall use $\delta, \delta_i, \ldots$ to denote o-metavariables (that is, δ may either be a pre-bound o-metavariable such as α, or a pre-free metavariable such as $\widehat{\alpha}$).

Lemma A.21 (valid de Bruijn valuations translate to safe valuations) Let κ be a valid de Bruijn valuation for a rewrite rule (L, R), θ_v a variable assignment satisfying the requirements of Definition 6.67, S a finite set of variables, k a label of variables, and $U(L, R) = (G, D)$ the translation of (L, R). If the following conditions hold

1. $U_{(\theta_v,S,k)}(\kappa)$ is defined for *all* metavariables of G and D, and
2. θ_v is injective on the set of bound o-metavariables of (G, D).

then $U_{(\theta_v,S,k)}(\kappa)$ is safe for (G, D).

Proof. Recall that $U_{(\theta_v,S,k)}(\kappa) \stackrel{\text{def}}{=} (\theta_v, \theta_t)$ where:

$$\theta_t X \stackrel{\text{def}}{=} U^S_{\theta_v(l)k}(\kappa X_l) \quad \text{for any } X_l \in Dom(\kappa)$$

So first we must verify that for every t-metavariable in (G, D), $U_{(\theta_v,S,k)}(\kappa)$ is indeed defined, but this is guaranteed by Hypothesis 1.

In what follows we shall abbreviate $U_{(\theta_v,S,k)}(\kappa)$ with θ' for the sake of readability. Suppose that θ' is not safe for (G, D), then unwanted variable capture arises in $\theta' D$ (since the metasubstitution operator does not occur on the *LHS* of a rewrite rule, no renaming problems can arise in G). Thus there exist metaterms M_1 and M_2 and a formal parameter α such that

- $M_1[\alpha \leftarrow M_2]$ occurs in D (or equivalently $D = C[M_1[\alpha \leftarrow M_2]]$ for some metacontext C),
- θ' is defined for M_1 and M_2,
- $\theta'\alpha \in FV(\theta' M_1)$, and
- for some variable x we have $x \in BV(\theta' M_1)$ and also $x \in FV(\theta' M_2)$.

The metaterm D may be depicted as in Figure A.3(a) where l_1 denotes the label of the metacontext C.

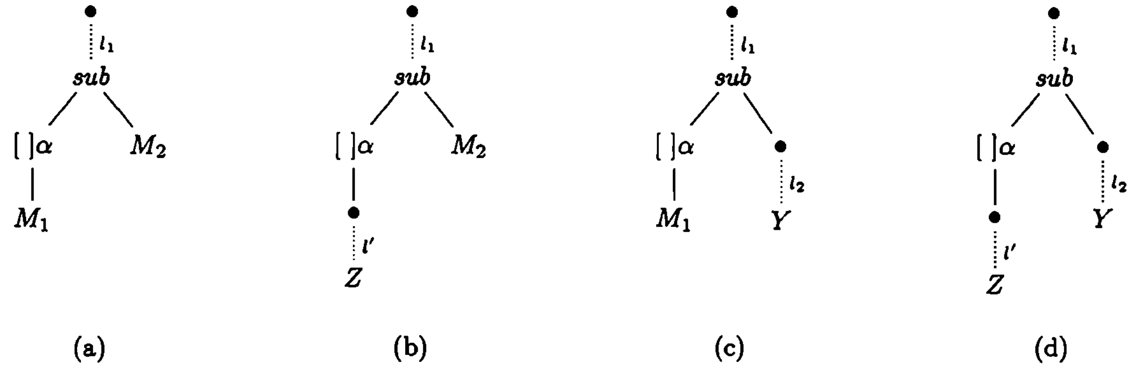


Figure A.3: Tree form for D

Before proceeding we will show the following:

Fact A.22 The free variable occurrence $x \in FV(\theta' M_2)$ cannot be bound by a formal parameter $\beta \in l_1$ (i.e. for all $\beta \in l_1$ we have $\theta'(\beta) \neq x$.). This may be verified by contradiction as follows. Suppose that for some $\beta \in l_1$ we have $\theta'(\beta) = x$. Thus, by definition of $U_{(\theta_v,S,k)}(\kappa)$ we have $\theta_v(\beta) = x$. Let us consider the *bound* occurrence of x in $\theta' M_1$. There are two possibilities:

1. x comes from the instantiation of a bound o-metavariable, so that $x = \theta'(\beta')$ for some formal parameter β' in M_1. Now since D is a well-formed pre-metaterm we must have $\beta \neq \beta'$. But $\theta'(\beta')$ is equal to $\theta_v(\beta')$ by definition, so that θ_v assigns the same value, namely x, to two different bound o-metavariables β and β' of D, thus contradicting Hypothesis 2.

2. x comes from the instantiation of a t-metavariable, so that $x \in BV(t)$ with $t = \theta' Y$ for some t-metavariable Y occurring in M_1. By Definition 6.67 we have

   $$t = U^S_{\theta_v(l)k}(\kappa Y_l)$$

   for some t-metavariable Y_l occurring in $Dom(\kappa)$ with $l = l'\alpha l_1$ (see Figure A.3(b)). Therefore by definition of the term translation function $U^\bullet_\bullet(\bullet)$ (Definition 6.63) the variable x cannot be a candidate for binding in κY_l since it already occurs in the label $\theta_v(l)k$, indeed, $\beta \in l$ and $\theta_v(\beta) = x$.

Thus we have proven that the free variable occurrence x in $\theta' M_2$ cannot be bound by a formal parameter in the label of the metacontext C.

We now return to the proof of the lemma. Let us consider where the free variable occurrence of x comes from in $\theta' M_2$. We have two possible cases:

1. There is an occurrence of an o-metavariable δ in M_2 such that $\theta_v(\delta) = x$. As observed above, since x may not be bound by a formal parameter in l_1 (i.e. there is no $\beta \in l_1$ with $\theta_v(\beta) = x$) then $\delta \notin l_1$. Thus $\delta = \widehat{\beta'}$ for some free o-metavariable $\widehat{\beta'}$ or else the pre-metaterm D would not be a metaterm. So then $\widehat{\beta'}$ is a free o-metavariable in D and thus by the Hypothesis 1 $U_{(\theta_v,S,k)}(\kappa)$ is defined on $\widehat{\beta'}$. Now, the assignment θ_v satisfies the requirements of Definition 6.67, so that in particular by the second requirement we must have $\theta_v(\widehat{\beta'}) = x \in S \cup k$.

   We now analyse where the bound occurrence of x comes from in $\theta' M_1$ in order to arrive at a contradiction. Here too we have two cases to consider:

   (a) $x = \theta_v(\beta'')$ for some formal parameter β'' occuring in M_1. Now, $\theta_v(\beta'') \notin S \cup k$ since θ_v satisfies the requirement of Definition 6.67 by hypothesis, so that we arrive at a contradiction.

   (b) x comes from instantiating some t-metavariable Z in M_1, i.e. $x \in BV(\theta' Z)$ for some t-metavariable Z in M_1 (Figure A.3(b)). Thus there is a t-metavariable Z_l with $l = l'\alpha l_1$ in $Dom(\kappa)$, a simple label k' and an index m such that $U^S_{k'\theta_v(l'\alpha l_1)k}(m) = x = \mathtt{at}(k'\theta_v(l'\alpha l_1)k, m)$.

   Now since x is bound in $\theta' Z$ we have $m \leq |k'\theta_v(l')|$. But then by definition of $U^\bullet_\bullet(\bullet)$ we have $x \notin S \cup \theta_v(\alpha l_1)k$, in other words, x cannot have been used as a candidate variable for binding. In particular, $x \notin S \cup k$. This is a contradiction since we already know that $x \in S \cup k$.

2. There is an occurrence of a t-metavariable Y in M_2 such that $x \in FV(\theta' Y)$. Then there is an occurrence of Y_l in $Dom(\kappa)$ with $l = l_2 l_1$ such that $x \in FV(U^S_{\theta_v(l_2 l_1)k}(\kappa Y_l))$ where l_1 is the label "above" M_2 (Figure A.3(c)). Note that since for this occurrence of x we have $x \in FV(\theta' M_2)$ then we must have that $x \in S$ or $x \in \theta_v(l_1)k$.

