UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Semántica dinámica de cálculos de sustituciones explícitas a distancia

Tesis presentada para optar al título de
Doctor de la Universidad de Buenos Aires
en el área Ciencias de la Computación

Pablo Barenbaum

Lugar de trabajo:	Departamento de Computación Facultad de Ciencias Exactas y Naturales Universidad de Buenos Aires
Directores de tesis:	Eduardo Bonelli Delia Kesner
Consejero de estudios:	Alejandro Ríos
Fecha de defensa:	20 de noviembre de 2020

Buenos Aires, 2020

Semántica dinámica de cálculos de sustituciones explícitas a distancia

Los cálculos de sustituciones explícitas son variantes del cálculo-λ en los que la operación de sustitución no se define a nivel del metalenguaje, sino con reglas de reescritura que la implementan. Nuestro principal objeto de estudio es un cálculo de sustituciones explícitas particular, el Linear Substitution Calculus (LSC), definido por Accattoli y Kesner en 2010. Se caracteriza por el hecho de que las reglas de reescritura operan no localmente (*a distancia*). En esta tesis, en primer lugar, definimos máquinas abstractas que implementan estrategias de evaluación en el LSC: call-by-name para evaluación débil y fuerte, call-by-value y call-by-need. Demostramos que dichas máquinas son correctas y preservan la complejidad temporal. En segundo lugar, definimos una extensión de la estrategia de evaluación call-by-need en el LSC para evaluación fuerte. Demostramos que la estrategia es completa con respecto a call-by-name, usando un sistema de tipos intersección no idempotente, y mostramos cómo extenderla para lidiar con *pattern matching* y recursión. Por último, estudiamos la teoría de residuos y familias de radicales en el LSC. Para ello definimos una variante del LSC con etiquetas de Lévy, lo que nos permite demostrar que cumple con la propiedad de *Finite Family Developments*. Aplicamos esta propiedad para obtener resultados de optimalidad, estandarización y normalización de estrategias en el LSC, y generalizamos algunos de estos resultados al marco axiomático de *Deterministic Family Structures*.

Palabras clave: semántica de lenguajes de programación, cálculo-λ, sustituciones explícitas, estrategias de evaluación, evaluación lazy, máquinas abstractas, sistemas de tipos, teoría de residuos.

Dynamic Semantics of Calculi with Explicit Substitutions at a Distance

Explicit substitution calculi are variants of the λ-calculus in which the operation of substitution is not defined at the metalanguage level, but rather implemented by means of rewriting rules. Our main object of study is a particular explicit substitution calculus, the Linear Substitution Calculus (LSC), introduced by Accattoli and Kesner in 2010. Its distinguishing feature is that rewriting rules operate non-locally (*at a distance*). In this thesis, first, we define abstract machines to implement evaluation strategies in the LSC: call-by-name for weak and strong evaluation, call-by-value, and call-by-need. We prove that these machines are correct and that they preserve computational time complexity. Second, we define an extension of the call-by-need evaluation strategy in the LSC for strong reduction. We show that the strong call-by-need strategy is complete with respect to call-by-name, using a non-idempotent intersection type system, and we show how to extend the strategy to deal with pattern matching and recursion. Finally, we study the theory of residuals and redex families in the LSC. To this aim, we define a variant of the LSC endowed with Lévy labels, which allows us to prove that it enjoys the *Finite Family Developments* property. We apply this property to obtain results on optimality, standardization, and normalization for the LSC, and we generalize some of this results to the axiomatic framework of Deterministic Family Structures.

Keywords: programming language semantics, λ-calculus, explicit substitutions, evaluation strategies, lazy evaluation, abstract machines, type systems, residual theory.

Agradecimientos

A mis directores, Eduardo y Delia, por su paciencia y apoyo, por el entusiasmo, la pasión y la dedicación admirables que tienen por su profesión, y por mostrarme que la investigación consiste en mucho más que en el aspecto puramente técnico. A mi consejero de estudios, Alejandro Ríos, por presentarme formalmente al cálculo-λ y por aliviar la burocracia.

A los miembros del jurado, Nazareno Aguirre, Zena Ariola y Alexandre Miquel por aceptar evaluar esta tesis.

A Beniamino, Damiano y Thibaut, con quienes tuve la fortuna de colaborar en los inicios de la tesis, por impartirme sus implacables metodologías de trabajo. A mis estudiantes, Gonzalo, Fede, Mariana y Teo, por ayudarme en el experimento de reproducir algunas de estas metodologías.

A todos los miembros de LoReL. Especialmente a Jano por su energía, y a Andrés, sin cuya ayuda no hubiera podido ni empezar el doctorado ni sobrevivirlo.

A las instituciones en las que ejercí tareas de docencia e investigación: el Departamento de Computación y la Universidad Nacional de Quilmes. A las instituciones que me recibieron: la Université Paris 7 y el Stevens Institute of Technology. A las instituciones que otorgaron financiamiento para estudios y viáticos: CONICET, LIA INFINIS/SINFIN, Ministerio de Educación, UNQ y MinCyT a través de proyectos de investigación. A las personas que integran dichas instituciones: investigadores, docentes, estudiantes, autoridades, personal administrativo, personal no docente, etc.

A las personas con las que interactué durante estos años: docentes y estudiantes de la UBA y la UNQ, miembros del Departamento de Computación, miembros de PPS, integrantes de "la banda lambda" y a mis *mejores amigos*, por conformar un entorno que estimula la creatividad, la discusión racional y el cultivo de uno mismo.

A mi familia, en la que la lógica y los lenguajes siempre tuvieron un lugar central, por construirme y por su apoyo incondicional.

A Elisa por ser una hipótesis tácita.

Índice general

1. Introducción **7**

1.0.1. Destilación de máquinas abstractas 8
1.0.2. Fundamentos de call-by-need fuerte 9
1.0.3. Call-by-need fuerte para patrones y recursión 10
1.0.4. Etiquetas de Lévy para el LSC 11
1.0.5. Aplicaciones del LSC etiquetado 12
1.0.6. Publicaciones y trabajo no incluido en esta tesis 12

2. Destilación de máquinas abstractas **14**

2.1. Introducción 14
2.2. Estrategias de reducción 15
2.2.1. Call-by-name fuerte 16
2.2.2. Determinismo 18
2.3. Equivalencia estructural 18
2.4. Destilerías 20
2.4.1. Destilerías reflectivas 20
2.5. Máquinas abstractas 21
2.5.1. Call-by-name: la KAM 21
2.5.2. Call-by-name con entorno global: la MAM 23
2.5.3. Call-by-value de izquierda a derecha: la CEK 24
2.5.4. Call-by-value de izquierda a derecha: la CEK dividida 26
2.5.5. Call-by-value de derecha a izquierda: la LAM 27
2.5.6. Call-by-need: la MAD 28
2.5.7. Call-by-need: la MAD fusionada 30
2.5.8. Call-by-need: la MAD con punteros 31
2.5.9. Call-by-name fuerte: la MAM fuerte 33
2.6. Análisis de complejidad 36
2.6.1. Call-by-name and call-by-value 37
2.6.2. Call-by-need 37
2.6.3. Call-by-name fuerte 38

3. Fundamentos de call-by-need fuerte **39**

3.1. Introducción 39
3.2. Call-by-need fuerte 40

3.2.1. La teoría de *sharing* . . . 40
3.2.2. La estrategia call-by-need fuerte . . . 41
3.2.3. Propiedades básicas del call-by-need fuerte . . . 43
3.3. Completitud del call-by-need fuerte . . . 46
3.3.1. El sistema de tipos intersección no idempotente $\mathcal{HW}$ . . . 47
3.3.2. Completitud de la teoría de *sharing* . . . 52
3.3.3. Factorización de la teoría de *sharing* . . . 52

4. Call-by-need fuerte para patrones y recursión **56**
4.1. Introducción . . . 56
4.2. Extensión de la teoría de *sharing* . . . 56
4.2.1. El cálculo-λ extendido . . . 57
4.2.2. La teoría de *sharing* extendida . . . 57
4.3. Extensión del sistema de tipos . . . 59
4.3.1. El sistema de tipos intersección no idempotente extendido . . . 59
4.3.2. Caracterización de términos débilmente normalizantes . . . 60
4.4. Extensión de la estrategia call-by-need fuerte . . . 62
4.4.1. La estrategia call-by-need fuerte extendida . . . 62

5. Etiquetas de Lévy para el LSC **65**
5.1. Introducción . . . 65
5.2. El LSC con etiquetas de Lévy . . . 65
5.2.1. Teoría de residuos para el LSC . . . 66
5.2.2. Definición del LSC etiquetado sin gc . . . 68
5.2.3. Definición del LSC etiquetado – extensión con gc . . . 69
5.3. Propiedades del LSC con etiquetas de Lévy . . . 71
5.3.1. Propiedades básicas . . . 71
5.3.2. Ortogonalidad . . . 76
5.3.3. Normalización débil de la reducción acotada . . . 77
5.3.4. Normalización fuerte de la reducción acotada . . . 79
5.3.5. Confluencia . . . 81

6. Aplicaciones del LSC etiquetado **82**
6.1. Introducción . . . 82
6.2. Estabilidad . . . 83
6.3. Familias de redexes . . . 83
6.4. Reducción optimal . . . 86
6.5. Estandarización . . . 91
6.6. Normalización de estrategias . . . 95

7. Conclusión **100**
7.1. Una máquina abstracta para reducción call-by-need fuerte . . . 100
7.2. Definición de un procedimiento de extracción . . . 102

Capítulo 1

Introducción

La computación se basa en resolver problemas manipulando representaciones abstractas de la realidad. Por ejemplo, usamos los dedos para contar objetos, usando cada dedo para denotar un objeto. Esta representación es *abstracta* en el sentido de que descarta las características que resultan irrelevantes, tales como el tamaño o el color de los objetos, y preserva sólo sus aspectos relevantes: en este caso, la cantidad.

La computación no está ligada inseparablemente a las computadoras digitales modernas. Si bien estas son herramientas invaluables para implementar procesos computacionales, la computación como disciplina estudia primordialmente los principios subyacentes que gobiernan la manipulación mecánica de representaciones, independientemente de su potencial implementación. Los babilonios, por ejemplo, desarrollaron algoritmos para resolver ecuaciones casi 4000 años antes del advenimiento de las computadoras digitales [25].

Pero, lejos de ser una empresa meramente teórica, la computación tiene grandes consecuencias prácticas. En nuestros tiempos, el *software* está presente en todos los aspectos de nuestras sociedades, e impacta no sólo en asuntos personales como el entretenimiento y la comunicación, sino también en asuntos públicos como la transmisión de noticias, transacciones monetarias y predicción del clima, en sistemas de naturaleza crítica como equipamiento médico, reactores nucleares y sistemas de aviación, y en tecnologías emergentes sensibles tales como vehículos autónomos y monedas digitales.

En contraste con la visión tradicional de los algoritmos como meras relaciones funcionales entre datos de entrada y datos de salida, los procesos computacionales exhiben comportamientos complejos y generalmente muchas de sus propiedades, y no sólo la salida, tienen importancia práctica. Por ejemplo, ¿cómo se puede asegurar que un proceso computacional no consumirá demasiados recursos para poder llevar a cabo su trabajo? ¿Cómo se puede evitar que un tercero interactúe con un sistema y consiga que se comporte de manera maliciosa? ¿Es posible diseñar lenguajes de programación en los cuales los programas se parezcan más a especificaciones declarativas de los problemas y menos a algoritmos concretos, pero de tal modo que la ejecución sea eficiente?

Desarrollar métodos para responder estas preguntas satisfactoriamente y asistir así al desarrollo de programas correctos tiene una gran importancia, si se considera el papel crítico que desempeña el *software*. En las últimas décadas, se ha desarrollado un vasto repertorio de méto-

dos formales, incluyendo lenguajes de especificación formal, demostradores automáticos de teoremas, analizadores y sintetizadores de programas y técnicas de verificación como el análisis de flujo de datos o la interpretación abstracta, entre muchos otros. En esta tesis, nos enfocamos en los fundamentos teóricos en los cuales se basan las implementaciones correctas y eficientes de *lenguajes de programación* y *asistentes de demostración*. Estos fundamentos abarcan un amplio espectro de temas dentro de las áreas de la teoría de reescritura, teoría de tipos y semántica formal.

La observación que guía esta tesis es la de que, en general, puede haber muchas maneras diferentes de llevar a cabo un mismo cómputo. Por ejemplo, en la expresión $2^{1234} * 0$, podríamos empezar por calcular la potencia 2^{1234}, o podríamos observar que el resultado será 0 independientemente de cuál sea el valor que tenga 2^{1234}. Las *maneras* en que los cómputos se llevan a cabo se conocen como *estrategias de evaluación*. Esta tesis trata sobre las estrategias de evaluación en un marco muy específico: un lenguaje conocido como el Cálculo de Sustituciones Lineales o *Linear Substitution Calculus* (LSC).

En las siguientes secciones presentamos un resumen de las contribuciones y describimos la estructura del documento.

1.0.1. Destilación de máquinas abstractas

El Capítulo 2 (**Destilación de máquinas abstractas**) es fruto del trabajo conjunto con Beniamino Accattoli y Damiano Mazza. En este capítulo, se presenta al Cálculo de Sustituciones Lineales como una "máquina abstracta abstracta".

Con este objetivo, estudiamos estrategias de reducción en el LSC y mostramos que destilan la esencia de varias máquinas abstractas. Para ello, definimos formalmente la noción de *destilería*. En términos generales, una estrategia de reducción en el LSC destila una máquina abstracta si:

- Cada estado S de la máquina se *decodifica* a un término $[\![S]\!]$ del LSC.
- Hay una relación binaria $(\equiv)$ de equivalencia estructural entre términos, que además es una *bisimulación fuerte*.
- Las transiciones de la máquina abstracta se pueden clasificar en dos tipos: *transiciones de búsqueda*, que cambian el foco de evaluación pero son computacionalmente irrelevantes y *transiciones principales*, que efectúan el cómputo, de tal modo que:
  - Si $S \rightsquigarrow S'$ es una transición de búsqueda, entonces $[\![S]\!] \equiv [\![S']\!]$.
  - Si $S \rightsquigarrow S'$ es una transición principal, entonces $[\![S]\!] \to\equiv [\![S']\!]$.

Luego mostramos que varias estrategias de reducción en el LSC destilan varias máquinas abstractas conocidas:

Estrategia de reducción	Máquina abstracta
call-by-name	máquina abstracta de Krivine [27]
call-by-value de izquierda a derecha	máquina CEK [17]
call-by-value de derecha a izquierda	máquina ZINC [29],
call-by-need	máquina de Sestoft's [35],
call-by-name fuerte	máquina de Crégut's [12],

Además, proponemos nuevas máquinas abstractas, sugeridas por el proceso de destilación, que se basan en *entornos globales (sin anidamiento)* en lugar de en *entornos locales (con anidamiento)*. En todos estos casos, el proceso de destilación asegura que la máquina abstracta implementa correctamente la estrategia de reducción dada.

Además, en cada caso, mostramos que la simulación de n pasos de reducción requiere $O(c \cdot n)$ transiciones de la máquina, donde c es un factor proporcional al tamaño del término inicial. Esto justifica que el LSC—con cualquiera de las estrategias de reducción estudiadas—constituye un modelo razonable de cómputo, en el sentido de que la ejecución de un programa se puede simular en el modelo RAM (*random access machine*) con un *overhead* a lo sumo polinomial en términos de complejidad temporal.

1.0.2. Fundamentos de call-by-need fuerte

El Capítulo 3 (**Fundamentos de call-by-need fuerte**) es fruto del trabajo en colaboración con Thibaut Balabonski, Eduardo Bonelli y Delia Kesner. En este capítulo, enfocamos nuestra atención en una extensión de la estrategia call-by-need para reducción fuerte.

La propia definición de una estrategia *call-by-need* fuerte es compleja. El motivo principal es que la evaluación call-by-need en el caso fuerte es altamente dependiente del contexto. Por ejemplo, en un término como $\lambda x.y[y\backslash xt]$, de acuerdo con la estrategia call-by-need fuerte correspondería evaluar el término t:

$$\lambda x.y[y\backslash xt] \to \lambda x.y[y\backslash xt']$$

puesto que la reducción es fuerte y buscamos obtener la forma normal *completa* del término. En contraste, en un término como $z[z\backslash\lambda x.y[y\backslash xt]]s$ la estrategia call-by-need fuerte debería efectuar el siguiente paso de sustitución:

$$z[z\backslash\lambda x.y[y\backslash xt]]s \to (\lambda x.y[y\backslash xt])[z\backslash\lambda x.y[y\backslash xt]]s$$

para ser fiel a su naturaleza "por necesidad". En este capítulo:

- **Teoría de *sharing*.** Definimos una teoría de reducción fuerte, la *teoría de sharing* (Def. 3.1). La teoría de *sharing* es un cálculo no determinístico cuyas reglas de reducción inducen una teoría ecuacional que caracteriza la equivalencia operacional de programas con sustituciones explícitas, forzando que se comparta la evaluación de subtérminos (*sharing*).

- **Estrategia call-by-need fuerte.** Definimos una estrategia para evaluación call-by-need fuerte (Def. 3.10), incluyendo varias nociones relacionadas, tales como formas normales y contextos de evaluación. La reducción call-by-need fuerte es una estrategia determinística incluida en la teoría de *sharing*.

La definición de esta estrategia se basa en la noción de *contexto de evaluación*. Los contextos de evaluación están parametrizados por un conjunto ϑ de variables que están "congeladas", es decir, son simbólicas, y por un *flag* binario que indica si el contexto de evaluación se puede componer con un contexto aplicativo de tal modo que el resultado siga siendo un contexto de evaluación.

- **Propiedades básicas de la estrategia call-by-need fuerte.** Probamos cuatro principios básicos que posee nuestra estrategia call-by-need fuerte, a saber, que las formas normales de la estrategia son β-formas normales, salvo aplicación de todas las sustituciones (Prop. 3.13), que la estrategia es determinística (Prop. 3.15), que es una extensión conservativa de formulaciones previas de call-by-need (Teo. 3.20), y que es correcta con respecto a la β-reducción (Prop. 3.22), es decir que si estrategia encuentra una forma normal, entonces el término tiene una β-forma normal.

- **Completitud de la estrategia call-by-need fuerte.** Estudiamos la *completitud* de la estrategia call-by-need fuerte con respecto a la β-reducción, es decir, si un λ-término tiene a β-forma normal, entonces la estrategia call-by-need fuerte también alcanza una forma normal. Establecemos una relación precisa entre la forma normal en el cálculo-λ y la forma normal en nuestro cálculo con sustituciones explícitas (desenrollando todas las sustituciones explícitas). La demostración de normalización combina un argumento *lógico* con un argumento *sintáctico*, extendiendo una técnica de Kesner [22]. Específicamente:

  - **Tipabilidad vs. normalización.** Proponemos un sistema de tipos intersección no idempotente para la teoría de *sharing* (Def. 3.24). Es una adaptación sencilla de sistemas existentes, siguiendo la línea de trabajo propuesta propuesta por Kesner [24]. Demostramos también que la tipabilidad en este sistema implica normalización en la teoría de *sharing* (Teo. 3.40).
  - **Completitud de la teoría.** Usamos el sistema de tipos para demostrar que la teoría de *sharing* es completa con respecto a la β-reducción (Prop. 3.42), es decir, que los términos que tienen β-forma normal también tienen forma normal en la teoría de *sharing*.
  - **Completitud de la estrategia.** Usando un resutado de factorización abstracta de Accattoli [1], demostramos que la estrategia call-by-need fuerte es completa con respecto a la teoría de *sharing* (Prop. 3.51). La demostración de este hecho se basa en un análisis de casos exhaustivo de diagramas de permutación.

1.0.3. Call-by-need fuerte para patrones y recursión

El Capítulo 4 (**Call-by-need fuerte para patrones y recursión**) es fruto del trabajo en conjunto con Eduardo Bonelli y Kareem Mohamed. En este capítulo, extendemos los resultados del capítulo anterior para incorporar *pattern matching* y recursión: los términos se extienden con constructores, una construcción `case` y un operador de punto fijo). Específicamente:

- **Teoría de sharing extendida.** Nuestro punto de partida es el cálculo-λ extendido de Grégoire y Leroy (que repasamos en Def. 4.2). Generalizamos la teoría de *sharing* para el cálculo-λ extendido (Def. 4.6) y proveemos una caracterización sintáctica de sus formas normales (Def. 4.6).

- **Sistema de tipos extendido.** Proponemos un sistema de tipos intersección no idempotente para la teoría de *sharing* extendida. (Def. 4.9). Demostramos que los términos débilmente normalizantes son tipables (Teo. 4.12) y que los términos tipables son débilmente normalizantes (Teo. 4.13). Esto requiere definir una propiedad sutil sobre los juicios de tipado (Def. 4.11).

- **Estrategia call-by-need fuerte extendida.** Proponemos una estrategia call-by-need fuerte extendida para la teoría de *sharing* extendida (Def. 4.16), y demostramos que la estrategia posee buenas propiedades, tal como en el capítulo anterior, a saber: la estrategia es determinística (Prop. 4.20), extiende conservativamente la estrategia call-by-need fuerte del capítulo anterior (Prop. 4.20), y es correcta (Prop. 4.21) y completa (Teo. 4.22) con respecto a la reducción en el cálculo-λ extendido.

1.0.4. Etiquetas de Lévy para el LSC

El capítulo 5 (**Etiquetas de Lévy para el LSC**) es fruto del trabajo conjunto con Eduardo Bonelli. En este capítulo, desarrollamos una variante del LSC en la que los términos están decorados con etiquetas, siguiendo el plan trazado por Lévy [31] en su estudio de la reducción optimal en el cálculo-λ.

A continuación estudiamos la metateoría del LSC etiquetado, probando que tiene la mayoría de las buenas propiedades esperables para un cálculo con etiquetas de Lévy. Más precisamente:

- **Un LSC etiquetado.** Motivamos algunas decisiones de diseño y definimos una variante del LSC con etiquetas de Lévy, el LLSC (Def. 5.6). Cada paso de reducción en el cálculo etiquetado tiene un *nombre*. Probamos algunas propiedades sintácticas básicas que posee el LLSC.

- **Residuos y ortogonalidad.** Probamos que el LLSC es un sistema de reescritura axiomático ortogonal (Prop. 5.32).

- **Normalización débil de reducción acotada.** Demostramos que el LLSC es débilmente normalizante si se restringe la reducción a contraer pasos cuyos nombres son etiquetas de altura acotada (Prop. 5.43).

- **Normalización fuerte de reducción acotada.** Fortalecemos el resultado anterior, demostrando que el LLSC es fuertemente normalizante si la reducción se restringe a contraer pasos cuyos nombres son etiquetas de altura acotada (Teo. 5.49). Esto implica que el LSC cumple con una versión fuerte del teorema de Finite Developments, conocida como *Finite Family Developments* (FFD).

- **Confluencia.** Damos dos demostraciones diferentes de que el LLSC es confluente (Teo. 5.51).

1.0.5. Aplicaciones del LSC etiquetado

El Capítulo 6 (**Aplicaciones del LSC etiquetado**) es una continuación del Capítulo 5, y también fruto del trabajo en conjunto con Eduardo Bonelli.

En este capítulo, aplicamos el LSC etiquetado para obtener más resultados sobre el LSC (sin etiquetas). La herramienta clave es el teorema de Finite Family Developments, demostrado en el capítulo anterior:

- **Estabilidad.** Demostramos que el LSC sin la regla gc posee la propiedad de estabilidad de radicales de Lévy (Prop. 6.1).

- **Deterministic Family Structure.** Una *Deterministic Family Structure* (DFS) es un sistema de reescritura abstracto que verifica un conjunto de axiomas en particular. Demostramos que el LSC sin la regla gc forma una DFS (Teo. 6.13).

- **Reducción optimal.** Obtenemos un resultado de reducción optimal para el LSC, como consecuencia inmediata del hecho de que el LSC sin gc forma una DFS, usando un resultado de Glauert y Khasidashvili (que recapitulamos en Teo. 6.24).

- **Estandarización.** La estandarización, hablando en general, se refiere a un mecanismo que convierte una secuencia de pasos de reducción en una forma estándar, de tal modo que dos secuencias de reducción son equivalentes por permutación si y sólo si tienen la misma forma estándar.

  Proponemos un procedimiento de estandarización para Deterministic Family Structures (Prop. 6.39), inspirado por un resultado de estandardización de Klop. Como corolario, obtenemos un resultado de estandarización para el LSC sin gc (Coro. 6.43).

- **Normalización.** Demostramos un resultado de normalización para Deterministic Family Structures (Prop. 6.54), proveyendo condiciones suficientes bajo la scuales una estrategia de reducción es normalizante. Como corolario, concluimos que, en el LSC sin gc la estrategia call-by-name (Coro. 6.56) y una variante de la estrategia call-by-need (Coro. 6.59) son ambas normalizantes.

1.0.6. Publicaciones y trabajo no incluido en esta tesis

Las siguientes publicaciones corresponden a resultados de esta tesis:

- B. Accattoli, P. Barenbaum, D. Mazza. **Distilling Abstract Machines.** *Proceedings of the International Conference on Functional Programming* (ICFP), ACM SIGPLAN Notices 49(9):363–376, 2014.

- B. Accattoli, P. Barenbaum, D. Mazza. **A Strong Distillery.** *Asian Symposium on Programming Languages and Systems* (APLAS), LNCS 9458:1–20, 2015.

- P. Barenbaum, E. Bonelli. **Optimality and the Linear Substitution Calculus.** *Formal Structures for Computation and Deduction* (FSCD), 9:1–9:16, 2017.

- T. Balabonski, P. Barenbaum, E. Bonelli, D. Kesner. **Foundations of Strong Call by Need.** *Proceedings of the International Conference on Functional Programming* (ICFP), ACM SIGPLAN Notices 20:1–20:29, 2017.

- P. Barenbaum, E. Bonelli, K. Mohamed. **Pattern Matching and Fixed Points: Resource Types and Strong Call-By-Need.** *Principles and Practice of Declarative Programming* (PPDP), 6:1–6:12, 2018.

Hay además otro trabajo en el que estuve involucrado durante mi doctorado que no se incluye en este documento. Junto con Gonzalo Ciruelos, usamos un cálculo confluente basado en un sistema de tipos intersección no idempotente para estudiar espacios de derivación en el cálculo-λ puro (sin tipos). Este fue el tema de la Tesis de Licenciatura de Gonzalo y resultó también en una publicación:

- P. Barenbaum, G. Ciruelos. **Factoring Derivation Spaces via Intersection Types.** *Asian Symposium on Programming Languages and Systems* (APLAS), 24–44, 2018.

Capítulo 2

Destilación de máquinas abstractas

2.1. Introducción

Este capítulo es fruto de la colaboración con Beniamino Accattoli y Damiano Mazza y se estructura de la siguiente manera. Destacamos en negrita las principales contribuciones:

- En la Sección 2.2 **presentamos cinco estrategias de reducción** (Def. 2.2) usando sustituciones explícitas a distancia. Específicamente, las cinco estrategias de reducción son: (1) call-by-name, (2) call-by-value, con evaluación de izquierda a derecha, (3) call-by-value, con evaluación de derecha a izquierda, (4) call-by-need, (5) call-by-name fuerte.

  Las primeras cuatro estrategias son sencillas de definir, a través de una noción adecuada de contexto de evaluación. Estas estrategias son conocidas en la literatura desde hace décadas y no reclamamos originalidad alguna por su definición, aunque cabe destacar que esta es la primera presentación que utiliza sustituciones explícitas a distancia. En particular, la estrategia call-by-need débil es simple en comparación con formulaciones previas [5, 33, 6, 11]—tiene dos reglas de reducción y la gramática de los contextos de evaluación consta únicamente de cuatro producciones.