   We now analyse where the bound occurrence of x comes from in $\theta' M_1$. Here too we have two cases to consider:

   (a) $x = \theta_v(\beta'')$ for some formal parameter β'' occurring in M_1. If $x \in S$ or $x \in k$ we arrive at a contradiction with the fact that θ_v verifies the requirements of Definition 6.67 (saying that $\theta_v(\beta'') \notin S \cup k$). Moreover, if $x \in \theta_v(l_1)$ we contradict Fact A.22.

   (b) x comes from instantiating some o-metavariable Z in M_1, i.e. $x \in BV(\theta' Z)$ for some t-metavariable Z in M_1 (Figure A.3(d)). Thus there is an o-metavariable Z_l in $Dom(\kappa)$ with $l = l'\alpha l_1$, a simple label k' and an index m such that $U^S_{k'\theta_v(l'\alpha l_1)k}(m) = x = \mathtt{at}(k'\theta_v(l'\alpha l_1)k, m)$.

   Now since x is bound in M_1 we have $m \leq |k'\theta_v(l')|$. But then by definition of $U^\bullet_\bullet(\bullet)$ we have $x \notin S \cup \theta_v(\alpha l_1)k$. In particular, $x \notin S \cup \theta_v(l_1)k$. This is a contradiction since we already know that $x \in S$ or $x \in \theta_v(l_1)k$.

∎

Lemma A.23 (From valid de Bruijn valuations to admissible valuations) Let κ be a valid de Bruijn valuation for a rewrite rule (L, R), θ_v a variable assignment verifying the hypothesis in Definition 6.67, and $U(L, R) = (G, D)$ the translation of (L, R). If the following conditions hold

1. $U_{(\theta_v,S,k)}(\kappa)$ is defined for *all* metavariables of G and D, and

2. θ_v is injective on the set of bound o-metavariables of (G, D).

then $U_{(\theta_v,S,k)}(\kappa)$ is admissible for (G, D).

Proof. We shall abbreviate $U_{(\theta_v,S,k)}(\kappa)$ by θ' in order to improve readability. Since by Lemma A.21 we have that θ' is safe then by Definition 6.22 we have still to check the following properties:

- θ verifies the path condition for X in (G, D): If no t-metavariable occurs more than once then the property is trivial so let us suppose that there is a t-metavariable X in (G, D) occuring at two different positions p and p'. Let us take any variable $x \in FV(\theta' X)$ and let l and l' be the parameter paths of p and p' in the trees corresponding to G or D. Suppose $\theta' X = U_{(\theta_v,S,k)}(\kappa)X \overset{\text{def}}{=} U^S_{\theta_v(l)k}(\kappa X_l)$. Then since κ is valid by Lemma 6.68 $U^S_{\theta_v(l)k}(\kappa X_l) =_\alpha U^S_{\theta_v(l')k}(\kappa X_{l'})$. As a consequence, the set of free variables of both terms is the same. Now, to show that θ verifies the path condition for X in (G, D) let us suppose that $x \in \theta_v(l)$. Since the o-metavariables in l are bound in the rule (G, D), and θ_v is defined for all the metavariables of (G, D) by Hypothesis 1, then by the requirements of Definition 6.67 $x \notin S \cup k$. Now, since x is free in $U^S_{\theta_v(l')k}(\kappa X_{l'})$ then x must be in $S \cup \theta_v(l')k$, which implies that x is necessarily in $\theta_v(l')$. This allows us to conclude that θ verifies the path condition for X in (G, D).

- if the pre-bound o-metavariables α and β occur in (G, D) with $\alpha \neq \beta$, then $\theta_v\alpha \neq \theta_v\beta$: this property trivially holds by Hypothesis 2.

A.3.3 Preserving Confluence

We start by a technical lemma that will be used later.

Lemma A.24 Let $M \in \mathcal{PMT}$ without occurrences of t-metavariables, and let

1. $k\alpha l$ be a simple label, k' a label such that $|k| = |k'|$, α' a pre-bound o-metavariable,

2. β a bound o-metavariable such that it does not occur in $U_{k\alpha l}(T_{k'\alpha' l}(M))$, and

3. $\mathcal{WF}_{k'\alpha' l}(M)$ hold.

Then $(U_{k\alpha l}(T_{k'\alpha' l}(M)))\ll\alpha\leftarrow\beta\gg =_v U_{k\beta l}(T_{k'\alpha' l}(M))$.

Proof. By induction on M. Let $k = \beta_1 \ldots \beta_n$ and $k' = \beta'_1 \ldots \beta'_n$. By Hypothesis 3 we have the following cases to consider:

- $M = \alpha'' \in k'$ and hence $\alpha'' = \beta'_j$ for some $1 \leq j \leq n$. Then we have
$$U_{k\alpha l}(T_{k'\alpha' l}(M))\ll\alpha\leftarrow\beta\gg = \beta_j\ll\alpha\leftarrow\beta\gg =_{hyp.1} \beta_j = U_{k\beta l}(T_{k'\alpha' l}(M))$$

- $M = \alpha'' \in l$ and $\alpha'' \notin k'$. Then we have
$$U_{k\alpha l}(T_{k'\alpha' l}(M))\ll\alpha\leftarrow\beta\gg = \alpha''\ll\alpha\leftarrow\beta\gg =_{hyp.1} \alpha'' = U_{k\beta l}(T_{k'\alpha' l}(M))$$

- $M = \alpha'$ and $\alpha' \notin k'$. Then we have
$$U_{k\alpha l}(T_{k'\alpha' l}(M))\ll\alpha\leftarrow\beta\gg = \alpha\ll\alpha\leftarrow\beta\gg = \beta = U_{k\beta l}(T_{k'\alpha' l}(M))$$

- $M = \widehat{\alpha}$. Then we have
$$U_{k\alpha l}(T_{k'\alpha' l}(M))\ll\alpha\leftarrow\beta\gg = \widehat{\alpha}\ll\alpha\leftarrow\beta\gg = \widehat{\alpha} = U_{k\beta l}(T_{k'\alpha' l}(M))$$

- $M = f(M_1, \ldots, M_n)$. Then we use the induction hypothesis.

- $M = \xi\alpha''.(M_1, \dots, M_n)$. We reason as follows:

$$(U_{k\alpha l}(T_{k'\alpha' l}(M)))\langle\!\langle\alpha\leftarrow\beta\rangle\!\rangle \quad = \quad (\xi\beta'.(U_{\beta' k\alpha l}(T_{\alpha'' k'\alpha' l}(M_1)), \dots, U_{\beta' k\alpha l}(T_{\alpha'' k'\alpha' l}(M_n))))\langle\!\langle\alpha\leftarrow\beta\rangle\!\rangle$$

for $\beta' \notin k\alpha l$ such that $\mathcal{WF}_{\beta' k\alpha l}(T_{\alpha'' k'\alpha' l}(M_i))$ holds for $1 \leq i \leq n$. Since $\beta \neq \beta'$ by hypothesis 2, continue

$$\begin{array}{ll} & (\xi\beta'.(U_{\beta' k\alpha l}(T_{\alpha'' k'\alpha' l}(M_1)), \dots, U_{\beta' k\alpha l}(T_{\alpha'' k'\alpha' l}(M_n))))\langle\!\langle\alpha\leftarrow\beta\rangle\!\rangle \\ = & \xi\beta'.((U_{\beta' k\alpha l}(T_{\alpha'' k'\alpha' l}(M_1)))\langle\!\langle\alpha\leftarrow\beta\rangle\!\rangle, \dots, (U_{\beta' k\alpha l}(T_{\alpha'' k'\alpha' l}(M_n)))\langle\!\langle\alpha\leftarrow\beta\rangle\!\rangle) \quad \text{by i.h.} \\ =_v & \xi\beta'.(U_{\beta' k\beta l}(T_{\alpha'' k'\alpha' l}(M_1)), \dots, U_{\beta' k\beta l}(T_{\alpha'' k'\alpha' l}(M_n))) \\ \overset{\text{def}}{=} & U_{k\beta l}(T_{k'\alpha' l}(\xi\alpha''.(M_1, \dots, M_n))) \end{array}$$

- $M = M_1[\alpha' \leftarrow M_2]$. Similar to the previous case.

Lemma A.25 Let $M \in \mathcal{PMT}$ and l a simple label. If $\mathcal{WF}_l(M)$ then $U_l(T_l(M)) =_v M$.

Proof. By induction on M.

- $M = \alpha$. Then since $\mathcal{WF}_l(M)$ we have $\alpha \in l$ and thus $U_l(T_l(\alpha)) = U_l(\text{pos}(\alpha, l)) = \alpha$.
- $M = \widehat{\alpha}$. Then $U_l(T_l(\widehat{\alpha})) = U_l(\mathsf{S}^{|l|}(\widehat{\alpha})) = \widehat{\alpha}$.
- $M = X$. Then $U_l(T_l(X)) = U_l(X_l) = X$.
- $M = f(M_1, \dots, M_n)$. We use the induction hypothesis.
- $M = \xi\alpha.(M_1, \dots, M_n)$. We reason as follows:

$$U_l(T_l(\xi\alpha.(M_1, \dots, M_n))) = U_l(\xi(T_{\alpha l}(M_1), \dots, T_{\alpha l}(M_n))) = \xi\beta.(U_{\beta l}(T_{\alpha l}(M_1)), \dots, U_{\beta l}(T_{\alpha l}(M_n)))$$

where $\beta \notin l$. We have two further cases to consider:

1. There are no occurrences of t-metavariables in M. Now if $\beta = \alpha$ we conclude by using the induction hypothesis so let us assume then that $\beta \neq \alpha$.

$$\begin{array}{lll} & \xi\beta.(U_{\beta l}(T_{\alpha l}(M_1)), \dots, U_{\beta l}(T_{\alpha l}(M_n))) & \\ =_v & \xi\beta'.(U_{\beta l}(T_{\alpha l}(M_1))\langle\!\langle\beta\leftarrow\beta'\rangle\!\rangle, \dots, U_{\beta l}(T_{\alpha l}(M_n))\langle\!\langle\beta\leftarrow\beta'\rangle\!\rangle) & (\beta' \text{ not in } U_{\beta l}(T_{\alpha l}(M_i))) \\ = & \xi\beta'.(U_{\beta' l}(T_{\alpha l}(M_1)), \dots, U_{\beta' l}(T_{\alpha l}(M_n))) & (\text{L.A.24}) \\ = & \xi\beta'.(U_{\alpha l}(T_{\alpha l}(M_1))\langle\!\langle\alpha\leftarrow\beta'\rangle\!\rangle, \dots, U_{\alpha l}(T_{\alpha l}(M_n))\langle\!\langle\alpha\leftarrow\beta'\rangle\!\rangle) & (\text{L.A.24}) \\ =_v & \xi\alpha.(U_{\alpha l}(T_{\alpha l}(M_1)), \dots, U_{\alpha l}(T_{\alpha l}(M_n))) & \\ =_v & \xi\alpha.(M_1, \dots, M_n) & (\text{i.h.}) \end{array}$$

2. There is an occurrence of a t-metavariable X in M. In this case since $U_l(T_l(M))$ is defined we observe that it must be that $\beta = \alpha$. Indeed, we have that $X_{l'\alpha l}$ occurs in $T_l(M)$ for some label l'. Hence when translating this metavariable to the de Bruijn setting we shall arrive at $U_{l''\beta l}(X_{l'\alpha l})$, which is defined only for $l''\beta l = l'\alpha l$. Therefore, $\beta = \alpha$ and we use the induction hypothesis.

- $M = M_1[\alpha \leftarrow M_2]$. We proceed as above.

Corollary A.26 Let $M \in \mathcal{PMT}$ such that $\mathcal{WF}(M)$. Then $U(T(M)) =_v M$.

Lemma A.27 Let $t \in \mathcal{T}$ such that $FV(t) \subseteq S \cup l$ for l any label and S a finite set of variables. Then $U_l^S(T_l(t)) =_\alpha t$.

Proof. By induction on the structure of t.

- $t = x$. Then there are two cases to consider:

- $x \in l$. Then $U_l^S(T_l(x)) = U_l^S(\mathtt{pos}(x,l)) = x$.
- $x \notin l$. Then $U_l^S(T_l(x)) = U_l^S(\mathcal{O}(x) + |l|)$. By hypothesis $x \in S \cup l$ so that $x \in S$ and then $U_l^S(\mathcal{O}(x) + |l|) = x_{\mathcal{O}(x)} = x$

- $t = f(t_1, \dots, t_n)$. Then $U_l^S(T_l(t)) = f(U_l^S(T_l(t_1)), \dots, U_l^S(T_l(t_n))) =_\alpha f(t_1, \dots, t_n)$. The last step holds by induction hypothesis.
- $t = \xi x.(t_1, \dots, t_n)$. Then

$$\begin{array}{lll}
 & U_l^S(T_l(\xi x.(t_1,\dots,t_n))) & \\
= & U_l^S(\xi(T_{xl}(t_1),\dots,T_{xl}(t_n))) & \\
= & \xi z.(U_{zl}^S(T_{xl}(t_1)),\dots,U_{zl}^S(T_{xl}(t_n))) & (z \notin S \cup l) \\
=_\alpha & \xi z'.(U_{zl}^S(T_{xl}(t_1))\{z \leftarrow z'\},\dots,U_{zl}^S(T_{xl}(t_n))\{z \leftarrow z'\}) & (z' \text{ not in } U_{z'l}^S(T_{xl}(t_i))) \\
= & \xi z'.(U_{z'l}^S(T_{xl}(t_1)),\dots,U_{z'l}^S(T_{xl}(t_n))) & (\text{L.A.18}) \\
= & \xi z'.(U_{xl}^S(T_{xl}(t_1))\{x \leftarrow z'\},\dots,U_{xl}^S(T_{xl}(t_n))\{x \leftarrow z'\}) & (\text{L.A.18}) \\
=_\alpha & \xi x.(U_{xl}^S(T_{xl}(t_1)),\dots,U_{xl}^S(T_{xl}(t_n))) & (x \notin l) \\
=_\alpha & \xi x.(t_1,\dots,t_n) & (\text{i.h.})
\end{array}$$

Corollary A.28 Let $t \in \mathcal{T}$. Then $U(T(t)) =_\alpha t$.

Lemma A.29 Let $A \in \mathcal{PMT}_{db}$ and l be any *simple* label. If $\mathcal{WF}_l(A)$ then $T_l(U_l(A)) = A$.

Proof. By induction on A.

- $A = \mathtt{S}^j(1)$. Then $T_l(U_l(A)) = T_l(\mathtt{at}(l, j+1)) = \mathtt{pos}(\mathtt{at}(l, j+1), l)$. Since l is simple then $\mathtt{pos}(\mathtt{at}(l, j+1), l) = \mathtt{S}^j(1)$ and we are done.
- $A = \mathtt{S}^j(\widehat{\alpha})$. Then $j = |l|$ and $T_l(U_l(A)) = T_l(\widehat{\alpha}) = \mathtt{S}^j(\widehat{\alpha})$.
- $A = X_k$. Then $l = k$ and $T_l(U_l(A)) = T_l(X) = X_k$.
- $A = f(A_1, \dots, A_n)$. Then

$$\begin{array}{rcl}
T_l(U_l(A)) & = & T_l(f(U_l(A_1),\dots,U_l(A_n))) \\
 & =_{i.h.} & f(T_l(U_l(A_1)),\dots,T_l(U_l(A_1))) \\
 & = & f(A_1,\dots,A_1)
\end{array}$$

- $A = \xi(A_1, \dots, A_n)$. Then

$$\begin{array}{rcl}
T_l(U_l(A)) & = & T_l(\xi\alpha.(U_{\alpha l}(A_1),\dots,U_{\alpha l}(A_n))) \\
 & = & \xi(T_{\alpha l}(U_{\alpha l}(A_1)),\dots,T_{\alpha l}(U_{\alpha l}(A_n))) \\
 & =_{i.h.} & \xi(A_1,\dots,A_n)
\end{array}$$

We remark that $\alpha \notin l$ by Definition 6.64 so the induction hypothesis can be applied.