  La estrategia call-by-name fuerte requiere más cuidado. Nuestra presentación sigue el trabajo de Accattoli y Dal Lago [4]. En la Sección 2.2.1 demostramos que el call-by-name fuerte, definido usando contextos de evaluación, se corresponde con la reducción *linear leftmost-outermost* en el LSC [2, 4]—es decir, es simultáneamente un refinamiento de la β-reducción *leftmost-outermost* y una extensión de la reduccíon *linear head* para evaluación a forma normal.

  Además, demostramos que **todas estas estrategias son determinísticas** (Prop. 2.10).

- En la Sección 2.3 definimos una noción de **equivalencia estructural** $\equiv_{\mathbb{S}}$ para cada estrategia de reducción $\mathbb{S}$ definida en la Sección 2.2. El principal resultado técnico es que, para cada estrategia $\mathbb{S}$, resulta que **la relación de equivalencia estructural** $\equiv_{\mathbb{S}}$ **es una bisimulación fuerte con respecto a la estrategia** $\mathbb{S}$ (Prop. 2.13).

- En la Sección 2.4 **introducimos la noción de destilería**, una estructura abstracta usada para relacionar las estrategias de reducción y las máquinas abstractas que las imple-

mentan.

- En la Sección 2.5 definimos máquinas abstractas que implementan cada una de las estrategias, y demostramos que **todas las máquinas abstractas definidas constituyen destilerías para las estrategias de reducción correspondientes**:

Strategy	**Abstract Machine**	
call-by-name	KAM	Sección 2.5.1
	MAM	Sección 2.5.2
call-by-value	CEK	Sección 2.5.3
	CEK dividida	Sección 2.5.4
	LAM	Sección 2.5.5
call-by-need	MAD	Sección 2.5.6
	MAD fusionada	Sección 2.5.7
	MAD con punteros	Sección 2.5.8
call-by-name fuerte	MAM fuerte	Sección 2.5.9

- Por último, en la Sección 2.6 mostramos que, para cada una de las máquinas abstractas definidas en Sección 2.5, la longitud de una ejecución en la máquina está **bilinealmente relacionada** con la longitud de la secuencia de reducción que comienza en el mismo término inicial, en la estrategia de reducción correspondiente.

2.2. Estrategias de reducción

En esta sección definimos cinco estrategias de reducción determinísticas: call-by-name (`name`), dos variantes de call-by-value ($\texttt{value}^{\texttt{LR}}$, $\texttt{value}^{\texttt{RL}}$), call-by-need (`need`), y call-by-name fuerte ($\texttt{name}^{\texttt{S}}$). Además, en Sección 2.2.2 demostramos que todas estas estrategias son determinísticas.

Definición 2.1 (Reglas de reescritura en la raíz)**.** El conjunto de *términos* está dado por la gramática $t ::= x \mid \lambda x.t \mid ts \mid t[x\backslash s]$. Los *valores* están dados por $\mathtt{v} ::= \lambda x.t$, y los *contextos de sustitución* están dados por $\mathtt{L} ::= \Box \mid \mathtt{L}[x\backslash t]$. Un término de la forma $\mathtt{vL}$ se denomina una *respuesta.*

Dada una familia fija de *contextos de evaluación* ($\mathtt{E}, \mathtt{E}', \ldots$), definimos las cuatro siguientes *reglas de reescritura en la raíz*—dos reglas tipo `db` y dos reglas tipo `ls`:

$$\begin{array}{rcl}
(\lambda x.t)\mathtt{L}\, s & \mapsto_{\mathtt{db}} & t[x\backslash s]\mathtt{L} \\
(\lambda x.t)\mathtt{L}\, \mathtt{vL}' & \mapsto_{\mathtt{dbv}} & t[x\backslash \mathtt{vL}']\mathtt{L} \\
\mathtt{E}\langle\!\langle x \rangle\!\rangle[x\backslash s] & \mapsto_{\mathtt{ls}} & \mathtt{E}\langle s \rangle[x\backslash s] \\
\mathtt{E}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{vL}] & \mapsto_{\mathtt{lsv}} & \mathtt{E}\langle \mathtt{v} \rangle[x\backslash \mathtt{v}]\mathtt{L}
\end{array}$$

Observar que en las reglas terminadas en "v", se espera que el argumento de la aplicación o ustitución sea una respuesta. Además, usamos las notaciones $\overset{\mathtt{E}}{\mapsto}_{\mathtt{ls}}$ y $\overset{\mathtt{E}}{\mapsto}_{\mathtt{lsv}}$ para especificar la familia de contextos que utilizan las reglas, donde $\mathtt{E}$ es la meta-variable utilizada para designar contextos de esa familia.

Una estrategia de reducción se especifica por medio de una **elección de reglas de reducción en la raíz**, elección que consta siempre de una regla *multiplicativa* (db or dbv) una regla *exponencial* (ls or lsv), y una **familia de contextos de evaluación**. La regla multiplicativa (resp. exponencial) elegida se nota, genéricamente, $\mapsto_{\mathtt{m}}$ (resp. $\mapsto_{\mathtt{e}}$). Si $\mathtt{E}$ toma valores sobre una familia fija de contextos de evaluación, las clausuras contextuales de las reglas de reescritura en la raíz se notan $\to_{\mathtt{m}} \stackrel{\text{def}}{=} \mathtt{E}\langle\mapsto_{\mathtt{m}}\rangle$ y $\to_{\mathtt{e}} \stackrel{\text{def}}{=} \mathtt{E}\langle\mapsto_{\mathtt{e}}\rangle$. La relación de reescritura que define la estrategia de reducción es la unión $\to \stackrel{\text{def}}{=} \to_{\mathtt{m}} \cup \to_{\mathtt{e}}$.

Definición 2.2 (Estrategias de reducción name, $\mathtt{value}^{\mathrm{LR}}$, $\mathtt{value}^{\mathrm{RL}}$, need, $\mathtt{name}^{\mathrm{S}}$)**.** Las estrategias de reducción *call-by-name* (name), *call-by-value de izquierda a derecha* ($\mathtt{value}^{\mathrm{LR}}$), *call-by-value de derecha a izquierda* ($\mathtt{value}^{\mathrm{RL}}$), *call-by-need* (need), y *call-by-name fuerte* ($\mathtt{name}^{\mathrm{S}}$), se especifican por las siguientes elecciones de reglas de reducción en la raíz y familias de contextos de evaluación:

Estrategia	Contextos de evaluación	$\mapsto_{\mathtt{m}}$	$\mapsto_{\mathtt{e}}$	$\to_{\mathtt{m}}$	$\to_{\mathtt{e}}$
name	$\mathtt{H} ::= \Box \mid \mathtt{H}\,t \mid \mathtt{H}[x\backslash t]$	$\mapsto_{\mathtt{db}}$	$\stackrel{\mathtt{H}}{\mapsto}_{\mathtt{ls}}$	$\mathtt{H}\langle\mapsto_{\mathtt{db}}\rangle$	$\mathtt{H}\langle\stackrel{\mathtt{H}}{\mapsto}_{\mathtt{ls}}\rangle$
$\mathtt{value}^{\mathrm{LR}}$	$\mathtt{V} ::= \Box \mid \mathtt{V}\,t \mid \mathtt{vL}\,\mathtt{V} \mid \mathtt{V}[x\backslash t]$	$\mapsto_{\mathtt{dbv}}$	$\stackrel{\mathtt{V}}{\mapsto}_{\mathtt{lsv}}$	$\mathtt{V}\langle\mapsto_{\mathtt{dbv}}\rangle$	$\mathtt{V}\langle\stackrel{\mathtt{V}}{\mapsto}_{\mathtt{lsv}}\rangle$
$\mathtt{value}^{\mathrm{RL}}$	$\mathtt{R} ::= \Box \mid \mathtt{R}\,\mathtt{vL} \mid t\,\mathtt{R} \mid \mathtt{R}[x\backslash t]$	$\mapsto_{\mathtt{dbv}}$	$\stackrel{\mathtt{R}}{\mapsto}_{\mathtt{lsv}}$	$\mathtt{R}\langle\mapsto_{\mathtt{dbv}}\rangle$	$\mathtt{R}\langle\stackrel{\mathtt{R}}{\mapsto}_{\mathtt{lsv}}\rangle$
need	$\mathtt{N} ::= \Box \mid \mathtt{N}\,t \mid \mathtt{N}[x\backslash t] \mid \mathtt{N}'\langle x\rangle[x\backslash \mathtt{N}]$	$\mapsto_{\mathtt{db}}$	$\stackrel{\mathtt{N}}{\mapsto}_{\mathtt{lsv}}$	$\mathtt{N}\langle\mapsto_{\mathtt{db}}\rangle$	$\mathtt{N}\langle\stackrel{\mathtt{N}}{\mapsto}_{\mathtt{lsv}}\rangle$
$\mathtt{name}^{\mathrm{S}}$	$\mathtt{S} ::=$ (contextos $\mathcal{S}$, ver Def. 2.3)	$\mapsto_{\mathtt{db}}$	$\stackrel{\mathtt{S}}{\mapsto}_{\mathtt{ls}}$	$\mathtt{S}\langle\mapsto_{\mathtt{db}}\rangle$	$\mathtt{S}\langle\stackrel{\mathtt{S}}{\mapsto}_{\mathtt{ls}}\rangle$

2.2.1. Call-by-name fuerte

La estrategia call-by-name fuerte es la única estrategia de reducción *fuerte* que estudiamos en este capítulo. Para completar la definición de la estrategia call-by-name fuerte, debemos dar una definición para la correspondiente familia de contextos de evaluación ($\mathtt{S}, \mathtt{S}', \ldots$):

Definición 2.3 (Contextos de evaluación call-by-name fuerte)**.** Un término es *neutral* si es $\to_{\mathtt{db}\cup\mathtt{ls}}$-normal en el LSC y no es de la forma $(\lambda x.t)\mathtt{L}$. Un contexto $\mathtt{C}$ es un *contexto de evaluación call-by-name fuerte* si el juicio "$\mathtt{C} \in \mathcal{S}$" es derivable a partir de las siguientes reglas de inferencia inductivas:

$$\frac{}{\Box \in \mathcal{S}}\,(\textsc{ax}\text{-}\mathcal{S}) \qquad \frac{\mathtt{C} \in \mathcal{S} \quad \mathtt{C} \neq (\lambda x.\mathtt{C}')\mathtt{L}}{\mathtt{C}\,t \in \mathcal{S}}\,(@\textsc{l}\text{-}\mathcal{S})$$

$$\frac{\mathtt{C} \in \mathcal{S}}{\lambda x.\mathtt{C} \in \mathcal{S}}\,(\lambda\text{-}\mathcal{S}) \qquad \frac{t \text{ es neutral} \quad \mathtt{C} \in \mathcal{S}}{t\,\mathtt{C} \in \mathcal{S}}\,(@\textsc{r}\text{-}\mathcal{S})$$

$$\frac{\mathtt{C} \in \mathcal{S} \quad x \notin \mathtt{lfv}(\mathtt{C})}{\mathtt{C}[x\backslash t] \in \mathcal{S}}\,(\text{ES-}\mathcal{S})$$

Caracterización alternativa de la estrategia call-by-name fuerte

La estrategia call-by-name fuerte se puede caracterizar exactamente como la reducción *leftmost-outermost lineal* $\to_{\textsc{lo}}$. Para definir $\to_{\textsc{lo}}$ necesitamos algunas definiciones previas:

Definición 2.4 (Orden LO)**.** Escribimos $\mathtt{C} \prec_p t$ si hay un término s tal que $\mathtt{C}\langle s\rangle = t$. La llamamos *relación de prefijo.*

El orden *de adentro hacia afuera* $\mathtt{C} \prec_O \mathtt{C}'$ entre contextos arbitrarios $\mathtt{C}, \mathtt{C}'$ se define por medio de las siguientes reglas:

1. *Root*: $\Box \prec_O \mathtt{C}$ para todo contexto $\mathtt{C} \neq \Box$.

2. *Clausura contextual*: si $\mathtt{C} \prec_O \mathtt{C}'$ entonces $\mathtt{C}''\langle\mathtt{C}\rangle \prec_O \mathtt{C}''\langle\mathtt{C}'\rangle$ para cualquier contexto $\mathtt{C}''$.

Observar que $\prec_O$ se puede ver como la relación de prefijo $\prec_p$ sobre contextos. El orden *de izquierda a derecha* $\mathtt{C} \prec_L \mathtt{C}'$ se define por:

1. *Aplicación*: si $\mathtt{C} \prec_p t$ y $\mathtt{C}' \prec_p s$ entonces $\mathtt{C}\, s \prec_L t\, \mathtt{C}'$.

2. *Sustitución*: si $\mathtt{C} \prec_p t$ y $\mathtt{C}' \prec_p s$ entonces $\mathtt{C}[x\backslash s] \prec_L t[x\backslash\mathtt{C}']$.

3. *Clausura contextual*: si $\mathtt{C} \prec_L \mathtt{C}'$ entonces $\mathtt{C}''\langle\mathtt{C}\rangle \prec_L \mathtt{C}''\langle\mathtt{C}'\rangle$ para cualquier contexto $\mathtt{C}''$.

Por último, el orden *de izquierda a derecha, de afuera hacia adentro* se define así: $\mathtt{C} \prec_{\mathsf{LO}} \mathtt{C}'$ si $\mathtt{C} \prec_O \mathtt{C}'$ o $\mathtt{C} \prec_L \mathtt{C}'$.

Lema 2.5 (Totalidad de $\prec_{\mathsf{LO}}$)**.** *Si* $\mathtt{C} \prec_p t$ *y* $\mathtt{C}' \prec_p t$ *entonces o bien* $\mathtt{C} \prec_{\mathsf{LO}} \mathtt{C}'$*, o bien* $\mathtt{C}' \prec_{\mathsf{LO}} \mathtt{C}$*, o bien* $\mathtt{C} = \mathtt{C}'$*.*

Identificamos los redexes con el contexto que se enfoca en su *ancla.* Recordar que el ancla de un paso `db` es la aplicación que se contrae, y el ancla de un paso `ls` es la variable que se contrae.

Definición 2.6 (Reducción LO lineal $\to_{\text{LO}}$)**.** Sea t un término. Un redex $\mathtt{C}$ es el redex *leftmost-outermost* (LO) de t si $\mathtt{C} \prec_{\mathsf{LO}} \mathtt{C}'$ para cualquier otro redex $\mathtt{C}'$ de t. Escribimos $t \to_{\text{LO}} s$ para un paso que contrae el redex leftmost-outermost.

A continuación definimos los contextos LO y demostramos que la posición de un paso LO lineal es siempre un contexto LO. Necesitamos la noción de *variable libre a izquierda* de un contexto, es decir, la de una variable que ocurre libre a la izquierda del agujero.

Definición 2.7 (Variables libres a izquierda)**.** El conjunto $\mathtt{lfv}(\mathtt{C})$ de *variables libres a izquierda* de $\mathtt{C}$ se define por:

$$\begin{array}{rcl@{\qquad}rcl}
\mathtt{lfv}(\Box) & \stackrel{\text{def}}{=} & \varnothing & \mathtt{lfv}(t\mathtt{C}) & \stackrel{\text{def}}{=} & \mathtt{fv}(t) \cup \mathtt{lfv}(\mathtt{C}) \\
\mathtt{lfv}(\lambda x.\mathtt{C}) & \stackrel{\text{def}}{=} & \mathtt{lfv}(\mathtt{C})\backslash\{x\} & \mathtt{lfv}(\mathtt{C}[x\backslash t]) & \stackrel{\text{def}}{=} & \mathtt{lfv}(\mathtt{C})\backslash\{x\} \\
\mathtt{lfv}(\mathtt{C}t) & \stackrel{\text{def}}{=} & \mathtt{lfv}(\mathtt{C}) & \mathtt{lfv}(t[x\backslash\mathtt{C}]) & \stackrel{\text{def}}{=} & (\mathtt{fv}(t)\backslash\{x\}) \cup \mathtt{lfv}(\mathtt{C})
\end{array}$$

Por último, podemos definir los contextos LO:

Definición 2.8 (Contextos LO)**.** Un contexto $\mathtt{C}$ es LO si:

1. *Aplicación a derecha*: cada vez que $\mathtt{C} = \mathtt{C}'\langle t\, \mathtt{C}''\rangle$ se tiene que t es neutral.

2. *Aplicación a izquierda*: cada vez que $\mathtt{C} = \mathtt{C}'\langle \mathtt{C}''t\rangle$ se tiene que $\mathtt{C}'' \neq \mathtt{L}\langle \lambda x.\mathtt{C}'''\rangle$.

3. *Sustitución*: cada vez que $\mathtt{C} = \mathtt{C}'\langle \mathtt{C}''[x\backslash s]\rangle$ se tiene que $x \notin \mathtt{lfv}(\mathtt{C}'')$.

Lema 2.9 (Caracterización de contextos LO)**.**

1. *Sea* $\mathtt{C}$ *un contexto. Entonces* $\mathtt{C} \in \mathcal{S}$ *si y sólo si* $\mathtt{C}$ *es* LO.

2. *Sea* $t \to s$ *contrayendo un redex bajo un contexto* $\mathtt{C}$*. Entonces* $\mathtt{C}$ *es un paso* $\to_{LO}$ *si y sólo si* $\mathtt{C}$ *es* LO.

2.2.2. Determinismo

Todas las estrategias de reducción estudiadas en este capítulo son determinísticas:

Proposición 2.10 (Determinismo)**.** *Las cinco estrategias de Def. 2.2 son determinísticas. En cada caso, si* $\mathrm{E}_1, \mathrm{E}_2$ *son contextos de evaluación,* r_1, r_2 *son* anclas*, y* $\mathrm{E}_1\langle r_1\rangle = \mathrm{E}_2\langle r_2\rangle$*, entonces* $\mathrm{E}_1 = \mathrm{E}_2$ *y* $r_1 = r_2$*. Así, hay a lo sumo una manera de reducir un término de acuerdo con cualquiera de dichas estrategias.*

2.3. Equivalencia estructural

Cada una de las cinco estrategias de reducción $\mathbb{S} \in \{\mathtt{name}, \mathtt{value}^{\mathtt{LR}}, \mathtt{value}^{\mathtt{RL}}, \mathtt{need}, \mathtt{name}^{\mathtt{S}}\}$ presentadas hasta el momento viene equipada con una noción correspondiente de *equivalencia estructural*, escrita $\equiv_{\mathbb{S}}$. La equivalencia estructural permite manipular las sustituciones explícitas, "moviéndolas" de forma *computacionalmente irrelevante*. Técnicamente, esto se expresa con la propiedad de que la equivalencia estructural es una *bisimulación fuerte*.

Cada noción de equivalencia estructural está dada eligiendo algunos de los siguientes axiomas:

Definición 2.11 (Axiomas para equivalencia estructural)**.**

$$
\begin{array}{rcll}
(\lambda x.t)[y\backslash s] & \equiv_{\lambda} & \lambda x.t[y\backslash s] & \text{if } x \notin \mathtt{fv}(s) \\
(t\,u)[x\backslash s] & \equiv_{@} & t[x\backslash s]u[x\backslash s] & \\
(t\,u)[x\backslash s] & \equiv_{@\mathtt{l}} & t[x\backslash s]\,u & \text{if } x \notin \mathtt{fv}(u) \\
(t\,u)[x\backslash s] & \equiv_{@\mathtt{r}} & t\,u[x\backslash s] & \text{if } x \notin \mathtt{fv}(t) \\
t[x\backslash s][y\backslash u] & \equiv_{\mathtt{com}} & t[y\backslash u][x\backslash s] & \text{if } y \notin \mathtt{fv}(s) \text{ and } x \notin \mathtt{fv}(u) \\
t[x\backslash s][y\backslash u] & \equiv_{[\cdot]} & t[x\backslash s[y\backslash u]] & \text{if } y \notin \mathtt{fv}(t) \\
t[x\backslash s] & \equiv_{\mathtt{gc}} & t & \text{if } x \notin \mathtt{fv}(t) \\
t[x\backslash s] & \equiv_{\mathtt{dup}} & t_{[y]_x}[x\backslash s][y\backslash s] &
\end{array}
$$

En la regla $\equiv_{\mathtt{dup}}$, $t_{[y]_x}$ denota un término obtenido a partir de t renombrando algunas (posiblemente ninguna) de las ocurrencias de x como y.

Definición 2.12 (Equivalencias estructurales)**.** Para cada estrategia $\mathbb{S}$, elegimos un subconjunto de los axiomas de equivalencia estructural, y una familia de contextos, del siguiente modo:

Estrategia	Axiomas de equivalencia estructural	Familia de contextos
$\mathtt{name}$	$\equiv_{@}, \equiv_{\mathtt{com}}, \equiv_{[\cdot]}, \equiv_{\mathtt{gc}}, \equiv_{\mathtt{dup}}$	$\mathtt{H}$
$\mathtt{value}^{\mathtt{LR}}$	$\equiv_{@}, \equiv_{\mathtt{com}}, \equiv_{[\cdot]}, \equiv_{\mathtt{gc}}, \equiv_{\mathtt{dup}}$	$\mathtt{V}$
$\mathtt{value}^{\mathtt{RL}}$	$\equiv_{@}, \equiv_{\mathtt{com}}, \equiv_{[\cdot]}, \equiv_{\mathtt{gc}}, \equiv_{\mathtt{dup}}$	$\mathtt{R}$
$\mathtt{need}$	$\equiv_{@\mathtt{l}}, \equiv_{\mathtt{com}}, \equiv_{[\cdot]}$	$\mathtt{N}$
$\mathtt{name}^{\mathtt{S}}$	$\equiv_{\lambda}, \equiv_{@\mathtt{l}}, \equiv_{@\mathtt{r}}, \equiv_{\mathtt{com}}, \equiv_{[\cdot]}, \equiv_{\mathtt{gc}}, \equiv_{\mathtt{dup}}$	$\mathtt{C}$ (contextos arbitrarios)

La equivalencia estructural $\equiv_{\mathbb{S}}$ correspondiente se define como la clausura reflexiva, simétrica, transitiva y contextual de los axiomas, bajo la familia de contextos especificada.

Observar que las equivalencias estructurales para call-by-name y call-by-value usan los mismos axiomas, pero cerrados bajo sus respectivas nociones de contexto de evaluación. La equivalencia estructural para call-by-name fuerte está cerrada bajo contextos arbitrarios. Por ejemplo:

$$\begin{array}{rcll} ((\lambda x.x)y)[x\backslash x'][y\backslash y'] & \equiv_{\mathtt{value}^{\mathtt{LR}}} & ((\lambda x.x)y)[y\backslash y'][x\backslash x'] & (\text{by } \equiv_{\mathtt{com}}) \\ & \equiv_{\mathtt{value}^{\mathtt{LR}}} & ((\lambda x.x)[y\backslash y']\, y[y\backslash y'])[x\backslash x'] & (\text{by } \equiv_{@}) \\ & \equiv_{\mathtt{value}^{\mathtt{LR}}} & ((\lambda x.x)\, y[y\backslash y'])[x\backslash x'] & (\text{by } \equiv_{\mathtt{gc}}) \end{array}$$

y:

$$\begin{array}{rcll} (\lambda x.y\, y)[y\backslash z] & \equiv_{\mathtt{name}^{\mathtt{S}}} & \lambda x.(y\, y)[y\backslash z] & (\text{by } \equiv_{\lambda}) \\ & \equiv_{\mathtt{name}^{\mathtt{S}}} & \lambda x.(y_1\, y_2)[y_1\backslash z][y_2\backslash z] & (\text{by } \equiv_{\mathtt{dup}}) \\ & \equiv_{\mathtt{name}^{\mathtt{S}}} & \lambda x.(y_1[y_1\backslash z]\, y_2)[y_2\backslash z] & (\text{by } \equiv_{@\mathtt{l}}) \\ & \equiv_{\mathtt{name}^{\mathtt{S}}} & \lambda x.y_1[y_1\backslash z]\, y_2[y_2\backslash z] & (\text{by } \equiv_{@\mathtt{r}}) \end{array}$$

El resultado clave es el siguiente:

Proposición 2.13 (La equivalencia estructural es una bisimulación fuerte)**.** *Sea* $\mathtt{x} \in \{\mathtt{m}, \mathtt{e}\}$*. Si* $t \equiv_{\mathbb{S}} t' \to_{\mathtt{x}} s$ *entonces existe* s' *tal que* $t \to_{\mathtt{x}} s' \equiv_{\mathbb{S}} s$*.*

Una propiedad esencial de las bisimulaciones fuertes es que pueden posponerse. En efecto, es inmediato probar el siguiente resultado para cualquiera de las cinco estrategias de Def. 2.2:

Lema 2.14 (Postposición de la equivalencia estructural)**.** *Sea* $t\ (\to_{\mathtt{m}} \cup \to_{\mathtt{e}} \cup \equiv)^*\ s$*. Entonces* $t\ (\to_{\mathtt{m}} \cup \to_{\mathtt{e}})^* \equiv s$ *y el número de pasos multiplicativos y exponenciales en las dos secuencias de reducción es exactamente el mismo.*

En los teoremas de simulación para máquinas con entornos globales usaremos también la siguiente propiedad de conmutación entre sustituciones y contextos de evaluación:

Lema 2.15 (Las sustituciones explícitas conmutan con los contextos de evaluación, salvo $\equiv$)**.** *Sea* $\mathtt{E}$ *un contexto de evaluación para una estrategia* $\mathbb{S}$*. Si* $x \notin \mathtt{fv}(\mathtt{E})$ *y* $\mathtt{E}$ *no liga ninguna de las variables libres de* s*, entonces* $\mathtt{E}\langle t\rangle[x\backslash s] \equiv_{\mathbb{S}} \mathtt{E}\langle t[x\backslash s]\rangle$*.*

2.4. Destilerías

En esta sección se presenta una visión abstracta, de alto nivel, de la relación entre máquinas abstractas y cálculos de sustituciones explícitas, a través de la siguiente noción:

Definición 2.16 (Destilería)**.** Una *destilería* $\mathbb{D} = (\mathbb{M}, \mathbb{S}, \equiv, [\![\cdot]\!])$ está dada por:

1. Una *máquina abstracta* $\mathbb{M}$, dada por:
   - 1.1 Una relación de reducción determinística $\rightsquigarrow_{\mathbb{M}}$ sobre un conjunto de estados $\mathsf{State} = \{S_1, S_2, \ldots\}$.
   - 1.2 Una clase distinguida de estados llamados estados *iniciales*, en biyección con λ-términos cerrados y de la cual se obtienen los estados *alcanzables* aplicando $\rightsquigarrow_{\mathbb{M}}{}^*$.
   - 1.3 Una partición de las transiciones que definen la relación $\rightsquigarrow_{\mathbb{M}}$:
     - 1.3.1 Transiciones de *búsqueda*, notadas $\rightsquigarrow_{\mathtt{s}}$.
     - 1.3.2 Transiciones *principales*, que a su vez se particionan en:
       - 1.3.2.1 Transiciones *multiplicativas*, notadas $\rightsquigarrow_{\mathtt{m}}$.
       - 1.3.2.2 Transiciones *exponenciales*, notadas $\rightsquigarrow_{\mathtt{e}}$.
2. Una *estrategia de reducción* determinística $\mathbb{S}$ dada por un par $(\to_{\mathtt{m}}, \to_{\mathtt{e}})$ de relaciones de reescritura sobre términos con sustituciones explícitas.
3. Una relación de *equivalencia estructural* $\equiv$ sobre términos con sustituciones explícitas, tal que $\equiv$ es una bisimulación fuerte con respecto a $\to_{\mathtt{m}}$ y $\to_{\mathtt{e}}$.
4. Una *destilación* $[\![\cdot]\!]$, es decir, una función de decodificación de estados a términos, tal que, sobre los estados alcanzables se tiene:
   - 4.1 *Búsqueda*: $S \rightsquigarrow_{\mathtt{s}} S'$ implica $[\![S]\!] \equiv [\![S']\!]$.
   - 4.2 *Multiplicativa*: $S \rightsquigarrow_{\mathtt{m}} S'$ implica $[\![S]\!] \to_{\mathtt{m}}\equiv [\![S']\!]$.
   - 4.3 *Exponencial*: $S \rightsquigarrow_{\mathtt{e}} S'$ implica $[\![S]\!] \to_{\mathtt{e}}\equiv [\![S']\!]$.