- $A = A_1[\![A_2]\!]$. Then

$$\begin{array}{rcl}
T_l(U_l(A)) & = & T_l(U_{\alpha l}(A_1)[\alpha \leftarrow U_l(A_2)]) \\
T_l(U_l(A)) & = & T_{\alpha l}(U_{\alpha l}(A_1))[\![T_l(U_l(A_2))]\!] \\
T_l(U_l(A)) & =_{i.h.} & A_1[\![A_2]\!]
\end{array}$$

We remark that $\alpha \notin l$ by Definition 6.64 so the induction hypothesis can be applied.

Corollary A.30 Let $A \in \mathcal{PMT}_{db}$. If $\mathcal{WF}(A)$ then $T(U(A)) = A$.

Lemma A.31 Let a be a de Bruijn term and l be any *simple* label such that $\mathtt{Names}(FV(a) \setminus\!\!\setminus |l|) \subseteq S$ and $l \cap S = \emptyset$. Then $T_l(U_l^S(a)) = a$.

Proof. By induction on the structure of a.

- $a = n$. Then there are two cases to consider:
  - $n \leq |l|$. Then $T_l(U_l^S(a)) = T_l(\mathtt{at}(l,n)) = \mathtt{pos}(\mathtt{at}(l,n),l)$. Since l is simple, then $\mathtt{pos}(\mathtt{at}(l,n),l) = n$.
  - $n > |l|$ and $x_{n-|l|} \in S$. $T_l(U_l^S(a)) = T_l(x_{n-|l|})$. Since l does not contain variables in S, then $T_l(x_{n-|l|}) = n - |l| + |l| = n$.
- $a = f(a_1, \ldots, a_n)$. Then $T_l(U_l^S(a)) = f(T_l(U_l^S(a_1)), \ldots, T_l(U_l^S(a_n))) =_{i.h} f(a_1, \ldots, a_n)$.
- $a = \xi(a_1, \ldots, a_n)$. Then $T_l(U_l^S(a)) = \xi(T_{xl}(U_{xl}^S(a_1)), \ldots, T_{xl}(U_{xl}^S(a_1))) =_{i.h} \xi(a_1, \ldots, a_n)$. We remark that $x \notin l \cup S$ by Definition 6.63 so that xl is simple and it does not contain variables of S so that we can apply the induction hypothesis.

Corollary A.32 Let $a \in \mathcal{T}_{db}$. Then $T(U(a)) = a$.

A.4 From Higher-Order to First-Order Rewriting

A.4.1 On Pivot Selection

It is clear that $C_P(L,R)$ and $C_Q(L,R)$ shall not be identical. Nevertheless, the rewrite relation generated by both of these converted rewrite rules is identical.

Before proving this proposition, let us consider a rewrite rule (L,R) and let $X_{l_1}, \ldots, X_{l_n}$ be all the X-based metavariables in (L,R) with $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$. Let X_{l_1} and X_{l_2} be two possible X-based pivots for (L,R). Note that we must have either $X_{l_1}, X_{l_2} \in L$, or $X_{l_1}, X_{l_2} \in R$ (in which case $|k| > |l_1|$ and $|k| > |l_2|$ for all $X_k \in L$). Also, we have $|l_1| = |l_2|$, a fact that shall be made use of freely below.

Let us consider two different conversions (a) and (b) as dictated by Definition 7.30 taking any metavariable X_{l_i} for $1 \leq i \leq n$ and yielding a first-order term:

$$\begin{array}{lll} \text{(a)} & X_{l_i} \rightsquigarrow & X[\mathit{cons}(a_1^i, \ldots, a_{|l_1|}^i, \mathit{shift}^{|l_i|+|l_1 \backslash \mathtt{Ba}_{(L,R)}(X)|})] \\ & \text{and} & \\ \text{(b)} & X_{l_i} \rightsquigarrow & X[\mathit{cons}(b_1^i, \ldots, b_{|l_2|}^i, \mathit{shift}^{|l_i|+|l_2 \backslash \mathtt{Ba}_{(L,R)}(X)|})] \end{array}$$

Note that clause 1 of Definition 7.30 does not present itself since the case of interest is when $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$.

The first translation (a) corresponds to the conversion dictated assuming X_{l_1} as the pivot, while the second translation (b) assumes that X_{l_2} is the pivot.

On an informal account, the substitution $\mathit{cons}(a_1^i, \ldots, a_{|l_1|}^i, \mathit{shift}^{|l_i|+|l_1 \backslash \mathtt{Ba}_{(L,R)}(X)|})$ may be seen as representing a function f_i from indices to indices (hence assuming X is only instantiated with indices). Likewise, $\mathit{cons}(b_1^i, \ldots, b_{|l_2|}^i, \mathit{shift}^{|l_i|+|l_2 \backslash \mathtt{Ba}_{(L,R)}(X)|})$ represents a function g_i. We shall therefore be intersted in finding a function h which may be represented by a pattern substitution such that $f_i = g_i \circ h$. We shall see that the pattern substitution $\mathit{cons}(c_1, \ldots, c_{|l_1|}, \mathit{shift}^{|l_1|})$ defined below satisfies this requirement. Define the following indices c_j for all $1 \leq j \leq |l_1|$:

$$c_j = \begin{cases} a_j^2 & \text{if } \mathtt{at}(l_1, j) \in \mathtt{Ba}_{(L,R)}(X) \\ \mathtt{pos}(a_j^2, b_1^1 \ldots b_{|l_1|}^1) & \text{otherwise} \end{cases} \tag{A.1}$$

Remark A.33 Note that the second clause of the definition of c_j is defined. Indeed, if $\mathtt{at}(l_1, j) \notin \mathtt{Ba}_{(L,R)}(X)$ then $a_j^2 = |l_2| + 1 + \mathtt{Sh}(X_{l_1}, j)$ and since $|l_1| = |l_2|$, l_1 and l_2 are simple, and $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$ both l_1 and l_2 have the same number of o-metavariables not included in $\mathtt{Ba}_{(L,R)}(X)$. Thus there is a $j' \in 1..|l_1|$ such that $b_{j'}^1 = |l_1| + 1 + \mathtt{Sh}(X_{l_2}, j')$ with $\mathtt{Sh}(X_{l_2}, j') = \mathtt{Sh}(X_{l_1}, j)$, and hence $a_j^2 = b_{j'}^1$.

The relation between the two translations (a) and (b) given above can be summarized by the following result:

Lemma A.34 Let n be the number of X-based metavariables in (L, R) and let X_{l_1} and X_{l_2} be two distinct pivots for (L, R). Let $h \geq 0$ and $1 \leq i \leq n$. Take any assignment ρ and indices a^i_j ($1 \leq j \leq |l_1|$) and b^i_j ($1 \leq j \leq |l_2|$) as indicated above in the translations (a) and (b). Then

$$\begin{array}{ll} & (\rho X)[\mathit{lift}^h(\mathit{cons}(a^i_1, \ldots, a^i_{|l_1|}, \mathit{shift}^{|l_i|+|l_1\backslash \mathtt{Ba}_{(L,R)}(X)|}))] \\ =_{\mathcal{W}} & (\rho X)[\mathit{lift}^h(s)][\mathit{lift}^h(\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\backslash \mathtt{Ba}_{(L,R)}(X)|}))] \end{array}$$

where $s = \mathit{cons}(c_1, \ldots, c_{|l_1|}, \mathit{shift}^{|l_1|})$ defined in the equation A.1.

Proof. Note that we may assume that ρX is a pure term without loss of generality. We proceed by induction on ρX.