Dada una destilería, el siguiente resultado de *simulation* se verifica de forma abstracta. Escribamos $|\rho|$ (resp. $|\pi|$), $|\rho|_m$ (resp. $|\pi|_m$), $|\rho|_e$ (resp. $|\pi|_e$), y $|\rho|_p$ para notar el número de pasos no especificados, multiplicativos, exponenciales y principales en una ejecución $\rho : S \rightsquigarrow_{\mathbb{M}}{}^* S'$ de la máquina (resp. en una derivación $\pi : t \to^*_{\mathbb{S}} t'$ de la estrategia). Entonces:

Proposición 2.17 (Simulación)**.** *Sea $\mathbb{D}$ una destilería. Entonces para cada ejecución $\rho : S \rightsquigarrow_{\mathbb{M}}{}^* S'$ hay una derivación $\pi : [\![S]\!] \to^*\equiv [\![S']\!]$ tal que $|\rho|_m = |\pi|_m$, $|\rho|_e = |\pi|_e$, y $|\rho|_p = |\pi|$.*

2.4.1. Destilerías reflectivas

Dada una destilería, uno esperaría que la reducción en la estrategia se refleje en la máquina. Este resultado requiere dos propiedades abstractas adicionales:

Definición 2.18 (Destilería reflectiva)**.** Una destilería es *reflectiva* cuando:

1. *Terminación:* las transiciones de búsqueda $\rightsquigarrow_{\mathtt{s}}$ *terminan* comenzando desde estados alcanzables. Así, por determinismo, todo estado S tiene una única *forma normal de búsqueda* $\mathtt{nf}_{\mathtt{s}}(S)$.

2. *Progreso:* si S es alcanzable, $\mathtt{nf}_{\mathtt{s}}(S) = S$ y $[\![S]\!] \to_{\mathtt{x}} t$ con $\mathtt{x} \in \{\mathtt{m}, \mathtt{e}\}$, entonces existe un estado S' tal que $S \rightsquigarrow_{\mathtt{x}} S'$.

Así podemos demostrar el siguiente resultado de reflexión abstracto:

Lema 2.19 (Reflexión)**.** *Sea $\mathbb{D}$ una destilería reflectiva. Sea S un estado alcanzable, y sea $\mathtt{x} \in \{\mathtt{m}, \mathtt{e}\}$. Entonces $[\![S]\!] \to_{\mathtt{x}} t$ implica que existe un estado S' tal que $\mathtt{nf}_{\mathtt{s}}(S) \rightsquigarrow_{\mathtt{x}} S'$ y $[\![S']\!] \equiv t$.*

El lema precedente se puede extender fácilmente a un resultado de simulación reversa:

Proposición 2.20 (Simulación reversa)**.** *Sea $\mathbb{D}$ una destilería reflexiva y sea S un estado inicial. Dada una derivación $\pi : [\![S]\!] \to^* t$ hay una ejecución $\rho : S \rightsquigarrow_{\mathbb{M}}{}^* S'$ tal que $t \equiv [\![S']\!]$ y $|\rho|_m = |\pi|_m$, $|\rho|_e = |\pi|_e$, y $|\rho|_p = |\pi|$.*

2.5. Máquinas abstractas

En esta sección introducimos *máquinas abstractas* y sus *destilaciones*, y demostramos que forman *destilerías reflectivas* con respecto a las estrategias de Sección 2.2.

2.5.1. Call-by-name: la KAM

La máquina abstracta de Krivine (KAM), definida originalmente por Jean-Louis Krivine [27], es la primera máquina que estudiamos en este capítulo.

Definición 2.21 (Máquina abstracta de Krivine)**.** Un *estado* de la KAM $(S, S', S'', \ldots)$ es un par (c, π), donde c es una *clausura* y π es una *pila* de clausuras. Las clausuras se definen mutuamente recursivamente con los *entornos locales* e:

$$\begin{array}{rcl@{\qquad}rcl} c & ::= & (\bar{t}, e) & e & ::= & \epsilon \mid [x\backslash c] :: e \\ \pi & ::= & \epsilon \mid c :: \pi & S & ::= & (c, \pi) \end{array}$$

Por legibilidad, notamos $\bar{t} \mid e \mid \pi$ para un estado (c, π) donde $c = (\bar{t}, e)$. Las transiciones de la KAM son:

$$\begin{array}{ccccccccc} \bar{t}\bar{s} & \mid & e & \mid & \pi & \rightsquigarrow_{\mathtt{s}} & \bar{t} & \mid & e & \mid & (\bar{s}, e) :: \pi \\ \lambda x.\bar{t} & \mid & e & \mid & c :: \pi & \rightsquigarrow_{\mathtt{m}} & \bar{t} & \mid & [x\backslash c] :: e & \mid & \pi \\ x & \mid & e & \mid & \pi & \rightsquigarrow_{\mathtt{e}} & \bar{t} & \mid & e' & \mid & \pi \end{array}$$

donde $\rightsquigarrow_{\mathtt{e}}$ aplica sólo si $e = e_1 :: [x\backslash(\bar{t}, e')] :: e_2$.

Un aspecto crucial de nuestra metodología es que los entornos y pilas pueden ser entendidos como contextos, a través de la siguiente decodificación, que verifica las propiedades enunciadas en el siguiente lema:

Definición 2.22 (Decodificación de la KAM)**.**

$$\begin{array}{rcl} [\![\epsilon]\!] & \stackrel{\text{def}}{=} & \square \\ [\![(\bar{t}, e)]\!] & \stackrel{\text{def}}{=} & [\![e]\!]\langle\bar{t}\rangle \\ [\![\bar{t} \mid e \mid \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle[\![e]\!]\langle\bar{t}\rangle\rangle \end{array} \qquad \begin{array}{rcl} [\![[x\backslash c] :: e]\!] & \stackrel{\text{def}}{=} & [\![e]\!]\langle\square[x\backslash[\![c]\!]]\rangle \\ [\![c :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle\square[\![c]\!]\rangle \end{array}$$

Lema 2.23 (Decodificación contextual)**.** *Sean e un entorno y π una pila de la KAM. Entonces $[\![e]\!]$ es un contexto de sustitución, y tanto $[\![\pi]\!]$ como $[\![\pi]\!]\langle[\![e]\!]\rangle$ son contextos de evaluación call-by-name.*

A continuación enunciamos los invariantes dinámicos de la máquina:

Lema 2.24 (Invariantes de la KAM)**.** *Sea $S = \bar{s} \mid e \mid \pi$ un estado alcanzable de la KAM cuyo código inicial $\bar{t}$ es consistente en nombres. Entonces:*

1. Clausura*: todas las clausuras de S son cerradas.*
2. Subtérmino*: cualquier código en S es un subtérmino de $\bar{t}$.*
3. Nombre*: cualquier clausura c en S es consistente en nombres y sus nombres son nombres de $\bar{t}$ (*i.e. $\mathsf{supp}(c) \subseteq \mathtt{fv}(\bar{t})$*).*
4. Tamaño del entorno*: la longitud de cualquier entorno en S está acotada por $|\bar{t}|$.*

Consideraciones abstractas sobre implementaciones concretas. El invariante de *nombre* es la propiedad abstracta que permite evitar tanto la α-equivalencia como la generación de nombres en las ejecuciones de la KAM. Debe tenerse en cuenta que, por definición de clausura consistente en nombres, no puede haber repeticiones en el soporte de un entorno. Por lo tanto la longitud de cualquier entorno en cualquier estado alcanzable está limitada por el número de nombres distintos en el código inicial $\bar{t}$, es decir con $|\bar{t}|$. Este hecho es importante, ya que el límite estático en el tamaño de los entornos garantiza que $\rightsquigarrow_{\mathtt{e}}$ y $\rightsquigarrow_{\mathtt{s}}$—las transiciones que hacen búsqueda y copias de entornos—pueden implementarse (independientemente de la representación concreta elegida) en peor caso en tiempo lineal en $|t|$, para que una ejecución ρ se pueda implementar en $O(|\rho| \cdot |t|)$. Lo mismo se aplicará a todas las máquinas con entornos locales. De hecho, esta observación puede entenderse como una definición: diremos que una máquina abstracta es *razonable* si su implementación posee la cota bilineal anterior. Así la duración de una ejecución de una máquina razonable proporciona una estimación precisa de su costo de implementación.

Por último, tenemos:

Teorema 2.25 (Destilación de la KAM)**.** $(\mathrm{KAM}, \mathtt{name}, \equiv, [\![\cdot]\!])$ *es una destilería reflectiva.*

2.5.2. Call-by-name con entorno global: la MAM

El LSC sugiere el diseño de una versión más simple de la KAM, que llamamos la *máquina abstrcta de Milner* (MAM), que no usa la noción de clausura. A nivel del lenguaje, la idea es que aplicando repetidametne los axiomas $\equiv_{\mathtt{dup}}$ y $\equiv_{@}$ de la equivalencia estructural, todas las sustituciones explícitas pueden llevarse hacia *afuera*. A nivel de la máquina, los entornos locales en las clausuras se reemplazan por un único entorno global que liga todas las variables libres del código y la pila, así como las variables libres presentes en el entorno global mismo.

Pasar naïvemente a un entorno global rompe el invariante de consistencia de nombres de la máquina. Este punto se resuelve utilizando un α-renombre, generando un nombre fresco, en la transición asociada a la variable (exponencial), es decir, cuando se efectúa una sustitución.

Definición 2.26 (Máquina abstracta de Milner)**.** La MAM emplea entornos globales $E ::= \epsilon \mid [x\backslash\bar{t}] :: E$. Las pilas son listas de códigos, es decir, $\pi ::= \epsilon \mid \bar{t} :: \pi$. Un estado es una tripla $S = (\bar{t}, \pi, E)$. Las transiciones de la MAM son:

$$\begin{array}{rcccl} \bar{t}\bar{s} \mid \pi \mid E & \rightsquigarrow_{\mathtt{s}} & \bar{t} \mid \bar{s} :: \pi \mid E \\ \lambda x.\bar{t} \mid \bar{s} :: \pi \mid E & \rightsquigarrow_{\mathtt{m}} & \bar{t} \mid \pi \mid [x\backslash\bar{s}] :: E \\ x \mid \pi \mid E & \rightsquigarrow_{\mathtt{e}} & \bar{t}^{\alpha} \mid \pi \mid E \end{array}$$

donde la transición $\rightsquigarrow_{\mathtt{e}}$ aplica sólo si $E = E''\langle E'[x\backslash\bar{t}]\rangle$ y $\bar{t}^{\alpha}$ es un código consistente en nombres, α-equivalente a $\bar{t}$ y tal que cualquier nombre ligado en $\bar{t}^{\alpha}$ es fresco con respecto a aquellos en π y E.

Definición 2.27 (Decodificación de la MAM)**.** La decodificación de un estado de la MAM $\bar{t} \mid \pi \mid E$ es similar a la decodificación de un estado de la KAM, pero los contextos de la pila y el entornoe aplican en orden inverso:

$$\begin{array}{rcl@{\qquad}rcl} [\![\epsilon]\!] & \stackrel{\text{def}}{=} & \Box & [\![[x\backslash\bar{t}] :: E]\!] & \stackrel{\text{def}}{=} & [\![E]\!]\langle\Box[x\backslash\bar{t}]\rangle \\ [\![\bar{t} :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle\Box\bar{t}\rangle & [\![\bar{t} \mid \pi \mid E]\!] & \stackrel{\text{def}}{=} & [\![E]\!]\langle[\![\pi]\!]\langle\bar{t}\rangle\rangle \end{array}$$

A cada estado $\bar{t} \mid \pi \mid E$ de la MAM le asociamos un par $([\![\pi]\!]\langle\bar{t}\rangle, E)$, al que llamamos la *clausura global* del estado. Observar que $[\![\pi]\!]\langle\bar{t}\rangle$ ahora es un código, es decir, no contiene sustituciones explícitas.

Lema 2.28 (Decodificación contextual)**.** *Sea E un entorno global y sea π una pila de la MAM. Entonces $[\![E]\!]$ es un contexto de sustitución, y tanto $[\![\pi]\!]$ como $[\![\pi]\!]\langle[\![E]\!]\rangle$ son contextos de evaluación.*

Para los invariantes dinámicos, necesitamos una noción diferente de clausura cerrada.

Definición 2.29. Dado un entorno global E y un código $\bar{t}$, definimos por inducción mutua dos predicados E *es cerrado* y $(\bar{t}, E)$ *es cerrado* del siguiente modo:

$$\begin{array}{rcl} & & \epsilon \text{ es cerrado} \\ (\bar{t}, E) \text{ es cerrado} & \Longrightarrow & [x\backslash t] :: E \text{ es cerrado} \\ \mathtt{fv}(\bar{t}) \subseteq \mathsf{supp}(E) \wedge E \text{ es cerrado} & \Longrightarrow & (\bar{t}, E) \text{ es cerrado} \end{array}$$

Los invariantes dinámicos son:

Lema 2.30 (Invariantes de la MAM). *Sea $S = \overline{s} \mid \pi \mid E$ un estado de la MAM alcanzado por una ejecución ρ de un código inicial $\overline{t}$ consistente en nombres. Entonces:*

1. Clausura global: *la clausura global $(\llbracket\pi\rrbracket\langle\overline{t}\rangle, E)$ de S es cerrada;*
2. Subtérmino: *cualquier código en S es un subtérmino de $\overline{t}$;*
3. Nombres: *la clausura global de S es consistente en nombres;*
4. Tamaño del entorno: *la longitud del entorno global en S está acotada por $|\rho|_m$.*

Consideraciones abstractas sobre implementaciones concretas. Observar la nueva versión del invariante de tamaño del entorno. La cota depende ahora del tamaño de la ejecución ρ, y no del tamaño del término inicial $\overline{t}$. Si uno implementa $\rightsquigarrow_{\mathtt{e}}$ buscando la variable x en el entorno E secuencialmente, cada transición $\rightsquigarrow_{\mathtt{e}}$ tiene costo $O(|\rho|_m)$, y se ve fácilmente que el costo de implementar ρ resulta cuadrático en $|\rho|$. Por consiguiente—a primera vista—la MAM no sería una máquina abstracta razonable. No obstante, la intención es que la MAM se implemente utilizando una representación de códigos en los que las variables son punteros, de tal modo que buscar el valor asociado a la variable x en E tome tiempo constante. Así, el entorno global, a pesar de que se formalice como una lista, debería ser entendido como una memoria.

El invariante de nombres es el que garantiza que las variables se pueden implementar como punteros, puesto que no hay colisiones. Observar que el costo de una transición $\rightsquigarrow_{\mathtt{e}}$ no es constante, pues la operación de renombre hace que el costo de $\rightsquigarrow_{\mathtt{e}}$ sea lineal en el tamaño $|t|$ (como resultado del invariante del subtérmino). Así, asumiendo una representación basada en punteros, ρ se puede implementar en tiempo $O(|\rho| \cdot |\overline{t}|)$, al igual que ocurre con máquinas con entorno local (como la KAM). En otras palabras, la MAM es una máquina abstracta razonable.

Teorema 2.31 (Destilación de la MAM). $(\mathrm{MAM}, \mathtt{name}, \equiv, \llbracket\cdot\rrbracket)$ *es una destilería reflectiva. En particular, si S es un estado alcanzable tenemos que:*

1. Búsqueda*: si $S \rightsquigarrow_{\mathtt{s}} S'$ entonces $\llbracket S\rrbracket = \llbracket S'\rrbracket$;*
2. Multiplicativa*: si $S \rightsquigarrow_{\mathtt{m}} S'$ entonces $\llbracket S\rrbracket \to_{\mathtt{m}}\equiv \llbracket S'\rrbracket$;*
3. Exponential*: si $S \rightsquigarrow_{\mathtt{e}} S'$ entonces $\llbracket S\rrbracket \to_{\mathtt{e}}=_{\alpha} \llbracket S'\rrbracket$.*

2.5.3. Call-by-value de izquierda a derecha: la CEK

En esta sección presentamos una adaptación de la KAM para call-by-value, correspondiente a la máquina CEK de Felleisen y Friedman [16] (excluyendo operadores de control). Esta máquina implementa call-by-value de izquierda a derecha.

Los estados de la CEK tienen la misma forma que los estados de la KAM, es decir, están dados por una clausura y una pila. La diferencia es que usan *pilas call-by-value*, cuyos elementos están etiquetados o bien como *argumentos* o bien como *funciones*, de tal modo que la máquina pueda determinar si el código actualmente evaluado es una función que debe ser aplicada al argumento (aún no evaluado) que se encuentra en el tope de la pila, o el argumento para la función (ya evaluada) que se encuentra en el tope de la pila.

Definición 2.32 (Máquina CEK)**.** Las pilas se definen como sigue:

$$\pi \quad ::= \quad \epsilon \mid \mathbf{f}(c) :: \pi \mid \mathbf{a}(c) :: \pi$$

Un estado es una tripla $S = (\bar{t}, e, \pi)$. Las transiciones de la CEK son:

$$\begin{array}{rcccl}
\bar{t}\bar{s} \mid e \mid & \pi & \rightsquigarrow_{\mathtt{s}_1} & \bar{t} \mid e \mid & \mathbf{a}(\bar{s}, e) :: \pi \\
\bar{\mathtt{v}} \mid e \mid & \mathbf{a}(\bar{s}, e') :: \pi & \rightsquigarrow_{\mathtt{s}_2} & \bar{s} \mid e' \mid & \mathbf{f}(\bar{\mathtt{v}}, e) :: \pi \\
\bar{\mathtt{v}} \mid e \mid & \mathbf{f}(\lambda x.\bar{t}, e') :: \pi & \rightsquigarrow_{\mathtt{m}} & \bar{t} \mid [x \backslash (\bar{\mathtt{v}}, e)] :: e' \mid & \pi \\
x \mid e \mid & \pi & \rightsquigarrow_{\mathtt{e}} & \bar{t} \mid e' \mid & \pi
\end{array}$$

donde la transición $\rightsquigarrow_{\mathtt{e}}$ aplica sólo si $e = e'' :: [x \backslash (\bar{t}, e')] :: e'''$.

Definición 2.33 (Decodificación de la CEK)**.** Las pilas se decodifican como sigue:

$$\begin{array}{rcl}
[\![\epsilon]\!] & \stackrel{\text{def}}{=} & \square \\
[\![\mathbf{f}(c) :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle [\![c]\!] \square \rangle \\
[\![\mathbf{a}(c) :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle \square [\![c]\!] \rangle
\end{array}$$

Los estados de la máquina se decodifican exactamente igual que en el caso de la KAM, es decir $[\![\bar{t} \mid e \mid \pi]\!] \stackrel{\text{def}}{=} [\![\pi]\!]\langle [\![e]\!]\langle \bar{t} \rangle\rangle$.

Es posible demostrar que los entornos decodifican a contextos de sustitución, pero para demostrar que $[\![\pi]\!]$ y $[\![\pi]\!]\langle [\![e]\!] \rangle$ son contextos de evaluación, es preciso establecer primero los invariantes dinámicos de la máquina.

Lema 2.34 (Invariantes de la CEK)**.** *Sea $S = \bar{s} \mid e \mid \pi$ un estado alcanzable de la CEK cuyo código inicial $\bar{t}$ es consistente en nombres. Entonces:*

1. Clausura*: todas las clausuras de S son cerradas;*
2. Subtérmino*: cualquier código en S es un subtérmino de $\bar{t}$;*
3. Valor*: cualquier código en e es un valor y, para cada elemento de π de la forma $\mathbf{f}(\bar{s}, e')$, $\bar{s}$ es un valor;*
4. Decodificación contextual*: $[\![\pi]\!]$ y $[\![\pi]\!]\langle [\![e]\!] \rangle$ son contextos de evaluación call-by-value de izquierda a derecha;*
5. Nombre*: cualquier clausura c en S es consistente en nombres y sus nombres son nombres de $\bar{t}$ (*i.e. $\mathsf{supp}(c) \subseteq \mathtt{fv}(\bar{t})$*);*
6. Tamaño del entorno*: la longitud de cualquier entorno en S está acotada por $|\bar{t}|$.*

Teorema 2.35 (Destilación de la CEK)**.** $(\text{CEK}, \mathtt{value}^{\text{LR}}, \equiv, [\![\cdot]\!])$ *es una destilería reflectiva. En particular, si S es un estado alcanzable tenemos que:*

1. Búsqueda 1*: si $S \rightsquigarrow_{\mathtt{s}_1} S'$ entonces $[\![S]\!] \equiv [\![S']\!]$;*
2. Búsqueda 2*: si $S \rightsquigarrow_{\mathtt{s}_2} S'$ entonces $[\![S]\!] = [\![S']\!]$.*
3. Multiplicativa*: si $S \rightsquigarrow_{\mathtt{m}} S'$ entonces $[\![S]\!] \to_{\mathtt{m}} [\![S']\!]$;*
4. Exponencial*: si $S \rightsquigarrow_{\mathtt{e}} S'$ entonces $[\![S]\!] \to_{\mathtt{e}} \equiv [\![S']\!]$;*

2.5.4. Call-by-value de izquierda a derecha: la CEK dividida

Para la máquina CEK demostramos que la pila, que contiene tanto argumentos como funciones, se decodifica a un contexto de evaluación (Lem. 2.34.4). En esta sección estudiamos otra máquina para evaluación call-by-value de izquierda a derecha, llamada la *CEK dividida* (SCEK), que tiene dos pilas: una para argumentos y otra para funciones. Ambas decodifican a contextos de evaluación.

Definición 2.36 (Máquina SCEK)**.** Las pilas se definen como en el caso de la KAM. La sintaxis de los *dumps* está dada por:

$$D ::= \epsilon \mid (c, \pi) :: D$$

Los estados son 4-uplas $(\bar{t}, e, \pi, D)$. Las transiciones de la SCEK son:

$$\begin{array}{lllllllllll}
\bar{t}\bar{s} & e & \pi & D & \rightsquigarrow_{\mathtt{s}_1} & \bar{t} & e & (\bar{s}, e) :: \pi & D \\
\bar{\mathtt{v}} & e & (\bar{t}, e') :: \pi & D & \rightsquigarrow_{\mathtt{s}_2} & \bar{t} & e' & \epsilon & ((\bar{\mathtt{v}}, e), \pi) :: D \\
\bar{\mathtt{v}} & e & \epsilon & ((\lambda x.\bar{t}, e'), \pi) :: D & \rightsquigarrow_{\mathtt{m}} & \bar{t} & [x \backslash (\bar{\mathtt{v}}, e)] :: e' & \pi & D \\
x & e :: [x \backslash (\bar{\mathtt{v}}, e')] :: e'' & \pi & D & \rightsquigarrow_{\mathtt{e}} & \bar{\mathtt{v}} & e' & \pi & D
\end{array}$$

Definición 2.37 (Decodificación de la SCEK)**.** La decodificación de términos, entornos, clausuras y pilas es igual que para la KAM. Un dump se decodifica a un contexto, de acuerdo con:

$$[\![\epsilon]\!] \stackrel{\text{def}}{=} \Box \qquad [\![((\bar{\mathtt{v}}, e), \pi) :: D]\!] \stackrel{\text{def}}{=} [\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\langle \bar{\mathtt{v}}\Box\rangle\rangle\rangle$$

La decodificación de estados se define por: $[\![\bar{t} \mid e \mid \pi \mid D]\!] \stackrel{\text{def}}{=} [\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\langle \bar{t}\rangle\rangle\rangle$.

La máquina SCEK está íntimamente relacionada con la máquina SECD de Landin [28], que también incorpora la noción de dump. En [14], Danvy estudia la máquina SECD y demuestra que la SECD implementa call-by-value de derecha a izquierda (y no call-by-value de izquierda a derecha como la SCEK). El punto en esta sección es ilustrar que "dividir la pila" en una pila de argumentos y un dump es una transformación general.

Lema 2.38 (Invariantes de la SCEK)**.** *Sea $S = \bar{s} \mid e \mid \pi \mid D$ un estado alcanzable de la SCEK cuyo código inicial $\bar{t}$ es consistente en nombres. Entonces:*

1. Clausura: *todas las clausuras de S son cerradas;*

2. Subtérmino: *cualquier código en S es un subtérmino de $\bar{t}$;*

3. Valor: *el código de cualquier clausura en el dump o en cualquier entorno de S es un valor;*

4. Decodificación contextual*: $[\![D]\!]$, $[\![D]\!]\langle [\![\pi]\!]\rangle$ y $[\![D]\!]\langle [\![\pi]\!]\langle [\![e]\!]\rangle\rangle$ son contextos de evaluación call-by-value de izquierda a derecha.*

5. Nombre*: cualquier clausura c en S es consistente en nombres y sus nombres son nombres de $\bar{t}$ (*i.e. $\mathsf{supp}(c) \subseteq \mathtt{fv}(\bar{t})$*).*

6. Tamaño del entorno*: la longitud de cualquier entorno en S está acotada por $|\bar{t}|$.*

Teorema 2.39 (Destilación de la SCEK). $(\text{SCEK}, \texttt{value}^{\texttt{LR}}, \equiv, [\![\cdot]\!])$ *es una destilería reflectiva. En particular, si S es un estado alcanzable tenemos que:*

1. Búsqueda 1*: si $S \rightsquigarrow_{\texttt{s}_1} S'$ entonces $[\![S]\!] \equiv [\![S']\!]$;*
2. Búsqueda 2*: si $S \rightsquigarrow_{\texttt{s}_2} S'$ entonces $[\![S]\!] \equiv [\![S']\!]$;*
3. Multiplicativa*: si $S \rightsquigarrow_{\texttt{m}} S'$ entonces $[\![S]\!] \to_{\texttt{m}} [\![S']\!]$;*
4. Exponencial*: si $S \rightsquigarrow_{\texttt{e}} S'$ entonces $[\![S]\!] \to_{\texttt{e}} \equiv [\![S']\!]$.*

2.5.5. Call-by-value de derecha a izquierda: la LAM

En esta sección presentamos otra adaptación a call-by-value de la KAM, una máquina que llamamos *Leroy Abstract Machine* (LAM) e implementa call-by-value de derecha a izquierda. La LAM debe su nombre a la máquina ZINC de Leroy [29], que implementa evaluación call-by-value de derecha a izquierda. Le otorgamos un nuevo nombre porque la ZINC es una máquina bastante más sofisticada que la LAM: posee un conjunto de instrucciones separado al que los términos se compilan, manipula expresiones aritméticas y evita la creación de clausuras innecesarias usando técnicas que no están capturadas por la LAM. La LAM se puede entender como una variación menor de la CEK; la presentamos sobre todo para enfatizar la modularidad de nuestra metodología basada en contextos.