- $\rho X = j$. We consider three subcases:
  - $j \leq h$. Then Lemma 1 allows us to conclude this case.
  - $h < j \leq |l_1| + h$. Then

$$\begin{array}{lll} LHS & =_{\mathcal{W}} & j - h[\mathit{cons}(a^i_1, \ldots, a^i_{|l_1|}, \mathit{shift}^{|l_i|+|l_1\backslash \mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h =_{\mathcal{W}} a^i_{j-h} + h \\ RHS & =_{\mathcal{W}} & c_{j-h}[\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\backslash \mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \end{array}$$

    We shall consider two further cases. Recall that X_{l_1} is an X-based pivot for conversion (a) and X_{l_2} is an X-based pivot for conversion (b).

    1. $i = 1$. Suppose
       * $\mathtt{at}(l_1, j - h) = \beta \in \mathtt{Ba}_{(L,R)}(X)$. Then

$$\begin{array}{lll} RHS & = & a^2_{j-h}[\mathit{cons}(b^1_1, \ldots, b^1_{|l_2|}, \mathit{shift}^{|l_1|+|l_2\backslash \mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \\ & = & \mathtt{pos}(\beta, l_2)[\mathit{cons}(b^1_1, \ldots, b^1_{|l_2|}, \mathit{shift}^{|l_1|+|l_2\backslash \mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \\ & =_{\mathcal{W}} & b^1_{\mathtt{pos}(\beta,l_2)} + h \\ & = & \mathtt{pos}(\beta, l_1) + h \\ & = & j - h + h \\ & =_{\mathcal{W}} & LHS \end{array}$$

         Recall that all labels are simple (no repeated elements).
       * $\mathtt{at}(l_1, j - h) \notin \mathtt{Ba}_{(L,R)}(X)$.

$$\begin{array}{ll} & RHS \\ = & \mathtt{pos}(a^2_{j-h}, b^1_1..b^1_{|l_1|})[\mathit{cons}(b^1_1, \ldots, b^1_{|l_2|}, \mathit{shift}^{|l_1|+|l_2\backslash \mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \\ = & \mathtt{pos}(|l_2| + 1 + \mathtt{Sh}(X_{l_1}, j - h), b^1_1..b^1_{|l_1|})[\mathit{cons}(b^1_1, \ldots, b^1_{|l_2|}, \mathit{shift}^{|l_1|+|l_2\backslash \mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \\ =_{\mathcal{W}} & |l_2| + 1 + \mathtt{Sh}(X_{l_1}, j - h) + h \\ = & |l_1| + 1 + \mathtt{Sh}(X_{l_1}, j - h) + h \\ =_{\mathcal{W}} & LHS \end{array}$$

    2. $i \geq 2$. Suppose
       * $\mathtt{at}(l_1, j - h) = \beta \in \mathtt{Ba}_{(L,R)}(X)$. Then

$$\begin{array}{lll} RHS & = & a^2_{j-h}[\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\backslash \mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \\ & = & \mathtt{pos}(\beta, l_2)[\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\backslash \mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \\ & =_{\mathcal{W}} & b^i_{\mathtt{pos}(\beta,l_2)} + h \\ & = & \mathtt{pos}(\beta, l_i) + h \\ & =_{\mathcal{W}} & LHS \end{array}$$

         If $i = 2$ then the last step follows from the case 2(a) of Definition 7.30 (since X_{l_2} is a pivot metavariable occurrence for conversion (b)), otherwise it follows from case 2(b) of the same definition.

- $*$ $\mathtt{at}(l_1, j-h) \notin \mathtt{Ba}_{(L,R)}(X)$. Then $LHS =_{\mathcal{W}} |l_i| + 1 + \mathtt{Sh}(X_{l_1}, j-h) + h$. Also,

$$\begin{array}{ll} & RHS \\ = & \mathtt{pos}(a^2_{j-h}, b^1_1..b^1_{|l_1|})[\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\setminus\mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \\ = & \mathtt{pos}(|l_2| + 1 + \mathtt{Sh}(X_{l_1}, j-h), b^1_1..b^1_{|l_1|})[\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\setminus\mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \end{array}$$

Now since $|l_2| + 1 + \mathtt{Sh}(X_{l_1}, j-h) > |l_1|$ and $|l_1| = |l_2|$ there must be some $1 \leq j' \leq |l_1|$ such that $b^1_{j'} = |l_1| + 1 + \mathtt{Sh}(X_{l_2}, j')$ (and hence $\mathtt{at}(l_2, j') \notin \mathtt{Ba}_{(L,R)}(X)$) with $\mathtt{Sh}(X_{l_1}, j-h) = \mathtt{Sh}(X_{l_2}, j')$ (see Remark A.33). Thus we may continue as follows:

$$\begin{array}{ll} & \mathtt{pos}(|l_2| + 1 + \mathtt{Sh}(X_{l_1}, j-h), b^1_1..b^1_{|l_1|})[\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\setminus\mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \\ = & j'[\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\setminus\mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \\ =_{\mathcal{W}} & b^i_{j'} + h \\ = & |l_i| + 1 + \mathtt{Sh}(X_{l_2}, j') + h \end{array}$$

The last equality follows from the fact that $\mathtt{at}(l_2, j') \notin \mathtt{Ba}_{(L,R)}(X)$.

- $-$ $j > |l_1| + h$. Then

$$\begin{array}{ll} & LHS \\ =_{\mathcal{W}} & j - h[\mathit{cons}(a^i_1, \ldots, a^i_{|l_1|}, \mathit{shift}^{|l_i|+|l_1\setminus\mathtt{Ba}_{(L,R)}(X)|})][\mathit{shift}]^h \\ =_{\mathcal{W}} & j - h - |l_1| + |l_i| + |l_1 \setminus \mathtt{Ba}_{(L,R)}(X)| + h \\ = & j - |l_1| + |l_i| + |l_1 \setminus \mathtt{Ba}_{(L,R)}(X)| \\ = & j - |l_2| + |l_i| + |l_2 \setminus \mathtt{Ba}_{(L,R)}(X)| \\ =_{\mathcal{W}} & j[\mathit{lift}^h(\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\setminus\mathtt{Ba}_{(L,R)}(X)|}))] \\ =_{\mathcal{W}} & j - h[\mathit{cons}(c_1, \ldots, c_{|l_1|}, \mathit{shift}^{|l_1|})][\mathit{shift}]^h[\mathit{lift}^h(\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\setminus\mathtt{Ba}_{(L,R)}(X)|}))] \\ =_{\mathcal{W}} & RHS \end{array}$$

- $\rho X = f(d_1, \ldots, d_n)$. We use the induction hypothesis.
- $\rho X = \xi(d_1, \ldots, d_n)$. Then by the induction hypothesis we have

$$\begin{array}{ll} & d_j[\mathit{lift}^{h+1}(\mathit{cons}(a^i_1, \ldots, a^i_{|l_1|}, \mathit{shift}^{|l_i|+|l_1\setminus\mathtt{Ba}_{(L,R)}(X)|}))] \\ =_{\mathcal{W}} & d_j[\mathit{lift}^{h+1}(\mathit{cons}(c_1, \ldots, c_{|l_1|}, \mathit{shift}^{|l_1|}))][\mathit{lift}^{h+1}(\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\setminus\mathtt{Ba}_{(L,R)}(X)|}))] \end{array}$$

for all $j \in 1..n$ which allows us to conclude the case.

■

Proof.[of Proposition 7.37] Let $(L_1, R_1) \overset{\text{def}}{=} \mathcal{C}_P(L, R)$ and $(L_2, R_2) \overset{\text{def}}{=} \mathcal{C}_Q(L, R)$. Suppose $a \to_{(L_1,R_1)} b$. Then there exists a context E and an assignment ρ such that $a =_{\mathcal{W}} E[\rho(L_1)]$ and $b =_{\mathcal{W}} E[\rho(R_1)]$.

For all $X \in \mathit{NFMVar}(L)$ define the assignment η as: $\overline{\eta}X \overset{\text{def}}{=} \rho(X)[s]$ where $s = \mathit{cons}(c_1, \ldots, c_{|l_1|}, \mathit{shift}^{|l_1|})$ and the c_is are defined in equation A.1. Consider now an occurrence of a metavariable $X_{l_i} \in (L, R)$ where $\{X_{l_1}, \ldots, X_{l_n}\}$ are all the X-based metavariables in (L, R).