Definición 2.40 (Máquina abstracta de Leroy). Las pilas y estados son los mismos que los de la CEK. Las transiciones de la LAM son:

$$\begin{array}{rcccccc}
\bar{t}\bar{s} & \mid e \mid & \pi & \rightsquigarrow_{\texttt{s}_1} & \bar{s} \mid & e & \mid \mathbf{f}(\bar{t}, e) :: \pi \\
\bar{\mathrm{v}} & \mid e \mid & \mathbf{f}(\bar{t}, e') :: \pi & \rightsquigarrow_{\texttt{s}_2} & \bar{t} \mid & e' & \mid \mathbf{a}(\bar{\mathrm{v}}, e) :: \pi \\
\lambda x.\bar{t} & \mid e \mid & \mathbf{a}(c) :: \pi & \rightsquigarrow_{\texttt{m}} & \bar{t} \mid & [x \backslash c] :: e \mid & \pi \\
x & \mid e \mid & \pi & \rightsquigarrow_{\texttt{e}} & \bar{t} \mid & e' & \mid \pi
\end{array}$$

donde $\rightsquigarrow_{\texttt{e}}$ aplica sólo si $e = e'' :: [x \backslash (\bar{t}, e')] :: e'''$.

Lema 2.41 (Invariantes de la LAM). *Sea $S = \bar{s} \mid e \mid \pi$ un estado alcanzable de la LAM cuyo código inicial $\bar{t}$ es consistente en nombres. Entonces:*

1. Clausura*: todas las clausuras de S son cerradas;*
2. Subtérmino*: cualquier código en S es un subtérmino de $\bar{t}$;*
3. Valor*: cualquier código en e es un valor y, para cada elemento de π de la forma $\mathbf{a}(\bar{s}, e')$, $\bar{s}$ es un valor;*
4. Decodificación de contextos*: $[\![\pi]\!]$ y $[\![\pi]\!]\langle[\![e]\!]\rangle$ son contextos de evaluación call-by-value de derecha a izquierda;*
5. Nombre*: cualquier clausura c en S es consistente en nombres y sus nombres son nombres de $\bar{t}$ (*i.e. $\mathsf{supp}(c) \subseteq \texttt{fv}(\bar{t})$*);*

6. Tamaño del entorno*: la longitud de cualquier entorno en S está acotada por $|\bar{t}|$.*

Teorema 2.42 (Destilación de la LAM)**.** $(\text{LAM}, \texttt{value}^{\text{RL}}, \equiv, [\![\cdot]\!])$ *es una destilería reflectiva. En particular, si S es un estado alcanzable tenemos que:*

1. Búsqueda 1*: si $S \rightsquigarrow_{\texttt{s}_1} S'$ entonces $[\![S]\!] \equiv [\![S']\!]$;*
2. Búsqueda 2*: si $S \rightsquigarrow_{\texttt{s}_2} S'$ entonces $[\![S]\!] = [\![S']\!]$.*
3. Multiplicativa*: si $S \rightsquigarrow_{\texttt{m}} S'$ entonces $[\![S]\!] \to_{\texttt{m}} [\![S']\!]$;*
4. Exponencial*: si $S \rightsquigarrow_{\texttt{e}} S'$ entonces $[\![S]\!] \to_{\texttt{e}} \equiv [\![S']\!]$;*

2.5.6. Call-by-need: la MAD

En esta sección definimos una nueva máquina abstracta para evaluación call-by-need, llamada la máquina abstracta de Milner by-need (MAD). La MAD surge naturalmente como una reformulación de la estrategia `need` (Def. 2.2) en el marco de las destilerías.

La MAD utiliza entornos globales como la MAM para implementar *memoization*, y utiliza *dumps* como la SCEK para evaluar dentro de sustituciones explícitas.

Definición 2.43 (Máquina abstracta de Milner by-need)**.** Los términos, entornos y pilas se definen igual que para la KAM. Los dumps (D) se definen por:

$$D ::= \epsilon \mid (E, x, \pi) :: D$$

Las transiciones están dadas por:

$$\begin{array}{ccccccccc}
\bar{t}\bar{s} & \pi & D & E & \rightsquigarrow_{\texttt{s}_1} & \bar{t} & \bar{s} :: \pi & D & E \\
\lambda x.\bar{t} & \bar{s} :: \pi & D & E & \rightsquigarrow_{\texttt{m}} & \bar{t} & \pi & D & [x\backslash\bar{s}] :: E \\
x & \pi & D & E_1 :: [x\backslash\bar{t}] :: E_2 & \rightsquigarrow_{\texttt{s}_2} & \bar{t} & \epsilon & (E_1, x, \pi) :: D & E_2 \\
\bar{\texttt{v}} & \epsilon & (E_1, x, \pi) :: D & E_2 & \rightsquigarrow_{\texttt{e}} & \bar{\texttt{v}}^{\alpha} & \pi & D & E_1 :: [x\backslash\bar{\texttt{v}}] :: E_2
\end{array}$$

Definición 2.44 (Decodificación de la MAD)**.** La decodificación de términos, entornos y pilas se define igual que para la KAM. La decodificación de dumps está dada por:

$$[\![\epsilon]\!] \stackrel{\text{def}}{=} \Box \qquad [\![(E, x, \pi) :: D]\!] \stackrel{\text{def}}{=} [\![E]\!]\langle[\![D]\!]\langle[\![\pi]\!]\langle x\rangle\rangle\rangle[x\backslash\Box]$$

La decodificación de estados se define así: $[\![\bar{t} \mid \pi \mid D \mid E]\!] := [\![E]\!]\langle[\![D]\!]\langle[\![\pi]\!]\langle\bar{t}\rangle\rangle\rangle$.

Observar que cuando el código es una variable, se debe ejecutar una transición de *búsqueda*. La idea es que cuando el código es una variable x y el entorno es de la form $E_1 :: [x\backslash\bar{t}] :: E_2$, la máquina debería "saltar" para evaluar $\bar{t}$, guardando el prefijo del entorno E_1, la variable x sobre la cual se sustituirá el resultado de evaluar $\bar{t}$, y la pila π. De hecho esto corresponde la a evaluación *weak head* hereditaria.

Lema 2.45 (Decodificación contextual)**.** *Sean D, π, y E respectivamente un dump, una pila y un entorno global de la MAD. Entonces $[\![D]\!]$, $[\![D]\!]\langle[\![\pi]\!]\rangle$, $[\![E]\!]\langle[\![D]\!]\rangle$ y $[\![E]\!]\langle[\![D]\!]\langle[\![\pi]\!]\rangle\rangle$ son contextos de evaluación call-by-need.*

La noción de *clausura cerrada* se define exactamente igual que para la KAM. Dado un estado $S = \bar{t} \mid \pi \mid D \mid E_0$ con $D = (E_1, x_1, \pi_1) :: \ldots :: (E_n, x_n, \pi_n)$, sus clausuras son $(\llbracket \pi \rrbracket \langle \bar{t} \rangle, E_0)$ y, para cada $i \in \{1, \ldots, n\}$:

$$(\llbracket \pi_i \rrbracket \langle x_i \rangle, E_i :: [x_i \backslash \llbracket \pi_{i-1} \rrbracket \langle x_{i-1} \rangle] :: \ldots :: [x_1 \backslash \llbracket \pi \rrbracket \langle \bar{t} \rangle] :: E_0)$$

Los invariantes dinámicos son:

Lema 2.46 (Invariantes de la MAD). *Sea $S = \bar{t} \mid \pi \mid D \mid E_0$ un estado alcanzable de la MAD cuyo código inicial $\bar{t}$ es consistente en nombres, y tal que $D = (E_1, x_1, \pi_1) :: \ldots :: (E_n, x_n, \pi_n)$. Entonces:*

1. Clausura global: *las clausuras de S son cerradas;*
2. Subtérmino: *cualquier código en S es un subtérmino de $\bar{t}$;*
3. Nombres: *las clausuras de S son consistentes en nombres.*

Teorema 2.47 (Destilación de la MAD). $(\mathrm{MAD}, \mathtt{need}, \equiv_{\mathtt{Need}}, \llbracket \cdot \rrbracket)$ *es una destilería reflectiva. En particular, si S es un estado alcanzable tenemos que:*

1. Búsqueda 1*: si $S \rightsquigarrow_{\mathtt{s}_1} S'$ entonces $\llbracket S \rrbracket = \llbracket S' \rrbracket$;*
2. Búsqueda 2*: si $S \rightsquigarrow_{\mathtt{s}_2} S'$ entonces $\llbracket S \rrbracket = \llbracket S' \rrbracket$;*
3. Multiplicativa*: si $S \rightsquigarrow_{\mathtt{m}} S'$ entonces $\llbracket S \rrbracket \to_{\mathtt{m}} \equiv_{\mathtt{Need}} \llbracket S' \rrbracket$;*
4. Exponencial*: si $S \rightsquigarrow_{\mathtt{e}} S'$ entonces $\llbracket S \rrbracket \to_{\mathtt{e}} =_{\alpha} \llbracket S' \rrbracket$.*

Consideraciones abstractas sobre implementaciones concretas. Considerar la transición $\rightsquigarrow_{\mathtt{s}_2}$. Observar que almacenar el prefijo E_1 en el dump fuerza a implementar E en forma de lista, y por lo tanto a recorrer E de manera secuencial. Este hecho va en contra de la intuición de que E es una memoria (más que una lista) y hace que la MAD sea una máquina abstracta no razonable (comparar con las consideraciones análogoas para la KAM y la MAM). Para resolver este problema, en las secciones siguientes presentamos la MAD con punteros, una variante de la MAD (similar a la máquina de Sestoft para call-by-need [**?**]) que evita el mecanismo de guardar E_1 en una entrada del dump. Esto permite volver a entender el entorno global como una memoria. El rodeo se justifica del siguiente modo:

1. la MAD con punteros es más compleja que la MAD;
2. para el análisis de complejidad de la destilación, es más sencillo razonar acerca de la MAD;

Obesrvar que el asunto sobre las implementaciones concretas es ortogonal al análisis de complejidad del proceso de destilación.

2.5.7. Call-by-need: la MAD fusionada

Al dividir la pila de la CEK en dos, obtuvimos una variante simple de la máquina SECD. En esta sección aplicamos la transformación inversa a la MAD. El resultado es una máquina que llamamod, *MAD fusionada*, que tiene una única pila y se puede entender como una versión sencilla de la KAM *lazy* de Crégut [12] (aunque nuestra definición está inspirada por la presentación de Danvy y Zerny en [15]).

Para diferenciar los dos tipos de objetos en la pila, utilizamos un marcados, al igual que para la CEK y la LAM. Formalmente:

Definición 2.48 (MAD fusionada)**.** Los términos y entornos se definen igual que para la MAM. La sintaxis de las pilas es:

$$\pi ::= \epsilon \mid \mathbf{a}(\overline{t}) :: \pi \mid \mathbf{h}(E, x) :: \pi$$

donde $\mathbf{a}(\overline{t})$ representa un término que se va a utilizar como argumento (igual que en el caso de la CEK) y $\mathbf{h}(E, x, \pi)$ es similar a una entrada del dump de la MAD, aunque en este caso no hay necesidad de guardar la pila. Las transiciones son:

$$\begin{array}{rcccccc}
\overline{t}\overline{s} & \mid \pi & \mid E & \rightsquigarrow_{\mathtt{s}_1} & \overline{t} & \mid \mathbf{a}(\overline{s}) :: \pi & \mid E \\
\lambda x.\overline{t} & \mid \mathbf{a}(\overline{s}) :: \pi & \mid E & \rightsquigarrow_{\mathtt{m}} & \overline{t} & \mid \pi & \mid [x\backslash\overline{s}] :: E \\
x & \mid \pi & \mid E_1 :: [x\backslash\overline{t}] :: E_2 & \rightsquigarrow_{\mathtt{s}_2} & \overline{t} & \mid \mathbf{h}(E_1, x) :: \pi & \mid E_2 \\
\overline{\mathtt{v}} & \mid \mathbf{h}(E_1, x) :: \pi & \mid E_2 & \rightsquigarrow_{\mathtt{e}} & \overline{\mathtt{v}}^{\alpha} & \mid \pi & \mid E_1 :: [x\backslash\overline{\mathtt{v}}] :: E_2
\end{array}$$

Definición 2.49 (Decodificación de la MAD fusionada)**.** La decodificación se define como sigue:

$$\begin{array}{rcl}
[\![\epsilon]\!] & \stackrel{\text{def}}{=} & \Box \\
[\![[x\backslash\overline{t}] :: E]\!] & \stackrel{\text{def}}{=} & [\![E]\!]\langle\Box[x\backslash\overline{t}]\rangle \\
[\![\mathbf{h}(E, x) :: \pi]\!] & \stackrel{\text{def}}{=} & [\![E]\!]\langle[\![\pi]\!]\langle x\rangle\rangle[x\backslash\Box] \\
[\![\mathbf{a}(\overline{t}) :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle\Box\overline{t}\rangle \\
[\![\overline{t} \mid \pi \mid E]\!] & \stackrel{\text{def}}{=} & [\![E]\!]\langle[\![\pi]\!]\langle\overline{t}\rangle\rangle
\end{array}$$

Lema 2.50 (Decodificación contextual)**.** *Sean π y E respectivamente una pila y un entorno global de la MAD fusionada. Entonces $[\![\pi]\!]$ y $[\![E]\!]\langle[\![\pi]\!]\rangle$ son contextos de evaluación call-by-need.*

Los invariantes dinámicos de la MAD fusionada son exactamente los mismos que los de la MAD, con respecto a un conjunto de clausuras asociado un estado definido de manera análoga al de la MAD(se omite la definición exacta).

Teorema 2.51 (Destilación de la MAD fusionada)**.** (MAD fusionada, $\mathtt{need}$, $\equiv_{\mathtt{Need}}$, $[\![\cdot]\!]$) *es una destilería reflectiva. En particular, si S es un estado alcanzable tenemos que:*

1. Búsqueda 1*: si $S \rightsquigarrow_{\mathtt{s}_1} S'$ entonces $[\![S]\!] = [\![S']\!]$;*
2. Búsqueda 2*: si $S \rightsquigarrow_{\mathtt{s}_2} S'$ entonces $[\![S]\!] = [\![S']\!]$;*
3. Multiplicativa*: si $S \rightsquigarrow_{\mathtt{m}} S'$ entonces $[\![S]\!] \rightarrow_{\mathtt{m}} \equiv_{\mathtt{Need}} [\![S']\!]$;*
4. Exponencial*: si $S \rightsquigarrow_{\mathtt{e}} S'$ entonces $[\![S]\!] \rightarrow_{\mathtt{e}} =_{\alpha} [\![S']\!]$.*

2.5.8. Call-by-need: la MAD con punteros

En la MAD, el entorno global se divide entre el entorno de la máquina y las entradas del dump. Esta elección hace que la decodificación sea muy natural. Pero uno querría tener el entorno global en un único lugar, para validar la intuición de que es una memoria más que una lista, de tal modo que el dump sólo contenga variables y pilas. Esto es lo que hacemos en esta sección, aprovechándonos del hecho de que los nombres de variables pueden entenderse como punteros (ver las *consideraciones abstractas* en la Sec. 2.5.2 y la Sec. 2.5.6).

Esta nueva máquina se puede entender como una versión más simple de la *máquina abstracta Sestoft* [?] (SAM). La nueva máquina utiliza una constante simbólica $\Box$ para las sustituciones que ligan una variable que se encuentra en el dump.

Definición 2.52 (La MAD con punteros)**.** Los dumps y entornos se definen de la siguiente manera:

$$\begin{array}{rcl} D & ::= & \epsilon \mid (x,\pi) :: D \\ E & ::= & \epsilon \mid [x\backslash\bar{t}] :: E \mid [x\backslash\Box] :: E \end{array}$$

Las transiciones están dadas por:

$$\begin{array}{ccccccccc} \bar{t}\bar{s} & \pi & D & E & \rightsquigarrow_{\mathtt{s}_1} & \bar{t} & \bar{s} :: \pi & D & E \\ \lambda x.\bar{t} & \bar{s} :: \pi & \epsilon & E & \rightsquigarrow_{\mathtt{m}_1} & \bar{t} & \pi & \epsilon & [x\backslash\bar{s}] :: E \\ \lambda x.\bar{t} & \bar{s} :: \pi & (y,\pi') :: D & E_1 :: [y\backslash\Box] :: E_2 & \rightsquigarrow_{\mathtt{m}_2} & \bar{t} & \pi & (y,\pi') :: D & E_1 :: [y\backslash\Box] :: [x\backslash\bar{s}] :: E_2 \\ x & \pi & D & E_1 :: [x\backslash\bar{t}] :: E_2 & \rightsquigarrow_{\mathtt{s}_2} & \bar{t} & \epsilon & (x,\pi) :: D & E_1 :: [x\backslash\Box] :: E_2 \\ \bar{\mathtt{v}} & \epsilon & (x,\pi) :: D & E_1 :: [x\backslash\Box] :: E_2 & \rightsquigarrow_{\mathtt{e}} & \bar{\mathtt{v}}^{\alpha} & \pi & D & E_1 :: [x\backslash\bar{\mathtt{v}}] :: E_2 \end{array}$$

Observar que hay dos transiciones multiplicativas. Ambas se simulan con pasos multiplicativos, dependiendo de cuál sea el contenido del dump. Una sustitución de la forma $[x\backslash\Box]$ se llama *pendiente*, y en tal caso decimos que la variable x está pendiente.

Observar también que las variables en las entradas del dump D aparecen en orden inverso con respecto a las correspondientes sustituciones en el entorno E. Demostramos que esto en efecto es un invariante, llamado *compatibilidad*.

Definición 2.53 (Compatibilidad $E \propto D$)**.** La relación de compatibilidad $E \propto D$ entre entornos y dumps se define por:

1. $\epsilon \propto \epsilon$;
2. $E :: [x\backslash\bar{t}] \propto D$ si $E \propto D$;
3. $E :: [x\backslash\Box] \propto (x,\pi) :: D$ si $E \propto D$.

Observar que en un par compatible el entorno siempre es al menos tan largo como el dump.

Definición 2.54 (Decodificación de la MAD con punteros)**.** Un par compatible $E \propto D$ se decodifica a un contexto como sigue:

$$\begin{array}{rcl} [\![(E,\epsilon)]\!] & \stackrel{\text{def}}{=} & [\![E]\!] \\ [\![(E :: [x\backslash\Box], (x,\pi) :: D)]\!] & \stackrel{\text{def}}{=} & [\![(E,D)]\!]\langle [\![\pi]\!]\langle x\rangle\rangle[x\backslash\Box] \\ [\![(E :: [x\backslash\bar{t}], (y,\pi) :: D)]\!] & \stackrel{\text{def}}{=} & [\![(E,(y,\pi) :: D)]\!][x\backslash\bar{t}] \end{array}$$

La decodificación de un estado se define así: $[\![\bar{t} \mid \pi \mid D \mid E]\!] := [\![(E, D)]\!]\langle[\![\pi]\!]\langle\bar{t}\rangle\rangle$ bajo la hipótesis de que E y D son compatibles.

El análisis de la MAD con punteros se basa en un invariante complejo que involucra la relación de compatibilidad, además de una generalización del invariante de clausura global. Necesitamos una definición auxiliar:

Definición 2.55 (Sección de un entorno)**.** Dado un entorno E, definimos su *sección* $E\upharpoonleft$ como la secuencia de sustituciones después de la sustitución pendiente más a la derecha. Formalmente:

$$\begin{array}{rcl} \epsilon\upharpoonleft & := & \epsilon \\ (E :: [x\backslash\bar{t}])\upharpoonleft & := & E\upharpoonleft :: [x\backslash\bar{t}] \\ (E :: [x\backslash\Box])\upharpoonleft & := & \epsilon \end{array}$$

Además, si un entorno E es de la forma $E_1 :: [x\backslash\Box] :: E_2$, definimos $E\upharpoonleft_x := E_1\upharpoonleft :: [x\backslash\Box] :: E_2$.

La noción de clausura cerrada con entorno global (Sec. 2.5.2) se extiende a constantes simbólicas $\Box$ de la manera esperada.

Lema 2.56 (Invariantes de la MAD con punteros)**.** *Sea $S = \bar{t} \mid E \mid \pi \mid D$ un estado alcanzable de la Pointing MAD cuyo código inicial $\bar{t}$ es consistente en nombres. Entonces:*

1. Subtérmino: *cualquier código en S es un subtérmino de $\bar{t}$;*
2. Nombres: *la clausura global de S es consistente en nombres;*
3. Compatibilidad dump-entorno:
   - *3.1* $([\![\pi]\!]\langle\bar{t}\rangle, E\upharpoonleft)$ *es cerrado;*
   - *3.2 para cada par* (x, π') *en* D, $([\![\pi']\!]\langle x\rangle, E\upharpoonleft_x)$ *es cerrado;*
   - *3.3 se verifica que* $E \propto D$.
4. Decodificación contextual: $[\![(E, D)]\!]$ *es un contexto de evaluación call-by-need.*

Teorema 2.57 (Destilación de la MAD con punteros)**.** $(\text{MAD con punteros}, \texttt{need}, \equiv_{\texttt{Need}}, [\![\cdot]\!])$ *es una destilería reflectiva. En particular, si S es un estado alcanzable tenemos que:*

1. Búsqueda*: si $S \rightsquigarrow_{\texttt{s}_1} S'$ o $S \rightsquigarrow_{\texttt{s}_2} S'$ entonces $[\![S]\!] = [\![S']\!]$;*
2. Multiplicativa*: si $S \rightsquigarrow_{\texttt{m}_1} S'$ o $S \rightsquigarrow_{\texttt{m}_2} S'$ entonces $[\![S]\!] \rightarrow_{\texttt{m}} \equiv_{\texttt{Need}} [\![S']\!]$;*
3. Exponencial*: si $S \rightsquigarrow_{\texttt{e}} S'$ entonces $[\![S]\!] \rightarrow_{\texttt{e}} =_\alpha [\![S']\!]$;*

2.5.9. Call-by-name fuerte: la MAM fuerte

La máquina que definimos en esta sección implementa call-by-name fuerte, y se puede entender como una versión fuerte de la MAM.

Definición 2.58 (La MAM fuerte)**.** Los conjuntos de *pilas*, *entornos*, *frames* y *fases* se definen como sigue:

$$\begin{array}{llll} \text{Frames} & F & ::= & \epsilon \mid (\bar{t}, \pi) :: F \mid x :: F \\ \text{Entornos} & E & ::= & \epsilon \mid [x \backslash \bar{t}] :: E \mid \rhd x :: E \mid x \lhd :: E \end{array} \qquad \begin{array}{llll} \text{Pilas} & \pi & ::= & \epsilon \mid \bar{t} :: \pi \\ \text{Fases} & \varphi & ::= & \Downarrow \mid \Uparrow \end{array}$$

Los estados de la máquina son 5-uplas $(F, \bar{t}, \pi, E, \varphi)$. Las transiciones están dadas por:

$$\begin{array}{rcccccl} F & \mid \bar{t}\bar{s} & \mid \pi & \mid E \mid \Downarrow & \rightsquigarrow_{\Downarrow \mathtt{s}_1} & F & \mid \bar{t} \mid \bar{s} :: \pi \mid E \mid \Downarrow \\ F & \mid \lambda x.\bar{t} & \mid \bar{s} :: \pi & \mid E \mid \Downarrow & \rightsquigarrow_{\mathtt{m}} & F & \mid \bar{t} \mid \pi \mid [x \backslash \bar{s}] :: E \mid \Downarrow \\ F & \mid \lambda x.\bar{t} & \mid \epsilon & \mid E \mid \Downarrow & \rightsquigarrow_{\Downarrow \mathtt{s}_2} & x :: F & \mid \bar{t} \mid \epsilon \mid \rhd x :: E \mid \Downarrow \\ F & \mid x & \mid \pi & \mid E \mid \Downarrow & \rightsquigarrow_{\mathtt{e}} & F & \mid \bar{t}^{\alpha} \mid \pi \mid E \mid \Downarrow \\ & & & & & & \text{if } E(x) = \bar{t} \\ F & \mid x & \mid \pi & \mid E \mid \Downarrow & \rightsquigarrow_{\Downarrow \mathtt{s}_3} & F & \mid x \mid \pi \mid E \mid \Uparrow \\ & & & & & & \text{if } E(x) = \rhd \\ x :: F & \mid \bar{t} & \mid \epsilon & \mid E \mid \Uparrow & \rightsquigarrow_{\Uparrow \mathtt{s}_4} & F & \mid \lambda x.\bar{t} \mid \epsilon \mid x \lhd :: E \mid \Uparrow \\ (\bar{t}, \pi) :: F & \mid \bar{s} & \mid \epsilon & \mid E \mid \Uparrow & \rightsquigarrow_{\Uparrow \mathtt{s}_5} & F & \mid \bar{t}\bar{s} \mid \pi \mid E \mid \Uparrow \\ F & \mid \bar{t} & \mid \bar{s} :: \pi & \mid E \mid \Uparrow & \rightsquigarrow_{\Uparrow \mathtt{s}_6} & (\bar{t}, \pi) :: F & \mid \bar{s} \mid \epsilon \mid E \mid \Downarrow \end{array}$$

Para el análisis de la máquina, utilizamos las siguientes nociones de frames ordinarios (F), frames débiles (F_{w}), y frames principales (F_{t}), y las siguientes nociones de entornos bien formados (E), entornos débiles (E_{w}), y entornos principales (E_{t}):

Definición 2.59 (Nociones auxiliares de frames y entornos)**.**

Frames ordinarios, débiles y principales	Entornos bien formados, débiles y principales
$F ::= F_{\mathrm{w}} \mid F_{\mathrm{t}} \mid F_{\mathrm{w}} :: F_{\mathrm{t}}$	$E ::= E_{\mathrm{w}} \mid E_{\mathrm{t}} \mid E_{\mathrm{w}} :: E_{\mathrm{t}}$
$F_{\mathrm{w}} ::= \epsilon \mid (\bar{t}, \pi) :: F$	$E_{\mathrm{w}} ::= \epsilon \mid [x \backslash \bar{t}] :: E_{\mathrm{w}} \mid x \lhd :: E_{\mathrm{w}} :: \rhd x :: E'_{\mathrm{w}}$
$F_{\mathrm{t}} ::= \epsilon \mid x :: F$	$E_{\mathrm{t}} ::= \epsilon \mid \rhd x :: E$

Frames débiles y principales. Un frame F se descompone de manera única como $F = F_{\mathrm{w}} :: F_{\mathrm{t}}$, donde $F_{\mathrm{w}} = (\bar{t}_1, \pi_1) :: \cdots :: (\bar{t}_n, \pi_n)$ (con $n \geqslant 0$) es un *frame débil* y F_{t} es un *frame principal.*

Entornos débiles y principales. Los entornos *bien formados* tienen una estructura débil/principal. Se verá que los entornos de estados alcanzables son siempre bien formados como parte del invariante de la máquina.

Acceso a entornos y garbage collection. Dentro de un entorno, los fragmentos de la forma $x \lhd :: E_{\mathrm{w}} :: \rhd x$ serán esencialmente ignorados—esta es una forma de *garbage collection* encapsulada en la decodificación. En particular, dado un entorno bien formado E definimos $E(x)$

así:

$$\begin{array}{rcl} \epsilon(x) & := & \bot \\ ([x\backslash\overline{t}] :: E)(x) & := & \overline{t} \\ ([y\backslash\overline{t}] :: E)(x) & := & E(x) \end{array} \qquad \begin{array}{rcl} (y\lhd :: E_{\mathrm{w}} :: \rhd y :: E)(x) & := & E(x) \\ (\rhd x :: E)(x) & := & \rhd \\ (\rhd y :: E)(x) & := & E(x) \end{array}$$

Escribimos $\Lambda(E)$ para denotar el conjunto de variables ligadas a $\rhd$ en el entorno E, es decir, aquellas variables cuyo alcance no está cerrado por $\lhd$.

Lema 2.60 (Los entornos débiles tienen sólo scopes cerrados)**.** *Si E_{w} es un entorno débil, entonces $\Lambda(E_{\mathrm{w}}) = \varnothing$.*

Consideraciones abstractas sobre implementaciones concretas. La intención es que las variables se implementen como punteros a ubicaciones de memoria, de tal modo que el entorno sea una memoria y el acceso tome tiempo constante en el modelo RAM. En particular, la estructura de lista de los entornos y los delimitadores de scope se utilizan para definir la decodificación, pero no se espera que formen parte de la implementación.

Compatibilidad. En la MAM fuerte, el frame y el entorno guardan información sobre las abstracciones bajo las cuales se está efectuando actualmente la evaluación. Dicha información debe ser coherente—de lo contrario la decodificación de un estado sería imposible. El siguiente predicado de *compatibilidad* describe la correlación que debe existir entre la estructura del frame y la del entorno.

Definición 2.61 (Compatibilidad $F \propto E$)**.** La relación de compatibilidad $F \propto E$ entre frames y entornos se define a través de las siguientes reglas inductivas:

1. *Base*: $\epsilon \propto \epsilon$.
2. *Extensión débil*: $(F_{\mathrm{w}} :: F_{\mathrm{t}}) \propto (E_{\mathrm{w}} :: E_{\mathrm{t}})$ si $F_{\mathrm{t}} \propto E_{\mathrm{t}}$.
3. *Abstracción*: $(x :: F) \propto (\rhd x :: E)$ si $F \propto E$.

Lema 2.62 (Propiedades de la compatibilidad)**.**

1. Entornos bien formados*: si F y E son compatibles entonces E es un entorno bien formado.*
2. Factorización*: todo par compatible $F \propto E$ se puede escribir como $(F_{\mathrm{w}} :: F_{\mathrm{t}}) \propto (E_{\mathrm{w}} :: E_{\mathrm{t}})$ de tal modo que F_{t} es de la forma $F_{\mathrm{t}} = x :: F'$ si y sólo si E_{t} es de la forma $E_{\mathrm{t}} = \rhd x :: E'$.*
3. Los scopes abiertos coinciden*: $\Lambda(F) = \Lambda(E)$.*
4. La compatibilidad conmuta con estructuras débiles*: para todo F_{w} y E_{w}, se tiene que $F \propto E$ si y sólo si $(F_{\mathrm{w}} :: F) \propto (E_{\mathrm{w}} :: E)$.*

Al igual que en las máquinas abstractas anteriores, enunciamos los siguientes invariantes dinámicos de la máquina, que se verifican en todos los estados alcanzables:

Lema 2.63 (Invariantes de la MAM fuerte)**.** *Sea $S = \varphi \mid F \mid \overline{s} \mid \pi \mid E$ un estado alcanzable desde un estado correspondiente a un término inicial $\overline{t}_0$. Entonces:*

1. Compatibilidad: *F y E son compatibles, es decir $F \propto E$.*

2. Forma normal:

   2.1 Código en fase de retroceso*: si $\varphi = \Uparrow$, entonces $\overline{s}$ es normal, y si π es no vacía, entonces $\overline{s}$ es neutral.*

   2.2 Frame*: si $F = F' :: (\overline{u}, \pi') :: F''$, entonces $\overline{u}$ es neutral.*

3. Variables libres en fase de retroceso:

   3.1 Código en fase de retroceso*: si $\varphi = \Uparrow$ entonces $\mathtt{fv}(\overline{s}) \subseteq \Lambda(F)$.*

   3.2 Pares en el Frame*: si $F = F' :: (\overline{u}, \pi') :: F''$ entonces $\mathtt{fv}(\overline{u}) \subseteq \Lambda(F'')$.*

4. Nombre:

   4.1 Sustituciones*: si $E = E' :: [x \backslash \overline{t}] :: E''$ entonces x es fresca con respecto a $\overline{t}$ y E''.*

   4.2 Delimitadores*: si $E = E' :: \rhd x :: E''$ y $F = F' :: x :: F''$ entonces x es fresca con respecto a E'' y F'', y $E'(y) = \bot$ para toda variable y en F''.*

   4.3 Abstracciones*: si $\mathbf{a}x\overline{t}$ es un subtérmino de F, $\overline{s}$, π, o E entonces x puede ocurrir sólo en $\overline{t}$ y en el subentorno cerrado $x\lhd :: E_{\mathrm{w}} :: \rhd x$ de E, si existe.*

5. Clausura:

   5.1 Entorno*: si $E = E' :: [x \backslash \overline{t}] :: E''$ entonces $E''(y) \neq \bot$ para toda $y \in \mathtt{fv}(\overline{t})$.*

   5.2 Código, pila y frame*: $E(x) \neq \bot$ para toda variable libre x que ocurra dentro de $\overline{s}$ y dentro de cualquier código de π y F.*

La definición de la decodificación se basa en la noción de par compatible:

Definición 2.64 (Decodificación de la MAM fuerte)**.** Sea $S = (F, \overline{t}, \pi, E, \varphi)$ un estado tal que $F \propto E$ es un par compatible. Entonces S decodifica a un contexto $\mathtt{C}_S$ y un término $[\![S]\!]$ de la siguiente manera:

- Entornos débiles:

$$
\begin{array}{rcl}
[\![\epsilon]\!] & \stackrel{\text{def}}{=} & \square \\
[\![[x \backslash \overline{s}] :: E_{\mathrm{w}}]\!] & \stackrel{\text{def}}{=} & [\![E_{\mathrm{w}}]\!]\langle \square[x \backslash \overline{s}] \rangle \\
[\![x\lhd :: E_{\mathrm{w}} :: \rhd x :: E'_{\mathrm{w}}]\!] & \stackrel{\text{def}}{=} & [\![E'_{\mathrm{w}}]\!]
\end{array}
$$

- Pares compatibles:

$$
\begin{array}{rcl}
[\![(\epsilon, \epsilon)]\!] & \stackrel{\text{def}}{=} & \square \\
[\![((F_{\mathrm{w}} :: F_{\mathrm{t}}), (E_{\mathrm{w}} :: E_{\mathrm{t}}))]\!] & \stackrel{\text{def}}{=} & [\![(F_{\mathrm{t}}, E_{\mathrm{t}})]\!]\langle [\![E_{\mathrm{w}}]\!]\langle [\![F_{\mathrm{w}}]\!] \rangle \rangle \\
[\![((x :: F), (\rhd x :: E))]\!] & \stackrel{\text{def}}{=} & [\![(F, E)]\!]\langle \lambda x.\square \rangle
\end{array}
$$

- Frames débiles:

$$\begin{array}{rcl} [\![\epsilon]\!] & \stackrel{\text{def}}{=} & \Box \\ [\![(\overline{s},\pi) :: F_{\mathrm{w}}]\!] & \stackrel{\text{def}}{=} & [\![F_{\mathrm{w}}]\!]\langle [\![\pi]\!]\langle \overline{s}\Box\rangle\rangle \end{array}$$

- Pilas:

$$\begin{array}{rcl} [\![\epsilon]\!] & \stackrel{\text{def}}{=} & \Box \\ [\![\overline{s} :: \pi]\!] & \stackrel{\text{def}}{=} & [\![\pi]\!]\langle \Box\overline{s}\rangle \end{array}$$

- Estados:

$$\begin{array}{rcl} \mathtt{C}_S & \stackrel{\text{def}}{=} & [\![(F,E)]\!]\langle [\![\pi]\!]\rangle \\ [\![S]\!] & \stackrel{\text{def}}{=} & \mathtt{C}_S\langle \overline{t}\rangle \end{array}$$

Los siguientes lemas resumen las propiedades de la decodificación:

Lema 2.65 (Los scopes cerrados se borran)**.** *Sea $F \propto E$ un par compatible. Entonces* $[\![(F,(x\lhd :: E_{\mathrm{w}} :: \rhd x :: E))]\!] = [\![(F,E)]\!]$.

Lema 2.66 (Invariante de decodificación LO)**.** *Sea $S = \langle \varphi \mid F \mid \overline{s} \mid \pi \mid E\rangle$ un estado alcanzable. Entonces $[\![(F,E)]\!]$ y $\mathtt{C}_S$ son contextos* LO.

Lema 2.67 (Decodificación y equivalencia estructural $\equiv$)**.**

1. Las pilas y sustituciones conmutan*: si x no ocurre libre en π entonces* $[\![\pi]\!]\langle t[x\backslash s]\rangle \equiv [\![\pi]\!]\langle t\rangle[x\backslash s]$*;*

2. Los pares compatibles absorben sustituciones*: si x no ocurre libre en F entonces* $[\![(F,E)]\!]\langle t[x\backslash s]\rangle \equiv [\![(F,([x\backslash s] :: E))]\!]\langle t\rangle$.

Teorema 2.68 (Destilación de la MAM fuerte)**.** $(\textit{MAM fuerte}, \to_{LO}, \equiv, [\![\,\cdot\,]\!])$ *es una destilería reflectiva. En particular:*

1. Búsqueda 1, 2, 3, 5, 6*: si $S \rightsquigarrow_{\mathtt{s}_{1,2,3,5,6}} S'$ entonces $[\![S]\!] = [\![S']\!]$.*

2. Búsqueda 4*: si $S \rightsquigarrow_{\mathtt{s}_4} S'$ entonces $[\![S]\!] \equiv_{\mathtt{gc}} [\![S']\!]$;*

3. Multiplicative*: si $S \rightsquigarrow_{\mathtt{m}} S'$ entonces $[\![S]\!] \to_{\mathtt{db}} \equiv_{\mathtt{name}^{\mathtt{s}}} [\![S']\!]$;*

4. Exponential*: si $S \rightsquigarrow_{\mathtt{e}} S'$ entonces $[\![S]\!] \to_{\mathtt{ls}} [\![S']\!]$, duplicando el mismo subtérmino.*

2.6. Análisis de complejidad

En esta sección, demostramos que la longitud de una ejecución $\rho : S \rightsquigarrow_{\mathbb{M}}{}^{*} S'$ en cada una de las máquinas abstractas se puede acotar linealmente por la longitud de la derivación destilada $[\![S]\!] \twoheadrightarrow_{\mathbb{S}}\equiv [\![S']\!]$, salvo un factor $|\overline{t}|$ proporcional al tamaño del código inicial $\overline{t}$.

Recordar que las transiciones principales (es decir, multiplicativas y exponenciales) se decodifican como exactamnte un paso en la estrategia de reducción, en tanto que las transiciones no principales (es decir, las de búsqueda) se decodifican como *cero* pasos en la estrategia. Por lo tanto, para obtener una cota para la longitud de la derivación destilada, alcanza con acotar el número de pasos de búsqueda $|\rho|_{\mathtt{s}}$ en una ejecución ρ en función de:

1. el número de pasos principales $|\rho|_{\neg \mathtt{s}}$,
2. el tamaño $|\bar{t}|$ del código inicial $\bar{t}$.

El análisis sólo concierne a las máquinas, pero a través de los teoremas de destilación, permite expresar también la longitud de ejecuciones en la máquina como una función lineal en la longitud de las derivaciones destiladas en la estrategia. Para cada destilería, demostramos que la relación es lineal en los dos parámetros; más concretamente se verifica que $|\rho|_{\mathtt{s}} \in O((|\bar{t}| + 1) \cdot |\rho|_{\neg \mathtt{s}})$.

Definición 2.69. Sea $\mathbb{M}$ una máquina abstracta destilada y sea $\rho : S \rightsquigarrow_{\mathbb{M}}{}^{*} S'$ una ejecución partiendo de un código inicial $\bar{t}$. La máquina $\mathbb{M}$ se dice:

1. **Localmente lineal** si cada vez que $S' \rightsquigarrow_{\mathtt{s}}^{k} S''$ se tiene que $k \in O(|\bar{t}|)$.
2. **Globalmente bilineal** si $|\rho|_{\mathtt{s}} \in O((|\bar{t}| + 1) \cdot |\rho|_{\neg \mathtt{s}})$.

El siguiente resultado asegura que la linealidad local es una condición suficiente para la bilinealidad global.

Proposición 2.70 (Localmente lineal $\Rightarrow$ globalmente bilineal)**.** *Sea $\mathbb{M}$ una máquina abstracta destilada localmente lineal y sea ρ una ejecución partiendo de un código inicial $\bar{t}$. Entonces $\mathbb{M}$ es globalmente bilineal.*

2.6.1. Call-by-name and call-by-value

Es fácil ver que las máquinas call-by-name y call-by-value machines son localmente lineales y por lo tanto globalmente bilineales.

Teorema 2.71 (Bilinealidad para call-by-name y call-by-value)**.** *Las destilerías para la KAM, MAM, CEK, SCEK y LAM son localmente lineales, y así también globalmente bilineales.*

2.6.2. Call-by-need

Las máquinas call-by-need no son localmente lineales, porque una secuencia de transiciones $\rightsquigarrow_{\mathtt{s}_2}$ puede ser tan larga como el entorno global E, es decir, puede no estar acotada por $|\bar{t}|$ sino por el número $|\rho|_{\neg \mathtt{s}}$ de transiciones principales precedentes (como en el caso de la MAM). Adaptar el razonamiento anterior a esta cota mostraría sólo que $|\rho|_{\mathtt{s}}$ es cuadrática en $|\rho|_{\neg \mathtt{s}}$, no lineal. Sin embargo, la linealidad local no es condición necesaria para la bilinealidad global. En efecto, las máquinas call-by-need son globalmente bilineales. La observación clave es que $|\rho|_{s_2}$ está acotado por $|\rho|_p$ no sólo localmente sino también globalmente, como demuestra el siguiente lema.

Hacemos el análisis para la MAD. El razonamiento para la MAD fusionada y la MAD con punteros es análogo. Definimos $|\epsilon| := 0$ y $|(E, x, \pi) :: D| := 1 + |D|$.

Lema 2.72. *Sea $S = \bar{t} \mid \pi \mid D \mid E$ un estado de la MAD alcanzado por la ejecución ρ. Se tiene entonces:*

1. $|\rho|_{s_2} = |\rho|_e + |D|$

2. $|E| + |D| \leqslant |\rho|_m$

3. $|\rho|_{s_2} \leqslant |\rho|_e + |\rho|_m = |\rho|_p$

Teorema 2.73 (Bilinealidad para call-by-need)**.** *La destilería para la MAD es globalmente bilineal.*

2.6.3. Call-by-name fuerte

El análisis de complejidad de la MAM fuerte requiere formular un invariante adicional, que acota el tamaño de los subtérminos que se duplican. En esta subsección, decimos que $\overline{s}$ es un *subtérmino* de $\overline{t}$ si lo es *salvo nombres de variables*, tanto ligadas como libres. Más precisamente: definimos t^- como el resultado de reemplazar, en el término t, todas las variables (incluyendo ligadores) por un símbolo fijo $*$. Decimos entonces que s es un subtérmino de t si s^- es subtérmino de t^- en el sentido usual. La propiedad clave que asegura esta definición es que el tamaño $|\overline{s}|$ de $\overline{s}$ está acotado por $|\overline{t}|$.

Lema 2.74 (Invariante del subtérmino)**.** *Sea ρ una ejecución partiendo de un código inicial $\overline{t}$. Entonces todos los códigos duplicados a lo largo de ρ usando $\rightsquigarrow_{\mathtt{e}}$ son subtérminos de $\overline{t}$.*

Lema 2.75 (Propiedad del subtérmino para $\to_{\text{LO}}$)**.** *Sea π una derivación $\to_{LO}$ desde un término inicial t. Todo término duplicado a lo largo de π usando $\to_{\mathtt{ls}}$ es un subtérmino de t.*

Por último, el siguiente teorema establece que la MAM fuerte es globalmente bilineal.

Teorema 2.76 (Bilinealidad para call-by-name fuerte)**.** *La destilería para la MAM fuerte es globalmente bilineal. Más precisamente, dada una ejecución $\rho : S \rightsquigarrow_{\mathbb{M}^*} S'$ desde un estado inicial con código t se tiene:*

1. Los pasos de búsqueda son bilineales*:* $|\rho|_{\Downarrow\mathtt{s}} \leqslant (1 + |\rho|_e) \cdot |t|$.

2. La búsqueda acota el retroceso*:* $|\rho|_{\Uparrow\mathtt{s}} \leqslant 2 \cdot |\rho|_{\Downarrow\mathtt{s}}$.

3. Los pasos conmutativos son bilineales*:* $|\rho|_s \leqslant 3 \cdot (1 + |\rho|_e) \cdot |t|$.

Capítulo 3

Fundamentos de call-by-need fuerte

3.1. Introducción

Este capítulo es fruto de la colaboración con Thibaut Balabonski, Eduardo Bonelli y Delia Kesner, y se estructura de la siguiente manera. Destacamos en negrita las principales contribuciones:

- En la Sección 3.2 definimos la estrategia call-by-need fuerte. Más específicamente:
  - En la Sección 3.2.1, definimos una teoría de reducción fuerte, la teoría de *sharing* (Def. 3.1).
  - En la Sección 3.2.2, motivamos la definición del call-by-need fuerte, y **definimos una estrategia para reducción call-by-need fuerte** (Def. 3.1), lo que involucra varias nociones relacionadas, tales como formas normales y contextos de evaluación.
  - En la Sección 3.2.3 demostramos cuatro principios básicos que verifica nuestra estrategia call-by-need fuerte, a saber: **alcanza formas normales** (Prop. 3.13), es **determinística** (Prop. 3.15), es una **extensión conservativa** de la noción de call-by-need débil de Ariola *et al.* (Teo. 3.20), y es **correcta** con respecto a la β-reducción (Prop. 3.22).
- En la Sección 3.3 demostramos que la estrategia call-by-need fuerte es *completa* con respecto a la β-reducción (Teo. 3.52). En otras palabras, si un λ-término tiene β-forma normal, entonces la estrategia call-by-need fuerte siempre la alcanza—salvo desenrollado de sustituciones explícitas. La demostración de completitud combina un argumento *lógico* y un argumento *sintáctico*. El argumento lógico se basa en un sistema de tipos auxiliar con tipos intersección no idempotente, y demuestra que la teoría de *sharing* es completa con respecto a la β-reducción. El argumento sintáctico demuestra que la estrategia call-by-need fuerte es completa con respecto a la teoría de *sharing*. Más específicamente:
  - En la Sección 3.3.1, proponemos un **sistema de tipos intersección no idempotente** llamado $\mathcal{HW}$, para la teoría de *sharing* (Def. 3.24). Es una adaptación simple

de sistemas existentes, continuando la línea de trabajo propuesta por Kesner [24]. Demostramos también que **tipabilidad implica normalización** (Teo. 3.40), es decir que los términos tipables en $\mathcal{HW}$ son débilmente normalizantes en la teoría de *sharing*.

- En la Sección 3.3.2, utilizamos el sistema $\mathcal{HW}$ para demostrar que **la teoría de *sharing* es completa** (Prop. 3.42) con respecto a la β-reducción, es decir, que los términos β-normalizantes son normalizantes también en la teoría de *sharing*.
- En Sección 3.3.3, recordamos un resultado de factorización abstracta de Accattoli [1]. Utilizando este resultado abstracto, demostramos que **la estrategia call-by-need fuerte es completa** (Prop. 3.51) con respecto a la teoría de *sharing*. Para hacer esto, mostramos que cualquier secuencia de reducción en la teoría de *sharing* se puede factorizar como un prefijo en la estrategia call-by-need fuerte, seguido por un sufijo cuyos pasos son basura, es decir, pasos dentro de sustituciones explícitas inalcanzables. El corazón de la demostración es un análisis de casos exhaustivo (y delicado) de diagramas de permutación.

3.2. Call-by-need fuerte

En esta sección definimos la estrategia call-by-need fuerte. En la Sección 3.2.1 empezamos definiendo un cálculo que denominamos la teoría de *sharing*. Por "cálculo" nos referimos, formalmente hablando, a un sistema de reescritura. Los objetos de la teoría de *sharing* están dados por el conjunto usual de términos del LSC (variables, abstracciones, aplicaciones y sustituciones explícitas). Los pasos de reescritura de la teoría de *sharing* están dados por una relación de reescritura no determinística $\to_{\mathtt{sh}}$ cuya clausura reflexiva, simétrica y transitiva da lugar a una teoría ecuacional (la relación de equivalencia $=_{\mathtt{sh}}$).

En la Sección 3.2.2 definimos la estrategia call-by-need fuerte. La estrategia se parametriza por un conjunto de variables ϑ, que se consideran *congeladas*. Más precisamente, para cada conjunto ϑ, definimos una relación de reescritura determinística $\overset{\vartheta}{\rightsquigarrow}$ que es un subconjunto de $\to_{\mathtt{sh}}$. La estrategia call-by-need fuerte corresponde al caso en que el conjunto ϑ está vacío, es decir, $\overset{\mathsf{S}}{\rightsquigarrow} \overset{\text{def}}{=} \overset{\varnothing}{\rightsquigarrow}$. En la Sección 3.2.3 estudiamos algunas de sus propiedades básicas, a saber, los principios de **reducción fuerte**, **determinismo**, **conservatividad** y **correctitud**.

3.2.1. La teoría de *sharing*

La estrategia call-by-need fuerte $\overset{\mathsf{S}}{\rightsquigarrow}$ se puede enmarcar en un panorama más amplio, el de la teoría de *sharing*, dada por la relación de reescritura $\to_{\mathtt{sh}}$ que definimos en esta subsección.

Definición 3.1. La *teoría de* sharing $\lambda_{\mathtt{sh}}$ está dada por el conjunto de términos $\mathcal{T}_{\mathtt{sh}}$ del LSC y la relación de reducción $\to_{\mathtt{sh}} \overset{\text{def}}{=} \to_{\mathtt{db}} \cup \to_{\mathtt{lsv}} \cup \to_{\mathtt{gc}}$, donde para cada $\mathcal{R} \in \{\mathtt{db}, \mathtt{lsv}, \mathtt{gc}\}$, $\to_{\mathcal{R}}$ es la clausura por contextos arbitarios de las correspondientes reglas de reescritura definidas

abajo, es decir $\to_{\mathcal{R}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathcal{R}} \rangle$.

$$\begin{array}{rcll} (\lambda x.t)\mathtt{L}\, s & \mapsto_{\mathtt{db}} & t[x\backslash s]\mathtt{L} & \\ \mathtt{C}\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{vL}] & \mapsto_{\mathtt{lsv}} & \mathtt{C}\langle \mathtt{v} \rangle[x\backslash \mathtt{v}]\mathtt{L} & \\ t[x\backslash s] & \mapsto_{\mathtt{gc}} & t & \text{if } x \notin \mathtt{fv}(t) \end{array}$$

Ejemplo 3.2. *La siguiente es una reducción en la teoría de* sharing*:*

$$\begin{array}{rcl} (\lambda x.zxx)((\lambda y.y)(\lambda w.w)) & \to_{\mathtt{sh}} & (\lambda x.zxx)(y[y\backslash \lambda w.w]) \\ & \to_{\mathtt{sh}} & (zxx)[x\backslash y[y\backslash \lambda w.w]] \\ & \to_{\mathtt{sh}} & (zxx)[x\backslash (\lambda w.w)[y\backslash \lambda w.w]] \\ & \to_{\mathtt{sh}} & (zx(\lambda w.w))[x\backslash \lambda w.w][y\backslash \lambda w.w] \\ & \to_{\mathtt{sh}} & (zx(\lambda w.w))[x\backslash \lambda w.w] \\ & \to_{\mathtt{sh}} & (z(\lambda w.w)(\lambda w.w))[x\backslash \lambda w.w] \\ & \to_{\mathtt{sh}} & z(\lambda w.w)(\lambda w.w) \end{array}$$

El siguiente lema caracteriza las formas normales de la teoría de *sharing*. Escribimos $\mathtt{NF}(\to_{\mathtt{sh}})$ para el conjunto de $\to_{\mathtt{sh}}$-formas normales, y $\mathtt{SNF}(\to_{\mathtt{sh}})$ para el conjunto de $\to_{\mathtt{sh}}$-formas normales que no son respuestas, es decir, $t \in \mathtt{SNF}(\to_{\mathtt{sh}})$ si $t \in \mathtt{NF}(\to_{\mathtt{sh}})$ y t no es de la forma $\mathtt{vL}$.

Definición 3.3 (Formas normales de la teoría de *sharing*)**.** El conjunto de $\mathtt{sh}$-*estructuras* ($\overline{\mathtt{S}}$) y el conjunto de $\mathtt{sh}$-*formas normales* ($\overline{\mathtt{N}}$) se definen mutuamente inductivamente como sigue:

$$\frac{}{x \in \overline{\mathtt{S}}} \qquad \frac{t \in \overline{\mathtt{S}} \quad u \in \overline{\mathtt{N}}}{tu \in \overline{\mathtt{S}}} \qquad \frac{t \in \overline{\mathtt{S}}}{t \in \overline{\mathtt{N}}} \qquad \frac{t \in \overline{\mathtt{N}}}{\lambda x.t \in \overline{\mathtt{N}}} \qquad \frac{t \in \mathbb{X} \quad u \in \overline{\mathtt{S}} \quad x \in \mathtt{fv}(t)}{t[x\backslash u] \in \mathbb{X}}$$

En la última regla, el símbolo $\mathbb{X}$ representa o bien $\overline{\mathtt{S}}$ o bien $\overline{\mathtt{N}}$.

Lema 3.4 (Caracterización de formas normales fuertes)**.** *Se verifican las siguientes igualdades:*

- $\mathtt{NF}(\to_{\mathtt{sh}}) = \overline{\mathtt{N}}$
- $\mathtt{SNF}(\to_{\mathtt{sh}}) = \overline{\mathtt{S}}$

3.2.2. La estrategia call-by-need fuerte

En esta subsección definimos una relación de reescritura determinística $\stackrel{\mathsf{S}}{\rightsquigarrow}$ que representa la *estrategia call-by-need fuerte*. En las siguientes subsecciones definimos las relaciones $\stackrel{\vartheta}{\rightsquigarrow}$ y la noción correspondiente de forma normal bajo el conjunto de variables congeladas ϑ. Antes de proseguir, necesitamos algunas definiciones auxiliares, y en particular la noción de *variable no basura*.

Definición 3.5 (Operación de *garbage collection*)**.** La operación de *garbage collection* $\downarrow_{\mathtt{gc}}(t)$ se define como sigue:

$$\begin{array}{rcl} \downarrow_{\mathtt{gc}}(x) & := & x \\ \downarrow_{\mathtt{gc}}(\lambda x.t) & := & \lambda x.\downarrow_{\mathtt{gc}}(t) \\ \downarrow_{\mathtt{gc}}(ts) & := & \downarrow_{\mathtt{gc}}(t)\downarrow_{\mathtt{gc}}(s) \\ \downarrow_{\mathtt{gc}}(t[x\backslash s]) & := & \begin{cases} \downarrow_{\mathtt{gc}}(t)[x\backslash \downarrow_{\mathtt{gc}}(s)] & \text{si } x \in \mathtt{fv}(\downarrow_{\mathtt{gc}}(t)) \\ \downarrow_{\mathtt{gc}}(t) & \text{en caso contrario} \end{cases} \end{array}$$

Definición 3.6 (Variables no basura)**.** El conjunto de *variables no basura* de un término t se define como $\mathsf{ngv}(t) \stackrel{\text{def}}{=} \mathtt{fv}(\downarrow_{\mathtt{gc}}(t))$. Informalmente, $\mathsf{ngv}(t)$ es el conjunto de variables libres de t que no elimina el proceso de recolección de basura. Una variable libre es una *variable basura* si no es no basura.

Lema 3.7 (Caracterización inductiva de variables no basura)**.** *El conjunto* $\mathsf{ngv}(t)$ *de variables no basura se puede caracterizar a través de las siguientes ecuaciones inductivas:*

$$\begin{array}{rcl}
\mathsf{ngv}(x) & = & \{x\} \\
\mathsf{ngv}(\lambda x.t) & = & \mathsf{ngv}(t)\backslash\{x\} \\
\mathsf{ngv}(ts) & = & \mathsf{ngv}(t) \cup \mathsf{ngv}(s) \\
\mathsf{ngv}(t[x\backslash s]) & = & (\mathsf{ngv}(t)\backslash\{x\}) \cup \begin{cases} \mathsf{ngv}(s) & \textit{si } x \in \mathsf{ngv}(t) \\ \varnothing & \textit{en caso contrario} \end{cases}
\end{array}$$

Formas normales y estructuras

La definición del conjunto de formas normales de la estrategia $\stackrel{\vartheta}{\rightsquigarrow}$ depende de la noción clave de *estructura*.