- If $\mathtt{Ba}_{(L,R)}(X) = \emptyset$ then both conversions shall convert X_{l_i} to the term $X[\mathit{shift}^{|l_i|}]$. This case needs no further consideration.
- If $\mathtt{Ba}_{(L,R)}(X) \neq \emptyset$ then each conversion shall convert X_{l_i} to (possibly) different terms: $X[\mathit{cons}(a^i_1, \ldots, a^i_{|l_1|}, \mathit{shift}^{|l_i|+|l_1\setminus\mathtt{Ba}_{(L,R)}(X)|})]$ on one hand, and $X[\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\setminus\mathtt{Ba}_{(L,R)}(X)|})]$, on the other. Here we may apply Lemma A.34 and obtain: $(\rho X)[\mathit{cons}(a^i_1, \ldots, a^i_{|l_1|}, \mathit{shift}^{|l_i|+|l_1\setminus\mathtt{Ba}_{(L,R)}(X)|})] =_{\mathcal{W}} (\eta X)[\mathit{cons}(b^i_1, \ldots, b^i_{|l_2|}, \mathit{shift}^{|l_i|+|l_2\setminus\mathtt{Ba}_{(L,R)}(X)|})]$.

  If conversion (a) deployed the identity optimization then

  $$\mathit{cons}(a^i_1, \ldots, a^i_{|l_1|}, \mathit{shift}^{|l_i|+|l_1\setminus\mathtt{Ba}_{(L,R)}(X)|}) = \mathit{cons}(1, \ldots, |l_1|, \mathit{shift}^{|l_1|})$$

  and X_{l_i} is converted to X and we may use the fact that $\rho X =_{\mathcal{W}} (\rho X)[\mathit{cons}(1, \ldots, |l_i|, \mathit{shift}^{|l_i|})]$ and Lemma A.34 as above. A similar observation holds for the (b) conversion.

Therefore we may obtain $\rho(L_1) =_{\mathcal{W}} \eta(L_2)$ and $\rho(R_1) =_{\mathcal{W}} \eta(R_2)$, so that $a =_{\mathcal{W}} E[\rho(L_1)] =_{\mathcal{W}} E[\eta(L_2)]$ and $b =_{\mathcal{W}} E[\rho(R_1)] =_{\mathcal{W}} E[\eta(R_2)]$, i.e. $a \to_{(L_2,R_2)} b$.

■

Bibliography

[ABR00] A. Arbiser, E. Bonelli, and A. Ríos. Zooming in on Explicit Substitutions. Technical Report 2000-009, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, 2000.

[AC96] M. Abadi and L. Cardelli. *A Theory of Objects.* Monographs in Computer Science. Springer-Verlag, 1996.

[ACCL91] M. Abadi, L. Cardelli, P-L. Curien, and J-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.

[Acz77] P. Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North Holland, 1977.

[AG00] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

[ARK00] M. Ayala-Rincón and F. Kamareddine. Unification via λse-style of Explicit Substitution. In *Proceedings of the Second International Conference on Principles and Practice of Declarative Programming.* ACM Press, September 2000.

[Bar84] H.P. Barendregt. *The Lambda Calculus: its Syntax and Semantics.* Studies in Logic and the Foundations of Mathematics 103. North-Holland, Amsterdam, revised edition, 1984.

[Bar92] H.P. Barendregt. Lambda Calculi with Types. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Oxford University Press, 1992.

[BBKV76] H.P. Barendregt, J.A. Bergstra, J.W. Klop, and H. Volken. Degrees, reductions and representability in the lambda calculus. Technical Report 22, Utrecht University, 1976.

[BBLRD96] Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. λv, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996.

[BD88] L. Bachmair and N. Dershowitz. Critical pair criteria for completion. *Journal of Symbolic Computation*, 6(1):1–18, 1988.

[BdV99] M. Bognar and R. de Vrijer. The context calculus lambda-c. In *Proceedings of the Workshop on Logical Frameworks and Meta-languages*, September 1999.

[Ber92] Y. Bertot. Origin Functions in Lambda Calculus and Term Rewriting Systems. In *Proceedings of 17th Colloquium on Trees in Algebra and Programming*, number 581 in LNCS, pages 49–65. Springer-Verlag, 1992.

[BG96] R. Bloo and H. Geuvers. Explicit substitution: on the edge of strong normalization. Technical Report TR96-10, TUE Computing Science Reports, Eindhoven University of Technology, 1996.

[BH98] R. Bloo and Geuvers H. Explicit substitution: on the edge of strong normalization. *Theoretical Computer Science*, 204, 1998.

[BK82] J.A. Bergstra and J.W. Klop. Strong normalization and perpetual reductions in the lambda calculus. *Journal of Information Processing and Cybernetics*, 18:403–417, 1982.

[BKR00] E. Bonelli, D. Kesner, and A. Ríos. A de Bruijn notation for Higher-Order Rewriting. In *Proceedings of the Eleventh International Conference on Rewriting Techniques and Applications*, number 1833 in LNCS, pages 62–79, Norwich, UK, July 2000. Springer-Verlag.

[BKR01] E. Bonelli, D. Kesner, and A. Ríos. From Higher-Order to First-Order Rewriting. In *Proceedings of the Twelth International Conference on Rewriting Techniques and Applications*, number 2051 in LNCS, pages 47–62, Utrecht, Holland, May 2001. Springer-Verlag.

[Blo95] R. Bloo. Preservation of strong normalization for explicit substitutions. Technical Report TR95-08, TUE Computing Science Reports, Eindhoven University of Technology, 1995.

[Blo97] R. Bloo. *Preservation of Termination for Explicit Substitutions.* PhD thesis, Eindhoven University, 1997.

[Blo99] R. Bloo. Pure type systems with explicit substitutions. In *Proceedings of the Second International Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs (FLoC'99 Workshop)*, Trento, Italy, 1999.

[Blo01] R. Bloo. Pure type systems with explicit substitutions. *Mathematical Structures in Computer Science*, 11(1), 2001.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[Bon99a] E. Bonelli. A Polymorphic Lambda Calculus with Explicit Substitutions. In *Proceedings of the Second International Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs (FLoC'99 Workshop)*, Trento, Italy, 1999.

[Bon99b] E. Bonelli. Using Fields and Explicit Substitutions to Implement Objects and Functions in a de Bruijn Setting. In *Proceedings of the Thirteenth International Conference CSL*, Madrid, Spain, September 1999.

[Bon01] E. Bonelli. Perpetuality in a Named Lambda Calculus with Explicit Substitutions. *Mathematical Structures in Computer Science*, 11(1), 2001.

[Bou85] G. Boudol. Computational semantics of term rewrite systems. In M. Nivat and J.C. Reynolds, editors, *Algebraic methods in Semantics.* Cambridge University Press, 1985.

[BR96] R. Bloo and K. Rose. Combinatory Reduction Systems with explicit substitution that preserve strong normalization. In *Proceedings of the International Conference on Rewriting Techniques and Applications*, number 1103 in LNCS. Springer-Verlag, 1996.

[Bri95] D. Briaud. An explicit eta rewrite rule. In M. Dezani, editor, *Proceedings of International Conference on Typed Lambda Calculus and Applications*, number 902 in LNCS. Springer-Verlag, 1995.

[Bru72] N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation with application to the church-rosser theorem. *Indag. Mat.*, 5(35), 1972.

[Bru78] N.G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. Technical Report 78-WSK-03, Eindhoven University of Technology, 1978.

[CF58] H.B. Curry and R. Feys. *Combinatory Logic*, volume 1. North Holland, 1958.

[CHL96] P-L. Curien, T. Hardin, and J-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, 1996.

[Chu32] A. Church. A set of postulates for the foundation of logic (1). *Annals of Mathematics*, 33:346–366, 1932.

[Chu41] A. Church. *The Calculi of Lambda-Conversion.* Princeton University Press, 1941.

[CK96] D. Clark and R. Kennaway. Event structures and non-orthogonal term graph rewriting. *Mathematical Structures in Computer Science*, 629:209–217, 1996.

[CR36] A. Church and J.B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39:472–482, 1936.