Definición 3.8. El conjunto de *formas normales bajo el conjunto de variables congeladas* ϑ, también llamadas ϑ*-formas normales* (N_ϑ), y el conjunto de *estructuras bajo el conjunto de variables congeladas* ϑ, también llamadas ϑ*-estructuras* (S_ϑ) se definen mutuamente inductivamente a través de las siguientes reglas:

$$\frac{x \in \vartheta}{x \in \mathsf{S}_\vartheta}\ \textsc{n-var} \quad \frac{t \in \mathsf{S}_\vartheta \quad s \in \mathsf{N}_\vartheta}{ts \in \mathsf{S}_\vartheta}\ \textsc{n-app} \quad \frac{t \in \mathsf{S}_\vartheta}{t \in \mathsf{N}_\vartheta}\ \textsc{nfStruct} \quad \frac{t \in \mathsf{N}_{\vartheta \cup \{\mathsf{x}\}}}{\lambda x.t \in \mathsf{N}_\vartheta}\ \textsc{nfLam}$$

$$\frac{t \in \mathbb{X}^{\vartheta \cup \{x\}} \quad s \in \mathsf{S}_\vartheta \quad x \in \mathsf{ngv}(t)}{t[x\backslash s] \in \mathbb{X}^\vartheta}\ \textsc{nfSub} \quad \frac{t \in \mathbb{X}^\vartheta \quad x \notin \mathsf{ngv}(t)}{t[x\backslash s] \in \mathbb{X}^\vartheta}\ \textsc{nfSubG}$$

En las dos últimas reglas, el símbolo $\mathbb{X}$ representa o bien S o bien N.

Contextos de evaluación

La estrategia call-by-need fuerte $\stackrel{\vartheta}{\rightsquigarrow}$ está dada por dos reglas de reducción, que son, respectivamente, instancias de las reglas $\rightarrow_{\mathtt{db}}$ y $\rightarrow_{\mathtt{lsv}}$ de la teoría de *sharing*. Estas reglas se aplican poniendo el foco de evaluación en ubicaciones específicas de un término, lo que se especifica utilizando *contextos de evaluación*.

Definición 3.9. Los conjuntos de *contextos de evaluación bajo el conjunto de variables congeladas* ϑ, también llamados ϑ*-contextos de evaluación* (E_ϑ) y de *contextos de evaluación inertes bajo el conjunto de variables congeladas* ϑ, también llamados ϑ*-contextos de evaluación inertes* ($\mathsf{E}^\circ_\vartheta$) se definen mutuamente inductivamente a través de las siguientes reglas:

$$\frac{}{\square \in \mathsf{E}^\circ_\vartheta}\ \textsc{EBox} \quad \frac{\mathsf{C} \in \mathsf{E}^\circ_\vartheta}{\mathsf{C}\,t \in \mathsf{E}^\circ_\vartheta}\ \textsc{EAppL} \quad \frac{t \in \mathsf{S}_\vartheta \quad \mathsf{C} \in \mathsf{E}_\vartheta}{t\,\mathsf{C} \in \mathsf{E}^\circ_\vartheta}\ \textsc{EAppRStr}$$

$$\frac{\mathtt{C} \in \mathsf{E}^{\circ}_{\vartheta}}{\mathtt{C} \in \mathsf{E}_{\vartheta}}\ \textsc{e-incl} \quad \frac{\mathtt{C} \in \mathsf{E}_{\vartheta \cup \{x\}}}{\lambda x.\mathtt{C} \in \mathsf{E}_{\vartheta}}\ \textsc{ELam} \quad \frac{\mathtt{C} \in \mathbb{X}^{\vartheta} \quad t \notin \mathsf{S}_{\vartheta} \quad x \notin \vartheta}{\mathtt{C}[x\backslash t] \in \mathbb{X}^{\vartheta}}\ \textsc{ESubLNonStr}$$

$$\frac{\mathtt{C} \in \mathbb{X}^{\vartheta \cup \{x\}} \quad t \in \mathsf{S}_{\vartheta}}{\mathtt{C}[x\backslash t] \in \mathbb{X}^{\vartheta}}\ \textsc{ESubLStr} \quad \frac{\mathtt{C}_1 \in \mathbb{X}^{\vartheta} \quad \mathtt{C}_2 \in \mathsf{E}^{\circ}_{\vartheta}}{\mathtt{C}_1\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{C}_2] \in \mathbb{X}^{\vartheta}}\ \textsc{ESubsR}$$

En las últimas tres reglas, el símbolo $\mathbb{X}$ representa o bien E o bien E°.

Reducción

Estamos en condiciones de definir la estrategia call-by-need fuerte como una relación binaria $\overset{\vartheta}{\rightsquigarrow}$.

Definición 3.10 (Reducción call-by-need fuerte)**.** La estrategia call-by-need fuerte $\overset{\vartheta}{\rightsquigarrow}$ está dada por la unión de las reglas de reducción $\overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}}$ y $\overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}}$ definidas abajo:

$$\begin{array}{rcll}
\mathtt{C}\langle(\lambda x.t)\mathtt{L}\, s\rangle & \overset{\vartheta}{\rightsquigarrow}_{\mathtt{db}} & \mathtt{C}\langle t[x\backslash s]\mathtt{L}\rangle & \text{si } \mathtt{C} \in \mathsf{E}_{\vartheta} \\
\mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x \rangle\!\rangle[x\backslash \mathtt{vL}]\rangle & \overset{\vartheta}{\rightsquigarrow}_{\mathtt{lsv}} & \mathtt{C}_1\langle \mathtt{C}_2\langle \mathtt{v}\rangle[x\backslash \mathtt{v}]\mathtt{L}\rangle & \text{si } \mathtt{C}_1\langle \mathtt{C}_2\langle\Box\rangle[x\backslash \mathtt{vL}]\rangle \in \mathsf{E}_{\vartheta}
\end{array}$$

Ejemplo 3.11. *La siguiente es una reducción en call-by-need fuerte. En cada paso subrayamos el foco de evaluación:*

$$\begin{array}{rcl}
\underline{(\lambda x.xx)(\lambda y.z(Iz)y)} & \overset{\{z\}}{\rightsquigarrow} & (\underline{x}x)[x\backslash \lambda y.z(Iz)y] \\
 & \overset{\{z\}}{\rightsquigarrow} & (\underline{(\lambda y'.z(Iz)y')x})[x\backslash \lambda y.z(Iz)y] \\
 & \overset{\{z\}}{\rightsquigarrow} & (z(\underline{Iz})y')[y'\backslash x][x\backslash \lambda y.z(Iz)y] \\
 & \overset{\{z\}}{\rightsquigarrow} & (z(w[w\backslash z])y')[y'\backslash \underline{x}][x\backslash \lambda y.z(Iz)y] \\
 & \overset{\{z\}}{\rightsquigarrow} & (z(w[w\backslash z])\underline{y'})[y'\backslash \lambda y.z(Iz)y][x\backslash \lambda y.z(Iz)y] \\
 & \overset{\{z\}}{\rightsquigarrow} & (z(w[w\backslash z])(\lambda y''.z(\underline{Iz})y''))[y'\backslash \lambda y.z(Iz)y][x\backslash \lambda y.z(Iz)y] \\
 & \overset{\{z\}}{\rightsquigarrow} & (z(w[w\backslash z])(\lambda y''.z(w[w\backslash z])y''))[y'\backslash \lambda y.z(Iz)y][x\backslash \lambda y.z(Iz)y]
\end{array}$$

3.2.3. Propiedades básicas del call-by-need fuerte

En cada una de las subsecciones de esta sección, demostramos cuatro principios básicos con los que cumple la estrategia call-by-need fuerte: **reducción fuerte** (Prop. 3.13), **determinismo** (Prop. 3.15), **conservatividad** (Teo. 3.20) y **correctitud** (Prop. 3.22). El quinto principio que buscamos, **completitud**, es mucho más complejo y postergamos su enunciado y demostración hasta la próxima sección.

Reducción fuerte

El siguiente lema auxiliar caracteriza el conjunto de formas normales de la estrategia $\overset{\vartheta}{\rightsquigarrow}$.

Lema 3.12 (Caracterización de ϑ-formas normales)**.** *Los siguientes conjuntos son iguales:*

- *El conjunto de ϑ-formas normales N_{ϑ} (cf. Def. 3.8).*

- *El conjunto de formas normales de la estrategia call-by-need fuerte* $\overset{\vartheta}{\rightsquigarrow}$.

La estrategia call-by-need fuerte alcanza formas normales, salvo el desenrollado de sustituciones explícitas:

Proposición 3.13 (Reducción fuerte)**.** *Si t es un término en $\overset{\vartheta}{\rightsquigarrow}$-forma normal, entonces el término desenrollado $t^{\diamond}$ es un λ-término en β-forma normal.*

Determinismo

El siguiente lema auxiliar afirma que hay un único foco de evaluación:

Lema 3.14 (Descomposición única)**.** *Si $\mathtt{C}\langle r\rangle$ es un término, decimos que r es un* ancla *si es un* db*-redex o una variable ligada a una respuesta. Sea t un término que se puede escribir como $\mathtt{C}_1\langle r_1\rangle$ y también como $\mathtt{C}_2\langle r_2\rangle$, donde $\mathtt{C}_1, \mathtt{C}_2 \in \mathsf{E}_\vartheta$ son contextos de evaluación y r_1, r_2 son anclas. Entonces $\mathtt{C}_1 = \mathtt{C}_2$ y $r_1 = r_2$.*

La estrategia call-by-need fuerte es determinística:

Proposición 3.15 (Determinismo)**.** *Si $t \overset{\vartheta}{\rightsquigarrow} s$ y $t \overset{\vartheta}{\rightsquigarrow} u$ entonces $s = u$.*

Conservatividad

En esta subsección demostramos que la estrategia call-by-need fuerte es conservativa sobre el call-by-need débil. Para ello, relacionamos la estrategia call-by-need fuerte con la estrategia call-by-need débil $\overset{\mathsf{W}}{\rightsquigarrow}$, como así también con la noción original de reducción call-by-need débil de [6, 5].

Definición 3.16 (La estrategia call-by-need débil de Ariola *et al.*)**.** La sintaxis del sistema de [6, 5] está dada por los siguientes conjuntos de términos (t), valores ($\mathtt{v}$), respuestas (a) y contextos de evaluación call-by-need débil ($\mathtt{E}$):

$$\begin{array}{rcl} t & ::= & x \mid \lambda x.t \mid t\,t \mid t[x\backslash t] \\ \mathtt{v} & ::= & \lambda x.t \\ a & ::= & \mathtt{v} \mid a[x\backslash t] \\ \mathtt{E} & ::= & \Box \mid \mathtt{E}\,t \mid \mathtt{E}\,t \mid \mathtt{E}\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{E}] \end{array}$$

El sistema cuenta con cuatro reglas de reescritura:

$$\begin{array}{rcll} (\lambda x.t)\,s & \overset{\mathrm{R}}{\mapsto}_{\mathrm{I}} & t[x\backslash s] & \\ \mathtt{E}\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{v}] & \overset{\mathrm{R}}{\mapsto}_{\mathrm{V}} & \mathtt{E}\langle \mathtt{v}\rangle[x\backslash \mathtt{v}] & \\ a[x\backslash t]\,s & \overset{\mathrm{R}}{\mapsto}_{\mathrm{C}} & (a\,s)[x\backslash t] & \text{if } x \notin \mathtt{fv}(s) \\ \mathtt{E}\langle\!\langle x\rangle\!\rangle[x\backslash a[y\backslash t]] & \overset{\mathrm{R}}{\mapsto}_{\mathrm{A}} & \mathtt{E}\langle\!\langle x\rangle\!\rangle[x\backslash a][y\backslash t] & \text{if } y \notin \mathtt{fv}(\mathtt{E}\langle\!\langle x\rangle\!\rangle) \end{array}$$

La reducción se define por: $\mapsto_{\mathrm{need}} \overset{\mathrm{def}}{=} \mapsto_{\mathrm{I}} \cup \mapsto_{\mathrm{V}} \cup \mapsto_{\mathrm{C}} \cup \mapsto_{\mathrm{A}}$, donde $\mapsto_{\mathtt{X}}$ es la clausura por contextos de evaluación de $\overset{\mathrm{R}}{\mapsto}_{\mathtt{X}}$, es decir $\mapsto_{\mathtt{X}} \overset{\mathrm{def}}{=} \mathtt{E}\langle \overset{\mathrm{R}}{\mapsto}_{\mathtt{X}}\rangle$ para cada $\mathtt{X} \in \{\mathrm{I}, \mathrm{V}, \mathrm{C}, \mathrm{A}\}$.

La relación $\mapsto_{\mathrm{need}}$ resulta ser determinística:

Proposición 3.17 (Determinismo de $\mapsto_{\text{need}}$). *Si $t \mapsto_{\text{need}} s$ entonces hay un único contexto $\mathtt{E}$ tal que $t = \mathtt{E}\langle t' \rangle$, $s = \mathtt{E}\langle s' \rangle$ y $t' \mapsto s'$, donde $\mapsto \stackrel{\text{def}}{=} \overset{R}{\mapsto}_{\text{I}} \cup \overset{R}{\mapsto}_{\text{V}} \cup \overset{R}{\mapsto}_{\text{C}} \cup \overset{R}{\mapsto}_{\text{A}}$.*

Demostración. Ver [6, Lemma 4.2]. □

Usando esta propiedad, se puede observar que toda reducción en $\mapsto_{\text{need}}$ se organiza en fragmentos de la forma $\mapsto_{\text{C}}^* \mapsto_{\text{I}}$ o de la forma $\mapsto_{\text{A}}^* \mapsto_{\text{V}}$:

Lema 3.18 (Organización de reducciones $\mapsto_{\text{need}}$).

$$\mapsto_{\text{need}}^* \quad = \quad ((\mapsto_{\text{C}}^* \mapsto_{\text{I}}) \cup (\mapsto_{\text{A}}^* \mapsto_{\text{V}}))^* (\mapsto_{\text{C}}^* \cup \mapsto_{\text{A}}^*)$$

Por otra parte, la estrategia de evaluación call-by-need débil $\overset{\mathsf{W}}{\rightsquigarrow}$ tiene la misma sintaxis que el sistema de Ariola *et al.* pero utiliza otras reglas de reescritura. Y la estrategia call-by-need débil $\overset{\mathsf{W}}{\rightsquigarrow}$ está dada por las siguientes reglas de reescritura, cerradas bajo contextos de evaluación call-by-need débil:

$$\begin{array}{rcl} (\lambda x.t)\mathtt{L}\, s & \overset{\mathsf{W}}{\rightsquigarrow}_{\mathtt{db}} & t[x \backslash s]\mathtt{L} \\ \mathtt{E}\langle\!\langle x \rangle\!\rangle[x \backslash \mathtt{vL}] & \overset{\mathsf{W}}{\rightsquigarrow}_{\mathtt{lsv}} & \mathtt{E}\langle\!\langle \mathtt{v} \rangle\!\rangle[x \backslash \mathtt{v}]\mathtt{L} \end{array}$$

El conjunto de términos dado por la gramática $\mathsf{N}_{\vartheta}^{\mathsf{w}} ::= \mathtt{vL} \mid \mathtt{E}\langle\!\langle x \rangle\!\rangle$ for $x \in \vartheta$ caracteriza el conjunto de formas normales con respecto a la estrategia de evaluación débil.

Es inmediato concluir que la relación $\overset{\mathsf{W}}{\rightsquigarrow}$ está incluida en $\mapsto_{\text{need}}$, en particular, un paso $\mathtt{db}$ (resp. $\mathtt{lsv}$) se traduce a un fragmento $\mapsto_{\text{C}}^* \mapsto_{\text{I}}$ (resp. $\mapsto_{\text{A}}^* \mapsto_{\text{V}}$). De hecho, estos fragmentos caracterizan $\overset{\mathsf{W}}{\rightsquigarrow}$:

Lema 3.19 (Descomposición de $\overset{\mathsf{W}}{\rightsquigarrow}$).

$$\overset{\mathsf{W}}{\rightsquigarrow} \quad = \quad (\mapsto_{\text{C}}^* \mapsto_{\text{I}}) \cup (\mapsto_{\text{A}}^* \mapsto_{\text{V}})$$

El principio de **conservatividad** afirma que nuestra estrategia call-by-need fuerte es conservativa con respecto a $\overset{\mathsf{W}}{\rightsquigarrow}$, es decir, que $t \overset{\mathsf{W}}{\rightsquigarrow} s$ implica $t \overset{\mathsf{S}}{\rightsquigarrow} s$. Más precisamente, si escribimos $\overset{\vartheta \backslash \mathsf{W}}{\rightsquigarrow}$ para denotar $\overset{\vartheta}{\rightsquigarrow} \backslash \overset{\mathsf{W}}{\rightsquigarrow}$, tenemos:

Teorema 3.20 (Conservatividad). *Si $t_0 \overset{\vartheta}{\rightsquigarrow} t_1 \overset{\vartheta}{\rightsquigarrow} \dots t_{n-1} \overset{\vartheta}{\rightsquigarrow} t_n$ existe un $1 \leqslant i \leqslant n$ tal que se verifican las tres condiciones siguientes:*

1. $t_0 \overset{\mathsf{W}}{\rightsquigarrow} t_1 \overset{\mathsf{W}}{\rightsquigarrow} \dots t_{n-1} \overset{\mathsf{W}}{\rightsquigarrow} t_i$
2. $t_i \overset{\vartheta \backslash \mathsf{W}}{\rightsquigarrow} t_{i+1} \overset{\vartheta \backslash \mathsf{W}}{\rightsquigarrow} \dots t_{n-1} \overset{\vartheta \backslash \mathsf{W}}{\rightsquigarrow} t_n$
3. *Si $i < n$, entonces $t_j \in \mathsf{N}_{\vartheta}^{\mathsf{w}}$ para todo $i \leqslant j \leqslant n$.*

Como corolario de Teo. 3.20 y Lem. 3.19, podemos deducir que nuestra estrategia $\overset{\vartheta}{\rightsquigarrow}$ tiene como prefijo a la noción de reducción call-by-need débil de Ariola *et al.*.

Corolario 3.21. *Si $t\ (\overset{\vartheta}{\rightsquigarrow})^*\ s$ entonces hay un término u tal que*

$$t\ ((\mapsto_{\text{C}}^* \mapsto_{\text{I}}) \cup (\mapsto_{\text{A}}^* \mapsto_{\text{V}}))^*\ u\ (\overset{\vartheta \backslash \mathsf{W}}{\rightsquigarrow})^*\ s$$

Más aún, si $s \in \mathsf{N}_{\vartheta}$, entonces u es una forma normal para $\mapsto_{\text{need}}$ salvo un número finito de pasos $\mapsto_{\text{C}} \cup \mapsto_{\text{A}}$.

Correctitud

Para finalizar esta sección, observamos que la estrategia call-by-need fuerte $\overset{\vartheta}{\rightsquigarrow}$ es correcta con respecto a la β-reducción:

Proposición 3.22 (Correctitud). *Si* $t \overset{\vartheta}{\rightsquigarrow} s$ *entonces* $t^\diamond =_\beta s^\diamond$.

3.3. Completitud del call-by-need fuerte

En esta sección nos dedicamos a demostrar el principio de **completitud** para nuestra estrategia call-by-need fuerte. Por *completitud* nos referimos a completitud con respecto a la β-reducción, en el sentido de que si un término t admite una β-forma normal s en el cálculo-λ, entonces la estrategia $\overset{\mathsf{S}}{\rightsquigarrow}$ computa una forma normal u, y que además dichas formas normales tienen una relación precisa, más específicamente $u^\diamond = s$.

Adoptamos las ideas de Kesner [23] usadas para call-by-need débil para dar una prueba de completitud de call-by-need fuerte.

Supongamos que $t =_\beta s$ son términos interconvertibles en el cálculo-λ, y supongamos que s es una β-forma normal. Por confluencia del cálculo-λ hay una reducción $t \twoheadrightarrow_\beta s$. La completitud de la estrategia call-by-need fuerte consistiría en ver que en tal caso siempre existe un término u tal que $t \overset{\vartheta}{\rightsquigarrow}{}^* u$ y $u^\diamond = s$. Descomponemos la demostración de completitud en dos partes:

1. **Completitud de la teoría de *sharing*.** En primer lugar, demostramos que la teoría de *sharing* $\to_{\mathtt{sh}}$ es completa con respecto a la β-reducción. De este hecho se desprende que hay un término r tal que $t \twoheadrightarrow_{\mathtt{sh}} r$ y $r^\diamond = s$.

2. **Factorización de la teoría de *sharing*.** En segundo lugar, demostramos que cualquier reducción en la teoría de *sharing* $\to_{\mathtt{sh}}$ se puede factorizar como un prefijo de pasos *externos* (es decir, una secuencia de pasos en la estrategia $\overset{\vartheta}{\rightsquigarrow}$) seguido por un sufijo de pasos *internos* que preservan el desenrollado de sustituciones explícitas. De este hecho se desprende que hay un término u tal que $t \overset{\vartheta}{\rightsquigarrow}{}^* u$ y tal que $u^\diamond = r^\diamond = s$.

La descomposición se representa gráficamente en Figura 3.1.

Las subsecciones que siguen se estructuran de la siguiente manera:

- En la Sección 3.3.1, recordamos el sistema de tipos intersección no idempotente $\mathcal{HW}$ de [24]. Además, damos un resultado que relaciona la normalización débil en la teoría de *sharing* con tipabilidad en el sistema $\mathcal{HW}$.

- En la Sección 3.3.2, demostramos la completitud de la teoría de *sharing*, como se muestra en la Figura 3.1(b). La demostración utiliza la tipabilidad en el sistema $\mathcal{HW}$ como un paso intermedio.

- En la Sección 3.3.2, demostramos un resultado de factorización para la teoría de *sharing*, como se muestra en Figura 3.1(c). Escribiendo $\xrightarrow{\neg\vartheta}_{\mathtt{sh}}$ para denotar la relación $\to_{\mathtt{sh}} \setminus \overset{\vartheta}{\rightsquigarrow}$, la demostración se basa en intercambiar repetidamente pares de pasos $t \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \overset{\vartheta}{\rightsquigarrow} s$ para escribirlos como $t \overset{\vartheta}{\rightsquigarrow} \twoheadrightarrow_{\mathtt{sh}} s$.

(a)	$\begin{tikzcd} t \arrow[r, two heads, "\beta"] \arrow[d, squiggly, "\vartheta"'] & \mathtt{nf}_\beta \\ \mathtt{nf}_\vartheta \arrow[ur, dotted, "\diamond"'] & \end{tikzcd}$	**Completitud de call-by-need fuerte** Si $t \twoheadrightarrow_\beta s \in \mathtt{NF}(\to_\beta)$ entonces existe un término u tal que $t \overset{\vartheta}{\rightsquigarrow}{}^* u \in \mathsf{N}_\vartheta$ y $u^\diamond = s$, donde $\vartheta = \mathtt{fv}(t)$. (Ver Teo. 3.52).
(b)	$\begin{tikzcd} t \arrow[r, two heads, "\beta"] \arrow[dr, "\mathtt{sh}"'] & \mathtt{nf}_\beta \\ & \mathtt{nf}_{\mathtt{sh}} \arrow[u, dotted, "\diamond"'] \end{tikzcd}$	**Completitud de la teoría de *sharing*** Si $t \twoheadrightarrow_\beta s \in \mathtt{NF}(\to_\beta)$ entonces existe un término u tal que $t \twoheadrightarrow_{\mathtt{sh}} u \in \mathtt{NF}(\to_{\mathtt{sh}})$ y $u^\diamond = s$. (Ver Prop. 3.42).
(c)	$\begin{tikzcd} t \arrow[d, squiggly, "\vartheta"'] \arrow[dr, "\mathtt{sh}"] & \\ \mathtt{nf}_\vartheta \arrow[r, dotted, "\diamond"'] & \mathtt{nf}_{\mathtt{sh}} \end{tikzcd}$	**Factorización de la teoría de *sharing*** Si $t \twoheadrightarrow_{\mathtt{sh}} s \in \mathtt{NF}(\to_{\mathtt{sh}})$ entonces existe un término u tal que $t \overset{\vartheta}{\rightsquigarrow}{}^* u \in \mathsf{N}_\vartheta$ y $u^\diamond = s^\diamond$, donde $\vartheta = \mathtt{fv}(t)$. (Ver Prop. 3.51).

Figura 3.1: Descomposición de la demostración de **completitud**: (b) y (c) implican (a)

3.3.1. El sistema de tipos intersección no idempotente $\mathcal{HW}$

Definición 3.23 (Sintaxis del sistema $\mathcal{HW}$)**.** Dado un conjunto infinito numerable $\mathcal{B}$ de *tipos básicos* $\alpha, \beta, \gamma, \ldots$, el conjunto de *tipos* y *multiconjuntos de tipos* se definen mutuamente inductivamente con la siguiente gramática:

Tipos	τ, σ, ρ	$::=$	$\alpha \mid \mathcal{M} \to \tau$	
Multiconjuntos de tipos	$\mathcal{M}$	$::=$	$[\tau_i]_{i \in I}$	donde I es un conjunto finito

Una *asignación de tipos* o *contexto de tipado*, escrita Γ, Δ, etc., es una función que a cada variable le asigna un multiconjunto de tipos. El dominio de Γ se define por $\mathsf{dom}(\Gamma) := \{x \mid \Gamma(x) \neq [\,]\}$. Suponemos que los contextos de tipado tienen *dominio finito*.

La *unión de contextos de tipado*, escrita $\Gamma + \Delta$, es el contexto de tipado definida por $(\Gamma + \Delta)(x) := \Gamma(x) + \Delta(x)$, donde el símbolo $+$ denota la unión (aditiva) de multiconjuntos. Observar que $\mathsf{dom}(\Gamma + \Delta) = \mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Delta)$. Escribimos $\Gamma \oplus \Delta$ para denotar $\Gamma + \Delta$ cuando $\mathsf{dom}(\Gamma)$ y $\mathsf{dom}(\Delta)$ son disjuntos. Escribimos $\Gamma +_{i \in I} \Delta_i$ para abreviar $\Gamma + \sum_{i \in I} \Delta_i$. La *inclusión entre contextos de tipado*, escrita $\Gamma \sqsubseteq \Delta$, se verifica si para cada variable x se tiene que $\Gamma(x) \sqsubseteq \Delta(x)$.

Por ejemplo $(x : [\sigma], y : [\tau]) + (x : [\sigma], z : [\sigma]) = x : [\sigma, \sigma], y : [\tau], z : [\sigma]$, y $x : [\sigma] \sqsubseteq x : [\sigma, \sigma], y : \rho$.

Definición 3.24 (El sistema $\mathcal{HW}$, [24])**.** Los *juicios de tipado* son de la forma $\Gamma \vdash t : \tau$, donde Γ es un contexto de tipado, t es un término y τ es un tipo. El *sistema de tipos* $\mathcal{HW}$ está dado

por las siguientes reglas:

$$\frac{}{x{:}[\tau] \vdash x : \tau}\ \textsc{t-var} \qquad \frac{\Gamma \vdash t : [\sigma_i]_{i\in I} \to \tau \quad (\Delta_i \vdash s : \sigma_i)_{i\in I}}{\Gamma +_{i\in I} \Delta_i \vdash t\, s : \tau}\ \textsc{t-app}$$

$$\frac{\Gamma \oplus (x : \mathcal{M}) \vdash t : \tau}{\Gamma \vdash \lambda x.t : \mathcal{M} \to \tau}\ \textsc{t-lam} \qquad \frac{\Gamma \oplus (x : [\sigma_i]_{i\in I}) \vdash t : \tau \quad (\Delta_i \vdash s : \sigma_i)_{i\in I}}{\Gamma +_{i\in I} \Delta_i \vdash t[x\backslash s] : \tau}\ \textsc{t-sub}$$

Si se restringe el sistema $\mathcal{HW}$ únicamente a λ-términos, de tal modo que sólo se utilicen las reglas (T-VAR), (T-LAM) y (T-APP), se obtiene el sistema presentado en [18, 10], que aquí llamamos *sistema-λ*. A continuación recordamos la definición usual de derivación de tipos:

Definición 3.25 (Derivaciones)**.** Una *derivación (de tipos)* es un árbol finito que se obtiene aplicando las reglas inductivas del sistema de tipos. Escribimos $\Phi \rhd \Gamma \vdash t : \tau$ si Φ termina en el juicio $\Gamma \vdash t : \tau$. En tal caso decimos que Φ *le da tipo* a t. Escribimos $\Phi \rhd_\lambda \Gamma \vdash t : \tau$ si, además, Φ es una derivación válida en el sistema-λ. Una derivación Φ' es una *subderivación inmediata* de Φ si Φ' es uno de los hijos de Φ. Un término t es *tipable* si hay una derivación que le da tipo a t. El *tamaño* de una derivación de tipos Φ es un número natural $\mathtt{size}(\Phi)$ que denota la cantidad de nodos del árbol Φ.

El siguientes es un ejemplo de derivación de tipos en el sistema $\mathcal{HW}$.

Ejemplo 3.26 (Una derivación de tipos en $\mathcal{HW}$)**.** *Sea Ω el término $(\lambda z.zz)(\lambda z.zz)$. Sea además $\tau = [\sigma] \to \sigma$, donde σ es un tipo arbitrario. Sea π la siguiente derivación:*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\tau] \vdash x : [\sigma] \to \sigma}\ \textsc{t-var} \qquad \dfrac{}{x : [\sigma] \vdash x : \sigma}\ \textsc{t-var}}{x : [\tau, \sigma] \vdash xx : \sigma}\ \textsc{t-app}}{x : [\tau, \sigma] \vdash \lambda y.xx : [\ ] \to \sigma}\ \textsc{t-lam}}{x : [\tau, \sigma] \vdash (\lambda y.xx)\Omega : \sigma}\ \textsc{t-app}$$

Entonces se tiene que:

$$\dfrac{\dfrac{\vdots}{\pi} \qquad \dfrac{}{z : [\tau] \vdash z : \tau}\ \textsc{t-var} \qquad \dfrac{}{z : [\sigma] \vdash z : \sigma}\ \textsc{t-var}}{z : [\tau, \sigma] \vdash ((\lambda y.xx)\Omega)[x\backslash z] : \sigma}\ \textsc{t-sub}$$

El sistema $\mathcal{HW}$ tiene la siguiente propiedad:

Lema 3.27 (Relevancia)**.** *Si hay una derivación $\Phi \rhd \Gamma \vdash t : \sigma$ entonces* $\mathsf{dom}(\Gamma) \subseteq \mathtt{fv}(t)$.

Para caracterizar la β-normalización a través de la tipabilidad, necesitamos restringir los tipos y contextos de tipado a aquellos que no tienen ocurrencias positivas de la constante $[\ ]$. Para ello damos la siguiente noción de ocurrencias *positivas* y *negativas* de un tipo.

Definición 3.28 (Ocurrencias positivas y negativas de tipos)**.** El conjunto de tipos que ocurren con signo $b \in \{+, -\}$ en un tipo σ (resp. en un multiconjunto de tipos $\mathcal{M}$, en un contexto Γ,

y en un par contexto/tipo (Γ, σ)) se escribe $\mathcal{O}^b(\sigma)$ (resp. $\mathcal{O}^b(\mathcal{M})$, $\mathcal{O}^b(\Gamma)$ y $\mathcal{O}^b(\Gamma \vdash \sigma)$). El conjunto $\mathcal{O}^+(X)$ es el conjunto de tipos que ocurren positivamente en X y $\mathcal{O}^-(X)$ es el conjunto de tipos que ocurren negativamente en X. Escribimos $\mathcal{O}^\pm(X)$ para denotar o bien $\mathcal{O}^+(X)$ o bien $\mathcal{O}^-(X)$ y $\mathcal{O}^\mp(...)$ para el conjunto opuesto en una regla dada. Todos estos conjuntos se definen mutuamente inductivamente por las siguientes condiciones, donde $\mathtt{T}$ denota o bien un tipo o bien un multiconjunto de tipos:

$$\frac{}{\sigma \in \mathcal{O}^+(\sigma)} \qquad \frac{}{\mathcal{M} \in \mathcal{O}^+(\mathcal{M})}$$

$$\frac{\mathtt{T} \in \mathcal{O}^\pm(\sigma_i) \quad I \neq \varnothing}{\mathtt{T} \in \mathcal{O}^\pm([\sigma_i]_{i \in I})} \qquad \frac{\mathtt{T} \in \mathcal{O}^\mp(\mathcal{M})}{\mathtt{T} \in \mathcal{O}^\pm(\mathcal{M} \to \tau)} \qquad \frac{\mathtt{T} \in \mathcal{O}^\pm(\tau)}{\mathtt{T} \in \mathcal{O}^\pm(\mathcal{M} \to \tau)}$$

$$\frac{y \in \mathsf{dom}(\Gamma) \quad \mathtt{T} \in \mathcal{O}^\mp(\Gamma(y))}{\mathtt{T} \in \mathcal{O}^\pm(\Gamma)} \qquad \frac{\mathtt{T} \in \mathcal{O}^\pm(\Gamma)}{\mathtt{T} \in \mathcal{O}^\pm(\Gamma \vdash \tau)} \qquad \frac{\mathtt{T} \in \mathcal{O}^\pm(\tau)}{\mathtt{T} \in \mathcal{O}^\pm(\Gamma \vdash \tau)}$$

Ejemplo 3.29 (Ocurrencias positivas y negativas)**.** *Valen los siguientes hechos:*

- $[\,] \in \mathcal{O}^+([\,])$
- $[\,] \in \mathcal{O}^-([\,] \to \sigma)$
- $[\,] \in \mathcal{O}^+(x : [[\,] \to \sigma])$
- $[\,] \in \mathcal{O}^+(x : [[\,] \to \sigma] \vdash \sigma)$

Se sabe que el sistema $\mathcal{HW}$ se puede utilizar para caracterizar los términos débilmente normalizantes en el cálculo-λ:

Teorema 3.30 (Caracterización de términos débilmente normalizantes en el cálculo-λ)**.** *Sea t un λ-término. Son equivalentes:*

1. *El término es débilmente normalizante, es decir $t \in \mathtt{WN}(\to_\beta)$.*
2. *El juicio $\Gamma \vdash t : \tau$ es derivable en $\mathcal{HW}$ y $[\,] \notin \mathcal{O}^+(\Gamma \vdash \tau)$*

Demostración. Ver [9]. Resulta de una adaptación de [26] al caso no idempotente. □

Extensión de la noción de tipado a contextos

Si $\mathtt{L}$ es un contexto, escribimos $\mathsf{dom}(\mathtt{L})$ para las variables ligadas por $\mathtt{L}$, y $\mathtt{fv}(\mathtt{L})$ para las variables libres de $\mathtt{L}$, tomando $\mathtt{fv}(\square) = \varnothing$. Usamos además la siguiente noción de *altura*:

Definición 3.31. La *altura* de un contexto de sustitución se define por:

$$\mathtt{height}(\square) \stackrel{\text{def}}{=} 1 \quad \mathtt{height}(\mathtt{L}[x \backslash t]) \stackrel{\text{def}}{=} \mathtt{height}(\mathtt{L}) + 1$$

Definición 3.32 (Extensión del sistema $\mathcal{HW}$ a contextos de sustitución)**.** El sistema de tipos $\mathcal{HW}$ se extiende con juicios de tipado de la forma $\Gamma \Vdash \mathtt{L} \rhd \Delta$, donde Γ y Δ son contextos de tipado y $\mathtt{L}$ es un contexto de sustitución, con las dos reglas siguientes:

$$\frac{}{\varnothing \Vdash \Box \rhd \varnothing} \qquad \frac{\Gamma \oplus x : [\sigma_i]_{i\in I} \Vdash \mathtt{L} \rhd \Delta \quad x \notin \mathsf{dom}\Delta \quad (\Sigma_k \vdash t : \sigma_k)_{k\in I\uplus J}}{\Gamma +_{k\in I\uplus J} \Sigma_k \Vdash \mathtt{L}[x\backslash t] \rhd \Delta \oplus x : [\sigma_j]_{j\in J}}$$

En la segunda regla, los conjuntos de índices I y J se suponen disjuntos.

Ejemplo 3.33. *Sea π la derivación de tipado para $[x\backslash yz]$:*

$$\frac{\frac{}{\varnothing \Vdash \Box \rhd \varnothing} \quad \frac{\vdots}{y : [[\gamma_1,\gamma_2] \to \alpha], z : [\gamma_1,\gamma_2] \vdash yz : \alpha}}{y : [[\gamma_1,\gamma_2] \to \alpha], z : [\gamma_1,\gamma_2] \Vdash [x\backslash yz] \rhd x : [\alpha]}$$

Entonces la siguiente es una derivación de tipado para $[x\backslash yz][y\backslash z]$:

$$\frac{\frac{\vdots}{\pi} \quad \frac{}{z : [[\gamma_1,\gamma_2] \to \alpha] \vdash z : [\gamma_1,\gamma_2] \to \alpha} \quad \frac{}{z : [\beta] \vdash z : \beta}}{z : [[\gamma_1,\gamma_2] \to \alpha, \beta, \gamma_1, \gamma_2] \Vdash [x\backslash yz][y\backslash z] \rhd x : [\alpha], y : [\beta]}$$

El siguiente lema enuncia algunas propiedades que se pueden demostrar fácilmente por inducción en $\mathtt{L}$.

Lema 3.34 (Propiedades de derivaciones de contextos de sustitución)**.**

1. *Si $\Gamma \Vdash \mathtt{L} \rhd \Delta$ entonces $\mathsf{dom}(\Gamma) \subseteq \mathtt{fv}(\mathtt{L})$ y $\mathsf{dom}(\Delta) \subseteq \mathsf{dom}(\mathtt{L})$.*

2. *Hay una derivación $\Phi_{t\mathtt{L}} \rhd \Lambda \vdash t\mathtt{L} : \sigma$ si y sólo si existen contextos Γ, Δ, Π tales que $\Lambda = \Gamma + \Pi$, y hay derivaciones $\Phi_{\mathtt{L}} \rhd \Gamma \Vdash \mathtt{L} \rhd \Delta$ y $\Phi_t \rhd \Delta; \Pi \vdash t : \sigma$. Además, $\mathtt{size}(\Phi_{\mathtt{L}[t]}) = \mathtt{size}(\Phi_{\mathtt{L}}) + \mathtt{size}(\Phi_t) - 1$.*

3. *Si $(\Phi^j_{\mathtt{L}} \rhd \Gamma_j \Vdash \mathtt{L} \rhd \Delta_j)_{j\in J}$, entonces $\Phi_{\mathtt{L}} \rhd +_{j\in J}\Gamma_j \Vdash \mathtt{L} \rhd +_{j\in J}\Delta_j$. Además, $\mathtt{size}(\Phi_{\mathtt{L}}) = +_{j\in J}\mathtt{size}(\Phi^j_{\mathtt{L}}) - (\mathtt{height}(\mathtt{L}) \cdot (|J| - 1))$.*

Tipabilidad implica normalización

Nuestro objetivo es ahora demostrar que los términos tipables en el sistema $\mathcal{HW}$ son débilmente normalizantes en la teoría de *sharing*. El resultado técnico clave es la propiedad de *reducción pesada del sujeto*. Recordar que en sistemas de tipos usuales la propiedad de *reducción del sujeto* establece que la reducción preserva el tipado. Más precisamente, si hay una derivación de tipos $\Phi \rhd \Gamma \vdash t : \tau$ y un paso de reducción $t \to t'$ entonces también hay una derivación de tipos $\Phi' \rhd \Gamma \vdash t' : \tau$. La propiedad de reducción *pesada* del sujeto establece que, bajo condiciones apropiadas sobre el paso $t \to t'$, también se puede asegurar que $\mathtt{size}(\Phi) > \mathtt{size}(\Phi')$.

En nuestro caso, podremos asegurar que el tamaño de la derivación decrece en tanto seleccionemos un paso $t \to t'$ que contrae un *redex tipado*. Intuitivamente, un redex es tipado si se encuentra dentro de un subtérmino que se debe utilizar en algún momento de la evaluación de t. Más rigurosamente, definimos la siguiente noción de *ocurrencias tipadas* de un término (T-ocurrencias). Para ello recordamos primero la definición de posición:

Definición 3.35 (Posiciones de un término)**.** El conjunto de *posiciones* de un término t, escrito $\mathtt{pos}(t)$, es el conjunto de palabras finitias sobre el alfabeto $\{0, 1\}$, definido inductivamente como sigue:

$$\frac{}{\epsilon \in \mathtt{pos}(t)} \quad \frac{p \in \mathtt{pos}(t)}{0p \in \mathtt{pos}(\lambda x.t)} \quad \frac{p \in \mathtt{pos}(t)}{0p \in \mathtt{pos}(ts)} \quad \frac{p \in \mathtt{pos}(t)}{1p \in \mathtt{pos}(st)} \quad \frac{p \in \mathtt{pos}(t)}{0p \in \mathtt{pos}(t[x\backslash s])} \quad \frac{p \in \mathtt{pos}(t)}{1p \in \mathtt{pos}(s[x\backslash t])}$$

El conjunto de posiciones de un contexto $\mathtt{C}$ se define de forma similar. El subtérmino de t (resp. $\mathtt{C}$) en la posición p se escribe $t|_p$ (resp. $\mathtt{C}|_p$) y se define de la manera esperada.

Definición 3.36 (T-occurrencia)**.** Supongamos dada una derivación $\Phi \rhd \Gamma \vdash t : \tau$. Una posición $p \in \mathtt{pos}(t)$ es una *T-occurrencia* de t en Φ si o bien $p = \epsilon$, o bien $p = ip'$ $(i = 0, 1)$ y $p' \in \mathtt{pos}(t|_i)$ es una T-occurrencia de $t|_i$ en *alguna* de las subderivaciones inmediatas de Φ. Una ocurrencia de un redex de t que es una T-occurrencia de t en Φ se dice una *T-occurrencia del redex* de t en Φ.

El siguiente lema estudia la relación entre derivaciones de tipo y la sustitución de una única ocurrencia de una variable, es decir, una derivación de tipos para $\mathtt{C}\langle\!\langle t \rangle\!\rangle$ se puede construir combinando una derivación de tipos para $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ y varias derivaciones de tipos para t.

Lema 3.37 (Sustitución parcial)**.** *Si* $\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle} \rhd x{:}[\sigma_i]_{i\in I}; \Gamma \vdash \mathtt{C}\langle\!\langle x \rangle\!\rangle : \tau$ *y* $(\Phi^i_u \rhd \Delta_i \vdash u : \sigma_i)_{i\in I}$ *entonces* $\Phi_{\mathtt{C}\langle\!\langle u \rangle\!\rangle} \rhd x{:}[\sigma_i]_{i\in I\backslash K}; \Gamma +_{k\in K} \Delta_k \vdash \mathtt{C}\langle\!\langle u \rangle\!\rangle : \tau$*, para algún* $K \subseteq I$ *donde* $\mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle u \rangle\!\rangle}) = \mathtt{size}(\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}) +_{k\in K} \mathtt{size}(\Phi^k_u) - |K|$*. Además, si* $p \in \mathtt{pos}(\mathtt{C})$ *es la ocurrencia del agujero en* $\mathtt{C}$ *y* p *es una T-occurrencia de* $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ *en* $\Phi_{\mathtt{C}\langle\!\langle x \rangle\!\rangle}$*, entonces* $K \neq \varnothing$*.*

Con esta herramienta estamos en condiciones de demostrar el siguiente resultado clave:

Lema 3.38 (Reducción pesada del sujeto para $\mathtt{sh}$)**.** *Sea* $\Phi \rhd \Gamma \vdash t : \tau$*. Si* $t \to_{\mathtt{sh}} t'$ *reduce una T-occurrencia de un* $\mathtt{sh}$*-redex de* t *en* Φ*, entonces existe una derivación* Φ' *tal que* $\Phi' \rhd \Gamma \vdash t' : \tau$ *y* $\mathtt{size}(\Phi) > \mathtt{size}(\Phi')$*.*

Ahora podemos relacionar las nociones de T-occurrencia y $\mathtt{sh}$-forma normal, antes de concluir con el resultado principal de esta subsección:

Lema 3.39. *Sea* $\Phi \rhd \Gamma \vdash t : \tau$ *tal que* $[\,] \notin \mathcal{O}^+(\Gamma \vdash \tau)$*. Entonces* $A(t, \Phi)$ *implica* $t \in \mathtt{NF}(\to_{\mathtt{sh}})$*.*

Teorema 3.40 (Tipabilidad implica $\mathtt{sh}$-normalización)**.** *Sea* $\Phi \rhd \Gamma \vdash t : \tau$ *tal que* $[\,] \notin \mathcal{O}^+(\Gamma \vdash \tau)$*. Entonces* t *es débilmente normalizante en la teoría de* sharing*.*

3.3.2. Completitud de la teoría de *sharing*

En esta sección demostramos el resultado de Fig. 3.1(b), es decir, la completitud de la teoría de *sharing* con respecto a la β-reducción en el cálculo-λ. Antes de proceder, es preciso enunciar algunas propiedades básicas del desenrollado de sustituciones.

Lema 3.41. *Sean $t, s \in \mathcal{T}$ términos, posiblemente con sustituciones explícitas. Entonces:*

1. *Si $t \to_{\mathtt{sh}} s$, entonces $t^\diamond \twoheadrightarrow_\beta s^\diamond$.*
2. *Si $t \in \mathtt{NF}(\to_{\mathtt{sh}})$, entonces $t^\diamond \in \mathtt{NF}(\to_\beta)$.*

Recordar que $\mathtt{NF}(\to)$ denota el conjunto de $\to$-formas normales.

En efecto, para ilustrar el primer punto tenemos $t = y[y\backslash(\lambda z.zz)(II)] \to_{\mathtt{db}} y[y\backslash(zz)[z\backslash II]] = u$ y $t^\diamond = (\lambda z.zz)(II) \to_\beta (II)\,(II) = u^\diamond$, y para ilustrar el segundo tenemos $t = x[y\backslash I[w'\backslash I]][z\backslash I] \in \mathtt{NF}(\to_{\mathtt{sh}})$ y $t^\diamond = x \in \mathtt{NF}(\to_\beta)$.

Concluimos con el resultado de completitud para el cálculo sh, *cf.* Fig. 3.1(b):

Proposición 3.42 (Completitud de la teoría de *sharing*)**.** *Si $t \twoheadrightarrow_\beta s \in \mathtt{NF}(\to_\beta)$ entonces existe un término u tal que $t \twoheadrightarrow_{\mathtt{sh}} u \in \mathtt{NF}(\mathtt{sh})$ y $u^\diamond = s$.*

3.3.3. Factorización de la teoría de *sharing*

En esta sección demostramos el resultado de Fig. 3.1(c), es decir, la factorización de la teoría de *sharing*. Para ello, demostramos que los pasos de reducción $\to_{\mathtt{sh}}$ que no son pasos de la estrategia $\overset{\vartheta}{\rightsquigarrow}$ siempre pueden postponerse después de los pasos de la estrategia $\overset{\vartheta}{\rightsquigarrow}$. Demostramos también que este proceso de postposición siempre termina y que, en última instancia, todos los pasos remanentes—no en la estrategia $\overset{\vartheta}{\rightsquigarrow}$—se pueden eliminar con la regla gc (y por lo tanto también son eliminados por la operación de desenrollado $\_^\diamond$). Más precisamente procedemos en tres etapas:

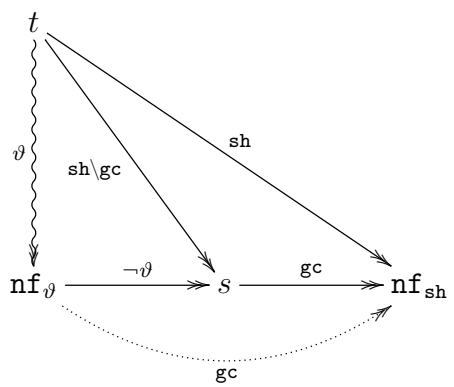


Figura 3.2: Descomposición de la Fig. 3.1(c)

1. Como paso preliminar, eliminamos los pasos gc: cualquier reducción $\to_{\mathtt{sh}}$ se puede factorizar como una reducción $\to_{\mathtt{sh}}$ sin pasos gc, que llamamos *estricta* y escribimos $\to_{\mathtt{sh}\backslash\mathtt{gc}}$, seguida por una secuencia de pasos gc (*cf.*).

2. A continuación demostramos un resultado de conmutación más complejo: las reducciones $\to_{\mathtt{sh}}$ sin pasos gc se pueden factorizar en dos partes (*cf.* Prop. 3.48):

   2.1 una secuencia de pasos $\to_{\mathtt{sh}}$ *externos*, que corresponden a la estrategia $\overset{\vartheta}{\rightsquigarrow}$

   2.2 una secuencia de pasos $\to_{\mathtt{sh}}$ *internos*, que no están en la estrategia $\overset{\vartheta}{\rightsquigarrow}$, escritos $\xrightarrow{\neg\vartheta}_{\mathtt{sh}}$.

   La demostración se basa en un resultado de factorización abstracta de Accattoli [1]. Escribimos $\xrightarrow{\neg\vartheta}_{\mathtt{sh}}$ en vez de $\to_{\mathtt{sh}}$ para los pasos internos. Dos ejemplos de pasos internos son $(xx)[x\backslash I] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (x\ I)[x\backslash I]$ y $(I\ x)[x\backslash I\ I] \xrightarrow{\neg\vartheta}_{\mathtt{sh}} (I\ x)[x\backslash z[z\backslash I]]$.

3. Por último, demostramos que los pasos internos que quedan después de haber alcanzado la $\overset{\vartheta}{\rightsquigarrow}$-forma normal sólo tienen lugar dentro de sustituciones basura, que se eliminan por la operación de desenrollado $\_^{\diamond}$ (*cf.* Lem. 3.50).

Postposición de gc

El siguiente resultado afirma que los pasos de *garbage collection* siempre pueden postponerse hacia el final de una secuencia de pasos de reducción.

Lema 3.43 (Postposición de gc)**.** *Si $t \twoheadrightarrow_{\mathtt{sh}} s$, entonces existe un término u tal que $t \twoheadrightarrow_{\mathtt{sh}\backslash\mathtt{gc}} u \twoheadrightarrow_{\mathtt{gc}} s$.*

Factorización de la reducción estricta

En esta subsección, mostramos que una secuencia de pasos de reducción estricta $\to_{\mathtt{sh}\backslash\mathtt{gc}}$ siempre puede factorizarse como una secuencia de pasos en la estrategia ($\overset{\vartheta}{\rightsquigarrow}$) seguida de pasos que no están en la estrategia ($\xrightarrow{\neg\vartheta}_{\mathtt{sh}}$). Más precisamente, decimos que t_1 reduce a t_2 a través de un paso ϑ*-interno*, y lo escribimos $t_1 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_2$, si y sólo si hay un paso estricto en la teoría de *sharing* que no es un paso en la estrategia call-by-need fuerte, es decir, $t_1\ (\to_{\mathtt{sh}\backslash\mathtt{gc}} \backslash \overset{\vartheta}{\rightsquigarrow})\ t_2$. A veces llamamos simplemente pasos *internos* a los pasos ϑ-internos, si el conjunto ϑ se deduce del contexto. Los pasos en la estrategia $\overset{\vartheta}{\rightsquigarrow}$ se llaman pasos ϑ*-externos* (o simplemente pasos *externos*).

La demostración del resultado de factorización es larga y técnica. Comenzamos recordando la definición de *sistema de factorización cuadrado* y un resultado de factorización abstracta, propuestos por Accattoli [1]:

Definición 3.44 (Sistema de factorización cuadrado)**.** Un *sistema de factorización cuadrado* está dado por un conjunto X y cuatro relaciones de reducción $(\rightsquigarrow_{\bullet}, \rightsquigarrow_{\circ}, \mapsto_{\bullet}, \mapsto_{\circ})$ tales que:

1. **Terminación**: $\rightsquigarrow_{\circ}$ y $\mapsto_{\circ}$ son fuertemente normalizantes.

2. **Intercambio de filas 1**: $(\rightsquigarrow_\bullet \rightsquigarrow_\circ) \subseteq (\rightsquigarrow_\circ^+ \rightsquigarrow_\bullet^*)$.

3. **Intercambio de filas 2**: $(\mapsto_\bullet \mapsto_\circ) \subseteq (\mapsto_\circ^+ \mapsto_\bullet^*)$.

4. **Intercambio diagonal 1**: $(\mapsto_\bullet \rightsquigarrow_\circ) \subseteq (\rightsquigarrow_\circ \mapsto^*)$.

5. **Intercambio diagonal 2**: $(\rightsquigarrow_\bullet \mapsto_\circ) \subseteq (\mapsto_\circ \rightsquigarrow^*)$.

con la siguiente notación:

$$\rightsquigarrow \stackrel{\text{def}}{=} (\rightsquigarrow_\bullet \cup \rightsquigarrow_\circ) \qquad \mapsto \stackrel{\text{def}}{=} (\mapsto_\bullet \cup \mapsto_\circ)$$
$$\to_\bullet \stackrel{\text{def}}{=} (\rightsquigarrow_\bullet \cup \mapsto_\bullet) \qquad \to_\circ \stackrel{\text{def}}{=} (\rightsquigarrow_\circ \cup \mapsto_\circ)$$

$$\to \stackrel{\text{def}}{=} (\to_\bullet \cup \to_\circ)$$

Teorema 3.45 (Factorización abstracta, Accattoli 2012)**.** *Sea* $(\rightsquigarrow_\bullet, \rightsquigarrow_\circ, \mapsto_\bullet, \mapsto_\circ)$ *un sistema de factorización cuadrado. Entonces* $\to^* \subseteq (\to_\circ^* \to_\bullet^*)$.

Demostración. Ver [1, Theorem 5.2]. □

Abajo enunciamos los dos lemas principales, *Estabilidad hacia atrás por pasos internos* y *Postposición de pasos internos*. El siguiente lema establece que varias nociones centrales en la definición de la estrategia call-by-need fuerte, tales como *respuestas*, *formas normales* y *contextos de evaluación*, se preservan por expansión a través de pasos internos:

Lema 3.46 (Estabilidad hacia atrás por pasos internos)**.** *Sea* $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t$ *un paso* ϑ*-interno. Entonces:*

1. *Si* t *es una respuesta (resp. un redex* $\mathtt{db}$*) entonces* t_0 *también es una respuesta (resp. un redex* $\mathtt{db}$*).*

2. *Si* t *es una* ϑ*-forma normal (resp.* ϑ*-estructura) entonces* t_0 *también es una* ϑ*-forma normal (resp.* ϑ*-estructura).*

3. *Si* $t = \mathtt{C}\langle\!\langle x \rangle\!\rangle$*, donde* $\mathtt{C}$ *es un* ϑ*-contexto de evaluación (resp.* ϑ*-contexto de evaluación inerte), entonces* t_0 *también es de la forma* $\mathtt{C}_0\langle\!\langle x \rangle\!\rangle$*, donde* $\mathtt{C}_0$ *es un* ϑ*-contexto de evaluación (resp.* ϑ*-contexto de evaluación inerte).*

El siguiente lema afirma que un paso externo se puede conmutar antes de un paso interno. En particular, un paso interno no puede crear un paso externo (ni creando un redex en una posición externa, ni convirtiendo una posición interna en externa).