[Cur86] P-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Pitman, 1986.

[Cur93] P-L. Curien. *Categorical combinators, sequential algorithms and functional programming*. Progress in Theoretical Computer Science. Birkhaüser, 1993.

[Der82] N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.

[DG99] R. David and B. Guillaume. A λ-calculus with explicit weakening and explicit substitutions. In *Proceedings of the Second International Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs (FLoC'99 Workshop)*, 1999.

[DG01] R. David and B. Guillaume. A λ-calculus with explicit weakening and explicit substitutions. *Mathematical Structures in Computer Science*, 11(1), 2001.

[DHK95] G. Dowek, Th. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. In Kozen D., editor, *LICS'95*, pages 366–374, 1995.

[DHK98] G. Dowek, Th. Hardin, and C. Kirchner. Theorem proving modulo. Technical Report RR 3400, INRIA, 1998.

[DHK00] G. Dowek, Th. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. *Information and Computation*, 157:183–235, 2000.

[DHK01] G. Dowek, Th. Hardin, and C. Kirchner. Hol-lambda-sigma: an intentional first-order expression of higher-order logic. *Mathematical Structures in Computer Science*, 11:1–25, 2001.

[DHKP98] G. Dowek, Th. Hardin, C. Kirchner, and F. Pfenning. Unification via explicit substitutions: The case of higher-order patterns. Technical Report RR3591, INRIA, December 1998.

[DJ90] N. Dershowitz and J-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.

[DL01] D. Dougherty and P. Lescanne. Reductions, intersection types and explicit substitutions. In *Proceedings of TLCA*, number 2044 in LNCS. Springer-Verlag, 2001.

[FHM94] K. Fisher, F. Honsell, and J.C. Mitchell. A lambda calculus of objects and method specialization. *Nordic Journal of Computing*, 1(1):3–37, 1994.

[Gal90] J.H. Gallier. On Girard's "candidats de reducibilité". In P. Odifreddi, editor, *Logic in Computer Science*, number 31 in APIC, pages 123–203. Academic Press, 1990.

[GGL00] H. Goguen and J. Goubault-Larrecq. Sequent combinators: a Hilbert system for the lambda calculus. *Mathematical Structures in Computer Science*, 10:1–79, 2000.

[Gir72] J-Y. Girard. *Interprétation Fonctionnelle et Elimination des Coupures de l'Arithmétique d'Ordre Supérieur*. PhD thesis, Université Paris VII, 1972. Thèse d'Etat.

[GJ85] J.A. Goguen and Meseguer J. Completeness of many sorted equational deduction. *Houston Journal of Mathematics*, 11(3):307–334, 1985.

[GKK00] J. Glauert, R. Kennaway, and Z. Khasidashvili. Stable results and relative normalization. *Journal of Logic and Computation*, 10(3), 2000.

[GLM92] G. Gonthier, J-J. Lévy, and P-A Melliès. An abstract standardization theorem. In *7th Annual IEEE Symposium on Logic in Computer Science*, 1992.

[GLT89] J-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

[Gra96] B. Gramlich. *Termination and Confluence Properties of Structured Rewrite Systems*. PhD thesis, Fachbereich Informatik der Universität Kaiserslautern, 1996.

[Gui99] B. Guillaume. *Un calcul des substitutions avec etiquettes*. PhD thesis, Université de Savoie, Chambéry, 1999.

[Har87] Th. Hardin. *Résultats de confluence pour les règles fortes de la logique combinatoire catégorique et liens avec les lambda-calculs*. Thèse de doctorat, Université de Paris VII, 1987.

[Har89] T. Hardin. Confluence results for the strong pure categorical logic CCL; λ-calculi as subsystems of CCL. *Theoretical Computer Science*, 65, 1989.

[Har92] Th. Hardin. Eta-reduction for explicit substitutions. In *Proceedings of the 3rd International Conference on Algebraic and Logic Programming*, number 632 in LNCS. Springer-Verlag, 1992.

[Her00] H. Herbelin. Explicit substitutions and reducibility. In *Proceedings of WESTAPP'00*, 2000.

[Hin78] J.R. Hindley. Reductions of residuals are finite. *Transaction of the AMS*, 240, 1978.

[HL79] G. Huet and J-J. Lévy. Call by need computations in non-ambiguous linear term rewriting systems. Technical Report RR 359, INRIA, 1979.

[HL89] Th. Hardin and J-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, 1989.

[HL91] G. Huet and J-J. Lévy. Computatinal Logic; Essays in honor of Alan Robinson. In J.L. Lassez and G.D. Plotkin, editors, *Computations in orthogonal rewriting systems*, pages 394–443. MIT Press, 1991.

[Hue80] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery*, 27(4):797–821, 1980.

[HW88] P. Hudak and P. (editors) Wadler. Report on the functional programming language Haskell: draft proposed standard. Technical Report YALEU/DCS/RR-RR666, Yale University, 1988.

[JR99] J-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, page ?, Trento, Italy, 1999.

[Kes96] D. Kesner. Confluence properties of extensional and non-extensional lambda calculi with explicit substitutions. In *Proceedings of the International Conference on Rewriting Techniques and Applications*, number 1103 in LNCS, pages 184–199. Springer-Verlag, 1996.

[Kes00] D. Kesner. Confluence of extensional and non-extensional lambda-calculi with explicit substitutions. *Theoretical Computer Science*, 238(1-2):183–220, 2000.

[Kha90] Z. Khasidashvili. Expression Reduction Systems. In *Proceedings of I. Vekua Institute of Applied Mathematics*, volume 36, Tbilisi, 1990.

[Kha93] Z. Khasidashvili. Optimal Normalisation in Oorthogonal Term Rewrite Systems. In *Proceedings of the 5th. International Conference on Rewriting Techniques and Applications*, number 690 in LNCS, pages 243–258. Springer-Verlag, 1993.

[Kha94] Z. Khasidashvili. The longest perpetual reductions in orthogonal Expression Reduction Systems. In *Proceedings of the 3rd. International Conference on Logical Foundations of Computer Science*, number 813 in LNCS. Springer-Verlag, 1994.

[Kha01] Z. Khasidashvili. On Longest Perpetual Reductions in Orthogonal Expression Reduction Systems. *Theoretical Computer Science*, to appear, 2001.

[KKSdV95] J.R. Kennaway, J.W. Klop, M.R. Sleep, and F.J. de Vries. Infinitary Lambda Calculus and Böhm Models. In *Proceedings of the International Conference on Rewriting Techniques and Applications*, number 914 in LNCS, pages 257–270. Springer-Verlag, 1995.

[KL80] S. Kamin and J-J. Lévy. Attempts for generalizing the recursive path orderings. University of Illinois, 1980.

[Klo80] J.W. Klop. *Combinatory Reduction Systems*. PhD thesis, CWI, Amsterdam, 1980. Mathematical Centre Tracts n.127.

[Klo92] J.W. Klop. Term rewriting systems. *Handbook of Logic in Computer Science*, 2:1–116, 1992.

[KML98] D. Kesner and P.E. Martínez López. Explicit Substitutions for Objects and Functions. In *Proceedings of the Joint International Symposiums: Programming Languages, Implementations, Logics and Program (PLILP'98) and Algebraic and Logic Programming (ALP)*, number 1490 in LNCS. Springer-Verlag, 1998.

[KOO01a] Z. Khasidashvili, M. Ogawa, and V. van Oostrom. Perpetuality and uniform normalization in orthogonal rewrite systems. *Information and Computation*, 164:118–151, 2001.

[KOO01b] Z. Khasidashvili, M. Ogawa, and V. van Oostrom. Uniform Normalization Beyond Orthogonality. In *Proceedings of the 12th. International Conference on Rewriting Techniques and Applications*, number 2051 in LNCS, pages 122–136. Springer-Verlag, 2001.

[KOvR93] J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory Reduction Systems: introduction and survey. *Theoretical Computer Science*, 121(1–2):279–308, 1993.

[KR95] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. In *Proceedings of the International Symposium on Programming Language Implementation and Logic Programming (PLILP)*, number 982 in LNCS. Springer Verlag, 1995.

[KR97] F. Kamareddine and A. Ríos. Extending a λ-calculus with explicit substitutions which preserves strong normalization into a confluent calculus on open terms. *Journal of Functional Programming*, 7(4), 1997.

[KR98] F. Kamareddine and A. Ríos. Bridging de Bruijn indices and variable names in explicit substitutions calculi. *Logic Journal of the Interest Group of Pure and Applied Logic (IGPL)*, 6(6):843–874, 1998.

[KR00] F. Kamareddine and A. Ríos. Relating the $\lambda\sigma$- and λs-Styles of Explicit Substitutions. *Journal of Logic and Computation*, 10(3):349–380, 2000.

[Kri90] J-L. Krivine. *Lambda Calcul*. Études et Recherches en Informatique. Masson, 1990.

[Ler76] B. Lercher. Lambda calculus terms that reduce to themselves. *Nôtre Dame Journal of Formal Logic*, XVII(2):291-292, 1976.

[Les94] P. Lescanne. From λ_σ to λ_υ, a journey through calculi of explicit substitutions. In *Proceedings of Ann. ACM Symp. on Principles of Programming Languages (POPL)*, number 1490 in LNCS, pages 60–69. ACM, 1994.

[Lév78] J-J. Lévy. *Réductions correctes et optimales dans le lambda-calcul*. PhD thesis, Université Paris VII, 1978.

[LLL98] F. Lang, P. Lescanne, and L. Liquori. A framework for defining object calculi. Technical Report RR 1998-51, LIP, École Normale Supérieure de Lyon, 1998.

[LRD95] P. Lescanne and J. Rouyer-Degli. Explicit substitutions with de Bruijn levels. In *Proceedings of the 6th Conference on Rewriting Techniques and Applications*, number 914 in LNCS. Springer-Verlag, 1995.

[Mel95] P-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate. In *Proceedings of Typed Lambda Calculi and Applications*, number 902 in LNCS. Springer-Verlag, 1995.

[Mel96] P-A. Melliès. *Description Abstraite des Systèmes de Réécriture*. PhD thesis, Université Paris VII, 1996.

[Mel00] P-A. Melliès. Axiomatic rewriting theory II: The $\lambda\sigma$-calculus enjoys finite normalisation cones. *Journal of Logic and Computation*, 10(3):461–487, 2000.

[Mel01] P-A. Melliès. Axiomatic Rewriting Theory I: An axiomatic standardization theorem, 2001. Submitted for publication.

[MTH90] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. MIT Press, 1990.

[Muñ96] C. Muñoz. Confiuence and preservation of strong normalisation in an explicit substitutions calculus. In *Proceedings of the Eleven Annual IEEE Symposium on Logic in Computer Science*, 1996.

[Muñ97a] C. Muñoz. A left-linear variant of $\lambda\sigma$. In Michael Hanus, J. Heering, and K. (Karl) Meinke, editors, *Proceedings of the 6th International Conference on Algebraic and Logic Programming (ALP'97)*, number 1298 in LNCS. Springer-Verlag, September 1997.

[Muñ97b] C. Muñoz. *Un calcul de substitutions pour la représentation de preuves partielles en théorie de types*. PhD thesis, Université Paris VII, 1997.

[Nip91] T. Nipkow. Higher-order critical pairs. In *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, July 1991.

[NW63] C.St.J.A. Nash-Williams. On well-quasi-ordering finite trees. *Proceedings of the Cambridge Philosophical Society*, 59(4):833–835, 1963.

[O'D77] M. O'Donnell. Computing in Systems Described by Equations. Number 58 in LNCS. Springer-Verlag, 1977.

[Oos94] V. van Oostrom. *Confluence for Abstract and Higher-order Rewriting*. PhD thesis, Vrije University, 1994.

[Oos96] V. van Oostrom. Higher-order families. In *Proceedings of the Seventh International Conference on Rewriting Techniques and Applications*, number 1103 in LNCS. Springer-Verlag, 1996.

[Oos97] V. van Oostrom. Finite family developments. In *Proceedings of the Eighth International Conference on Rewriting Techniques and Applications*, 1997.

[Oos98] V. van Oostrom. Eventually Increasing, 1998. Obtainable by ftp at `ftp://www.phil.uu.nl/õostrom/publication/rewriting.html`.

[OR93] V. van Oostrom and F. van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. In J. Heering, K. Meinke, B. Möller, and T. Nopkow, editors, *Proc. of the First International Workshop Higher-Order Algebra, Logic and Term Rewriting*, number 816 in LNCS. Springer-Verlag, 1993.

[OR94] V. van Oostrom and F. van Raamsdonk. Weak orthogonality implies confluence: the higher-order case. In Nerode A. and Yu.V. Matiyasevich, editors, *Proceedings of the Third International Symposium on Logical Foundations of Computer Science*, number 813 in LNCS, pages 379–392. Springer-Verlag, 1994.

[Pag98] B. Pagano. *Des calculs de substitution explicit et leur application à la compilation des langages fonctionnels*. PhD thesis, Université de Paris VII, 1998.

[Pol93] R. Pollack. Closure under alpha-conversion. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs (TYPES)*, number 806 in LNCS, Nijmegen, The Netherlands, 1993. Springer-Verlag.

[Raa96] F. van Raamsdonk. *Confluence and Normalization for Higher-Order Rewriting*. PhD thesis, Amsterdam University, 1996.

[Río93] A. Ríos. *Contribution à l'étude des λ-calculs avec substitutions explicites*. PhD thesis, Université de Paris VII, 1993.

[Rit93] E. Ritter. Normalization for typed lambda calculi with explicit substitutions. In *Proceedings of CSL*, 1993.

[Rit99] E. Ritter. Characterising Explicit Substitutions which Preserve Termination. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, 1999.

[Ros92] K.H. Rose. Explicit cyclic substitutions. In *Proceedings of CTRS*, number 656 in LNCS. Springer-Verlag, 1992.

[RS94] J.J. Rehof and M.H. Sørensen. The lambda delta calculus. In *Proceedings of the International Conference on Theoretical Aspects of Computer Science*, number 789 in LNCS, pages 516–542. Springer-Verlag, 1994.

[RS95] F. van Raamsdonk and P. Severi. On Normalization. Technical Report CS-R9545, CWI, 1995.

[RSSX99] F. van Raamsdonk, P. Severi, M.H. Sørensen, and H. Xi. Perpetual Reductions in Lambda Calculus. *Journal of Information and Computation*, 149(2), 1999.

[Sør96] M.H. Sørensen. Effective longest and infinite reduction paths in untyped λ-calculus. In H. Kirchner, editor, *Proceedings of the 19th Internatinal Colloqium on Trees in Algebra and Programming*, number 1059 in LNCS. Springer-Verlag, 1996.

[Ste00] M-O. Stehr. CINNI - a Generic Calculus of Explicit Substitutions and its Application to λ-,ς- and π-Calculi. In *Proceedings of the 3rd International Workshop on Rewriting Logic and its Applications*, number ? in Electronic Notes in Computer Science. Elsevier, 2000.

[Suz96] T. Suzuki. Standardization theorem revisited. In *Proceedings of the 5th International Conference on Algebraic and Logic Programming*, number ? in LNCS. Springer-Verlag, 1996.

[Tai75] W.W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Proceedings of the Logic Colloquium*, number 453 in LNCS. Springer-Verlag, 1975.

[Tak89] M. Takahashi. Parallel reduction in the λ-calculus. *Journal of Symbolic Computation*, 7:113–123, 1989.

[WL93] P. Weis and X. Leroy. *Le langage Caml*. InterEditions, 1993.

[WM00] J. Wells and R. Muller. Standardization and evaluation in combinatory reduction systems, 2000. Working paper.

[Wol93] D. Wolfram. The Clausul Theory of Types. In *Cambridge tracts in Theoretical Computer Science*, volume 21. Cambridge University Press, 1993.

[Xi96] H. Xi. An induction measure on λ-terms and its applications. Technical Report 96-192, Department of Mathematical Sciences, Carnegie Mellon University, 1996.

[Zan95] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24, 1995.

Index