Lema 3.47 (Postposición de pasos internos)**.** *Sea* $\mathtt{fv}(t_0) \subseteq \vartheta$*. Si* $t_0 \xrightarrow{\neg\vartheta}_{\mathtt{sh}} t_1 \overset{\vartheta}{\rightsquigarrow} t_3$*, entonces existe un término* t_2 *tal que* $t_0 \overset{\vartheta}{\rightsquigarrow}^* t_2 \twoheadrightarrow_{\mathtt{sh}}^{\neg\vartheta} t_3$*, donde la reducción de* t_0 *a* t_2 *tiene al menos un paso y la reducción de* t_2 *a* t_3 *tiene al menos dos pasos.*

El siguiente resultado es consecuencia de Teo. 3.45 y Lem. 3.47:

Proposición 3.48 (Factorización externa–interna)**.** *Sea* $\mathtt{fv}(t) \subseteq \vartheta$*. Si* $t \twoheadrightarrow_{\mathtt{sh}\backslash\mathtt{gc}} r$ *entonces hay un término* u *tal que* $t \overset{\vartheta}{\rightsquigarrow}^* u \twoheadrightarrow_{\mathtt{sh}}^{\neg\vartheta} r$.

Eliminación de pasos internos finales

Las dos subsecciones anteriores aseguran que cualquier reducción $\to_{\mathtt{sh}}$ se puede factorizar en un prefijo de pasos externos $\overset{\vartheta}{\rightsquigarrow}$ seguido por pasos internos o gc. En esta subsección refinamos el resultado, mostrando que si la reducción $\to_{\mathtt{sh}}$ alcanza una $\to_{\mathtt{sh}}$-forma normal, entonces todos los pasos internos que se postponen por la Prop. 3.48 se pueden eliminar con pasos gc.

Lema 3.49 (Inclusión de formas normales)**.** *Sean ϑ y t tales que $\mathtt{fv}(t) \subseteq \vartheta$. Si $t \in \mathtt{NF}(\to_{\mathtt{sh}})$ entonces $t \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$.*

Lema 3.50 (Formas normales módulo pasos internos y gc)**.** *Sean ϑ y t tales que $\mathtt{fv}(t) \subseteq \vartheta$.*

1. *Si $t \to_{\mathtt{gc}} \mathtt{nf}_\vartheta$ con $\mathtt{nf}_\vartheta \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ entonces $t \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$.*
2. *Si $t \xrightarrow{\neg\vartheta}_{\mathtt{sh}} \mathtt{nf}_\vartheta$ con $\mathtt{nf}_\vartheta \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ entonces $t \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ y existe un término u tal que $t \twoheadrightarrow_{\mathtt{gc}} u$ y $\mathtt{nf}_\vartheta \twoheadrightarrow_{\mathtt{gc}} u$.*

Los resultados de esta sección se pueden combinar para completar el argumento esbozado en la Fig. 3.2 para demostrar Fig. 3.1(c):

Proposición 3.51 (Factorización de la teoría de *sharing*)**.** *Sea $\vartheta = \mathtt{fv}(t)$. Si $t \twoheadrightarrow_{\mathtt{sh}} s \in \mathtt{NF}(\to_{\mathtt{sh}})$, entonces existe un término $u \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ tal que $t \overset{\vartheta}{\rightsquigarrow}^* u$ y $u^\diamond = s^\diamond$. (Más precisamente, $u \twoheadrightarrow_{\mathtt{gc}} s$).*

Por último, obtenemos el teorema de completitud de la Fig. 3.1(a):

Teorema 3.52 (Completitud de $\overset{\vartheta}{\rightsquigarrow}$ con respecto a la β-reducción)**.** *Sea $\vartheta = \mathtt{fv}(t)$. Si $t \twoheadrightarrow_\beta s \in \mathtt{NF}(\to_\beta)$ entonces existe un término $u \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow})$ tal que $t \overset{\vartheta}{\rightsquigarrow}^* u$ y $u^\diamond = s$.*

Capítulo 4

Call-by-need fuerte para patrones y recursión

4.1. Introducción

Este capítulo es fruto de la colaboración con Eduardo Bonelli y Kareem Mohamed, y se estructura de la siguiente manera. Destacamos en negrita las principales contribuciones:

- En la Sección 4.2, recordamos la definición del cálculo-λ extendido de Grégoire y Leroy (Def. 4.2), generalizamos la teoría de *sharing* al cálculo-λ extendido (Def. 4.6), y damos una caracterización sintáctica de las formas normales (Def. 4.6).

- En la Sección 4.3, proponemos un **sistema de tipos intersección no idempotente $\mathcal{HW}^{\mathsf{e}}$ para λ^{e}** (Def. 4.9), y probamos que **los términos débilmente normalizantes en λ^{e} son tipables** (Teo. 4.12) y que **los términos tipables son débilmente normalizantes en $\lambda^{\mathsf{e}}_{\mathtt{sh}}$** (Teo. 4.13). Más precisamente, ambos teoremas requieren no sólo que el término sea tipable, sino también que el correspondiente juicio de tipado sea "bueno" en un sentido preciso (*cf.* Def. 4.11). La noción de "bondad" generaliza la condición usual de que no haya ocurrencias positivas del multiconjunto vacío $[]$.

- En la Sección 4.4, proponemos una **estrategia call-by-need fuerte $\rightsquigarrow^{\mathsf{e}}$ para λ^{e}** (Def. 4.16), y demostramos que goza de buenas propiedades. Concretamente, es determinística (Prop. 4.20), extiende conservativamente a la estrategia call-by-need fuerte del capítulo anterior (Prop. 4.20), es correcta (Prop. 4.21) y **completa con respecto a la reducción en el cálculo-λ extendido** (Teo. 4.22).

4.2. Extensión de la teoría de *sharing*

En esta sección extendemos la teoría de *sharing* (*cf.* Def. 3.1) al cálculo-λ extendido. En la Sección 4.2.1, comenzamos repasando la definición del cálculo-λ extendido de Grégoire and Leroy [20]. En la Sección 4.2.2 damos la definición de la teoría de *sharing* extendida $\lambda^{\mathsf{e}}_{\mathtt{sh}}$.

4.2.1. El cálculo-λ extendido

Definición 4.1 (Sintaxis del cálculo-λ extendido, *cf.* [20])**.** Asumimos dado un conjunto infinito numerable de variables $x, y, z, \ldots$ y constantes $\mathbf{c}, \mathbf{c}', \mathbf{c}'', \ldots$. El conjunto de *términos* $\mathcal{T}^{\mathsf{e}}$ del cálculo-λ extendido se definen como sigue, mutuamente inductivamente con el conjunto de *ramas* (que corresponden a las ramas de una construcción `case`):

$$\begin{array}{lrcl} \text{Términos} & t, s, u, \ldots & ::= & x \mid \lambda x.t \mid t\,s \mid \mathbf{c} \mid \mathtt{fix}(x.t) \mid \mathtt{case}\ t\ \mathtt{of}\ \bar{b} \\ \text{Ramas} & b & ::= & \mathbf{c}\bar{x} \Rightarrow t \end{array}$$

Los contextos se definen de la manera esperada.

Definición 4.2 (El cálculo-λ extendido, *cf.* [20])**.** El *cálculo-λ^{e}* está dado por las siguientes reglas de reducción sobre $\mathcal{T}^{\mathsf{e}}$, cerradas por contextos arbitrarios. Escribimos $\to^{\mathsf{e}}$ para la relación de reducción resultante.

$$\begin{array}{rcll} (\lambda x.t)s & \mapsto_{\mathtt{db}} & t\{x := s\} & (\beta) \\ \mathtt{fix}(x.t) & \mapsto_{\mathtt{fix}} & t\{x := \mathtt{fix}(x.t)\} & (\mathtt{fix}) \\ \mathtt{case}\ \mathbf{c}_j\bar{t}\ \mathtt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} & \mapsto_{\mathtt{case}} & s_j\{\bar{x}_j := \bar{t}\} & (\mathtt{case}) \\ & & \text{si } j \in I \text{ y } |\bar{t}| = |\bar{x}_j| & \end{array}$$

La sustitución *simultánea*, sin captura, de una lista de variables $\bar{x}$ por una lista de términos $\bar{s}$ de la misma longitud en un término t se escribe $t\{\bar{x} := \bar{s}\}$. Un término t *encaja* con una rama $\mathbf{c}\bar{x} \Rightarrow s$ si $t = \mathbf{c}\bar{s}$ con $|\bar{s}| = |\bar{x}|$. Un término t encaja con una lista de ramas si encaja con al menos una rama. Dada la condición sintáctica de formación sobre las expresiones `case`, un término encaja con a lo sumo una rama. Observar que la reducción se puede bloquear si la guarda de un `case` no encaja con ninguna de las ramas (y nunca lo hace a pesar de que se reduzca la guarda). Las formas normales de λ^{e} se caracterizan como sigue:

Lema 4.3 (Formas normales)**.** *Las formas normales de λ^{e} están caracterizadas por la gramática:*

$$N \quad ::= \quad \lambda\bar{x}.x\bar{N} \mid \lambda\bar{x}.\mathbf{c}\bar{N} \mid \lambda\bar{x}.(\mathtt{case}\ N_0\ \mathtt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow N_i)_{i\in I})\bar{N}$$

donde N_0 no encaja con $(\mathbf{c}_i\bar{x}_i \Rightarrow N_i)_{i\in I}$. Observar que las listas $\bar{x}$ y $\bar{N}$ pueden estar vacías.

4.2.2. La teoría de *sharing* extendida

Definición 4.4 (Sintaxis de la teoría de *sharing* extendida)**.** Los *términos de la teoría de* sharing *extendida* $\mathcal{T}^{\mathsf{e}}_{\mathtt{sh}}$ se definen como sigue, extendiendo la sintaxis de λ^{e} con sustituciones explícitas:

$$t, s, u, \ldots \quad ::= \quad x \mid \lambda x.t \mid t\,s \mid \mathtt{fix}(x.t) \mid \mathbf{c} \mid \mathtt{case}\ t\ \mathtt{of}\ \bar{b} \mid t[x\backslash s]$$

Recordar que los términos sin sustituciones explícitas se denominan *términos puros*. Un término puro $t^{\diamond}$ se obtiene de un término $t \in \mathcal{T}^{\mathsf{e}}_{\mathtt{sh}}$ desenrollando las sustituciones explícitas, *ej.* $((\mathtt{case}\ z\ \mathtt{of}\ \mathbf{c} \Rightarrow z)[z\backslash \mathbf{d}\,\mathbf{d}])^{\diamond} = \mathtt{case}\ \mathbf{d}\,\mathbf{d}\ \mathtt{of}\ \mathbf{c} \Rightarrow \mathbf{d}\,\mathbf{d}$.

Para describir la reducción en la teoría de *sharing* extendida $\lambda^{\mathsf{e}}_{\mathtt{sh}}$, necesitamos introducir categorías sintácticas adicionales que generalizan las nociones de *respuesta* y *valor* en presencia de constructores:

$$
\begin{array}{lcl}
\text{Respuestas} & a & ::= & \mathtt{vL} \\
\text{Valores} & \mathtt{v} & ::= & \lambda x.t \mid \mathtt{A}\langle \mathbf{c} \rangle \\
\text{Contextos aplicativos} & \mathtt{A} & ::= & \Box \mid \mathtt{AL}\, t \\
\text{Contextos de sustitución} & \mathtt{L} & ::= & \Box \mid \mathtt{L}[x \backslash t]
\end{array}
$$

Una respuesta de la forma $(\lambda x.t)\mathtt{L}$ es una *respuesta abstracción* y una de la forma $\mathtt{A}\langle \mathbf{c} \rangle\mathtt{L}$ es una *respuesta aplicativa*. Un ejemplo de esta última es $((\mathbf{c}\, x)[x \backslash y]\, \mathbf{d})[y \backslash s]$.

Definición 4.5 (La teoría de *sharing* extendida)**.** La *teoría de* sharing *extendida* $\lambda^{\mathtt{e}}_{\mathtt{sh}}$ consta de las reglas de reducción sobre $\mathcal{T}^{\mathtt{e}}_{\mathtt{sh}}$ dadas abajo, cerradas por contextos arbitrarios. Escribimos $\rightarrow^{\mathtt{e}}_{\mathtt{sh}}$ para la relación de reducción.

$$
\begin{array}{rcll}
(\lambda x.t)\mathtt{L}\, s & \mapsto_{\mathtt{db}} & t[x \backslash s]\mathtt{L} & \\
\mathtt{C}\langle\!\langle x \rangle\!\rangle[x \backslash \mathtt{vL}] & \mapsto_{\mathtt{lsv}} & \mathtt{C}\langle \mathtt{v} \rangle[x \backslash \mathtt{v}]\mathtt{L} & \\
t[x \backslash s] & \mapsto_{\mathtt{gc}} & t & \text{if } x \notin \mathsf{fv}(t) \\
\mathtt{fix}(x.t) & \mapsto_{\mathtt{fix}} & t[x \backslash \mathtt{fix}(x.t)] & \\
\mathtt{case}\ \mathtt{A}\langle \mathbf{c}_j \rangle\mathtt{L}\ \mathtt{of}\ (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} & \mapsto_{\mathtt{case}} & s_j[\bar{x}_j \backslash \mathtt{A}]\mathtt{L} & \text{if } |\mathtt{A}\langle \Box \rangle| = |\bar{x}_j| \text{ and } j \in I
\end{array}
$$

La siguiente definición proporciona una caracterización inductiva de las $\rightarrow^{\mathtt{e}}_{\mathtt{sh}}$-formas normales.

Definición 4.6 (Formas normales de $\lambda^{\mathtt{e}}_{\mathtt{sh}}$)**.** Un término t *habilita* una lista de ramas $(\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I}$, escrito $t \succ (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I}$, si el término es de la forma $t = \mathtt{A}\langle \mathbf{c}_j \rangle\mathtt{L}$, para ciertos $\mathtt{A}, \mathtt{L}$, y $j \in I$ tales que $|\mathtt{A}| = |\bar{x}_j|$. El juicio judgment que define el conjunto de *formas normales* ($t \in \mathcal{N}$) se define simultáneamente con otros cuatro juicios, a saber *constantes normales* ($t \in \mathcal{K}$), *estructuras normales* ($t \in \mathcal{S}$), *formas normales de error* ($t \in \mathcal{E}$), y *abstracciones normales* ($t \in \mathcal{L}$).

$$
\frac{}{\mathbf{c} \in \mathcal{K}}\ \textsc{cNfCons} \qquad \frac{t \in \mathcal{K} \quad s \in \mathcal{N}}{t\, s \in \mathcal{K}}\ \textsc{cNfApp}
$$

$$
\frac{}{x \in \mathcal{S}}\ \textsc{sNfVar} \qquad \frac{t \in \mathcal{S} \quad s \in \mathcal{N}}{t\, s \in \mathcal{S}}\ \textsc{sNfApp}
$$

$$
\frac{t \in \mathcal{K} \cup \mathcal{L} \cup \mathcal{S} \quad t \not\succ (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \quad (s_i \in \mathcal{N})_{i \in I}}{\mathtt{case}\ t\ \mathtt{of}\ (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \in \mathcal{E}}\ \textsc{eNfStrt}
$$

$$
\frac{t \in \mathcal{E} \quad s \in \mathcal{N}}{t\, s \in \mathcal{E}}\ \textsc{eNfApp} \qquad \frac{t \in \mathcal{E} \quad (s_i \in \mathcal{N})_{i \in I}}{\mathtt{case}\ t\ \mathtt{of}\ (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \in \mathcal{E}}\ \textsc{eNfCase}
$$

$$
\frac{t \in \mathcal{N}}{\lambda x.t \in \mathcal{L}}\ \textsc{lNfLam} \qquad \frac{t \in \mathbb{X} \quad s \in \mathcal{S} \cup \mathcal{E} \quad x \in \mathsf{fv}(t)}{t[x \backslash s] \in \mathbb{X}}\ \textsc{nfSub}
$$

$$
\frac{t \in \mathcal{K}}{t \in \mathcal{N}}\ \textsc{nfCons} \quad \frac{t \in \mathcal{S}}{t \in \mathcal{N}}\ \textsc{nfStruct} \quad \frac{t \in \mathcal{E}}{t \in \mathcal{N}}\ \textsc{nfError} \quad \frac{t \in \mathcal{L}}{t \in \mathcal{N}}\ \textsc{nfLam}
$$

Lema 4.7 (Caracterización de formas normales en $\lambda^{\mathtt{e}}_{\mathtt{sh}}$)**.** *Son equivalentes:*

1. $t \in \mathcal{N}$

2. *t está en $\to^{\mathsf{e}}_{\mathtt{sh}}$-forma normal.*

4.3. Extensión del sistema de tipos

En esta sección presentamos el sistema $\mathcal{HW}^{\mathsf{e}}$, un sistema de tipos intersección no idempotente para la teoría de *sharing* extendida $\lambda^{\mathsf{e}}_{\mathtt{sh}}$, y demostramos que caracteriza normalización.

4.3.1. El sistema de tipos intersección no idempotente extendido

Asumimos que $\alpha, \beta, \gamma, \ldots$ varían sobre un conjunto de *variables de tipo*. El conjunto de *tipos* se denota con las letras $\tau, \sigma, \rho, \ldots$, y *los multiconjuntos finitos de tipos* con las letras $\mathcal{M}, \mathcal{N}, \mathcal{P}, \ldots$. El multiconjunto vacío se escribe $[]$, y $[\tau_1, \ldots, \tau_n]$ representa el multiconjunto que contiene a cada uno de los tipos τ_i con sus multiplicidades correspondientes. Además, $\mathcal{M} + \mathcal{N}$ denota la unión (aditiva) de multiconjuntos. Por ejemplo $[\mathbf{a}, \mathbf{b}] + [\mathbf{b}, \mathbf{c}] = [\mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{c}]$.

Definición 4.8 (Sintaxis de los tipos)**.** El conjunto de *tipos* de $\mathcal{HW}^{\mathsf{e}}$ está dado por la siguiente gramática, mutuamente recursivamente con los conjuntos de *tipos de datos*, *tipos de pre-error*, *tipos de error* y *tipos de rama*:

$$
\begin{array}{lrcl}
\text{Tipos} & \tau & ::= & \alpha \mid \mathcal{M} \to \tau \mid D \mid E \\
\text{Tipos de datos} & D & ::= & \mathbf{c} \mid D\,\mathcal{M} \\
\text{Tipos de pre-error} & G & ::= & \mathbb{E}\,\tau\,\bar{B} \mid G\,\tau \\
\text{Tipos de error} & E & ::= & \langle G \rangle \mid E\,\tau \\
\text{Tipos de rama} & B & ::= & \bar{\mathcal{M}} \Rightarrow \tau
\end{array}
$$

Un tipo τ *encaja* con una rama $\mathbf{c}\bar{x} \Rightarrow s$ si es de la forma $\tau = \mathbf{c}\bar{\mathcal{M}}$ con $|\bar{\mathcal{M}}| = |\bar{x}|$. Un tipo encaja con una lista de remas si encaja con al menos una rama.

Definición 4.9 (El sistema de tipos $\mathcal{HW}^{\mathsf{e}}$)**.** El sistema de tipos $\mathcal{HW}^{\mathsf{e}}$ está definido por medio de las siguientes reglas de tipado inductivas. Estas reglas definen la derivabilidad para cuatro formas de *juicios de tipado*, con las siguientes interpretaciones informales:

1. **Tipado** ($\Gamma; \Sigma \vdash t : \tau$) — El término t tiene tipo τ bajo el contexto Γ y el registro de errores Σ.

2. **Multi-tipado** ($\Gamma; \Sigma \vdash t : \mathcal{M}$) — El término t tiene los tipos de $\mathcal{M}$ bajo el contexto Γ y el registro de errores Σ.

3. **Aplicación** ($\tau \ @ \ \mathcal{M} \Rightarrow \sigma$) — Un término de tipo τ se puede aplicar a un argumento que tiene todos los tipos en $\mathcal{M}$, resultando en un término de tipo σ.

4. **Matching** ($\tau \ \langle \bar{b} \rangle \ \Gamma; \Sigma, \sigma$) — El tipo τ puede usarse como la guarda de un `case` con ramas $\bar{b}$, lo que puede tener éxito y resultar en un término de tipo σ, asumiendo ciertas hipótesis Γ y registros de errores Σ, o en caso contrario fallar.

Las reglas del sistema $\mathcal{HW}^{\mathsf{e}}$ son:

$$\frac{}{x : [\tau]; \Sigma \vdash x : \tau}\ \textsc{tVar} \qquad \frac{}{\varnothing; \Sigma \vdash \mathbf{c} : \mathbf{c}}\ \textsc{tCons}$$

$$\frac{\Gamma, x : \mathcal{M}; \Sigma \vdash t : \tau}{\Gamma; \Sigma \vdash \lambda x.t : \mathcal{M} \to \tau}\ \textsc{tAbs} \qquad \frac{\Gamma; \Sigma \vdash t : \tau \quad \tau \mathbin{@} \mathcal{M} \Rightarrow \sigma \quad \Delta; \Sigma \vdash s : \mathcal{M}}{\Gamma + \Delta; \Sigma \vdash ts : \sigma}\ \textsc{tApp}$$

$$\frac{\Gamma, x : \mathcal{M}; \Sigma \vdash t : \tau \quad \Delta; \Sigma \vdash \mathtt{fix}(x.t) : \mathcal{M}}{\Gamma + \Delta; \Sigma \vdash \mathtt{fix}(x.t) : \tau}\ \textsc{tFix} \qquad \frac{\Gamma; \Sigma \vdash t : \tau \quad \tau \langle \bar{b} \rangle \Delta; \Sigma, \sigma}{\Gamma + \Delta; \Sigma \vdash \mathtt{case}\ t\ \mathtt{of}\ \bar{b} : \sigma}\ \textsc{tCase}$$

$$\frac{\Gamma, x : \mathcal{M}; \Sigma \vdash t : \tau \quad \Delta; \Sigma \vdash s : \mathcal{M}}{\Gamma + \Delta; \Sigma \vdash t[x \backslash s] : \tau}\ \textsc{tES} \qquad \frac{(\Gamma_i; \Sigma \vdash t : \tau_i)_{1 \leqslant i \leqslant n} \quad (n \geqslant 0)}{\sum_{i=1}^{n} \Gamma_i; \Sigma \vdash t : \sum_{i=1}^{n} [\tau_i]}\ \textsc{tMulti}$$

$$\frac{}{\mathcal{M} \to \tau \mathbin{@} \mathcal{M} \Rightarrow \tau}\ \textsc{tAppFun} \qquad \frac{}{D \mathbin{@} \mathcal{M} \Rightarrow D\mathcal{M}}\ \textsc{tAppData}$$

$$\frac{(n \geqslant 1)}{\langle G \rangle\, \tau_1 \ldots \tau_n \mathbin{@} [\tau_1] \Rightarrow \langle G\, \tau_1 \rangle\, \tau_2 \ldots \tau_n}\ \textsc{tAppErr}$$

$$\frac{\mathbf{c}_j \bar{\mathcal{M}} \text{ matches } (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \quad \Gamma, \bar{x}_j : \bar{\mathcal{M}}; \Sigma \vdash s_j : \sigma_j}{\mathbf{c}_j \bar{\mathcal{M}}\, \langle (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \rangle\, \Gamma; \Sigma, \sigma_j}\ \textsc{tCMatch}$$

$$\frac{\tau \text{ does not match } (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \quad \left(\Gamma_i, \bar{x}_i : \bar{\mathcal{M}}_i; \Sigma \vdash s_i : \sigma_i\right)_{i \in I}}{\tau\, \langle (\mathbf{c}_i \bar{x}_i \Rightarrow s_i)_{i \in I} \rangle\, (\sum_{i \in I} \Gamma_i); \Sigma \cup \{\langle \mathbb{E}\, \tau\, (\bar{\mathcal{M}}_i \Rightarrow \sigma_i)_{i \in I} \rangle\, \bar{\rho}\}, \langle \mathbb{E}\, \tau\, (\bar{\mathcal{M}}_i \Rightarrow \sigma_i)_{i \in I} \rangle\, \bar{\rho}}\ \textsc{tCMismatch}$$

Escribimos $\pi, \xi, \ldots$ para las derivaciones de tipos y $\pi(\Gamma; \Sigma \vdash t : \tau)$ si π es una derivación de tipos del juicio $\Gamma; \Sigma \vdash t : \tau$. Las reglas son *lineales* con respecto al contexto de tipos en el sentido de que cada hipótesis se usa exactamente una vez. Las reglas son, en cambio, *cartesianas* con respecto al registro de errores, en el sentido de que cada hipótesis se puede usar cero, una o más veces.

4.3.2. Caracterización de términos débilmente normalizantes

En esta subsección, relacionamos tipabilidad en el sistema $\mathcal{HW}^{\mathsf{e}}$ con la propiedad de normalización débil en el cálculo-λ extendido, y con la propiedad de normalización débil en la teoría de *sharing* extendida. Comenzamos definiendo una noción apropiada de juicio "bueno" para $\mathcal{HW}^{\mathsf{e}}$ (Def. 4.11). En líneas generales, un juicio es bueno si no tiene ocurrencias positivas de $[]$ ni ocurrencias negativas de constructores. La razón para rechazar ocurrencias negativas de constructores se puede ilustrar en un término como $\mathtt{case}\ x\ \mathtt{of}\ (\mathbf{c} \Rightarrow \mathbf{d}) \cdot (\mathbf{e} \Rightarrow \Omega)$. Dicho término es tipable con tipo $\mathbf{d}$ si uno asume que $x : [\mathbf{c}]$. Sin embargo, no es débilmente normalizante en $\lambda^{\mathsf{e}}_{\mathtt{sh}}$. Observar que una variable libre de tipo $\mathbf{c}$ corresponde a una ocurrencia negativa del constructor $\mathbf{c}$ (en el juicio de tipado).

Sin embargo, prohibir las ocurrencias positivas de $[]$ y negativas de constructores no alcanza. La razón es la presencia de expresiones `case` bloqueadas. Considerar por ejemplo el

término $(\texttt{case}\ \mathbf{c}\ \texttt{of}\ (\mathbf{d} \Rightarrow \mathbf{d}))\,\Omega$. Este término es tipable; por ejemplo, se le puede asignar el tipo $\langle \mathbb{E}\,\mathbf{c}\,([\mathbf{d}] \Rightarrow \mathbf{d})[]\rangle$. Notar que el tipo del `case` bloqueado incluye los tipos de los argumentos a los que se aplica—en este caso el multiconjunto de tipos vacío. Además, este tipo queda registrado en el registro de errores. Esto nos permite extender la restricción de que $[]$ no ocurra positivamente y de que los constructores no ocurran negativamente a las expresiones `case` bloqueadas.

Como nota adicional, observar que un término como $\texttt{case}\ x\ \texttt{of}\ (\mathbf{c} \Rightarrow \mathbf{d}) \cdot (\mathbf{e} \Rightarrow \mathbf{d})$ está en forma normal, y por ende debería ser tipable. En efecto, se lo puede tipar asignándole a x un *tipo de error* apropiado.

Definición 4.10 (Ocurrencias positivas y negativas de tipos)**.** El conjunto de ocurrencias positivas (resp. negativas) de tipos en τ, se escribe $\mathcal{P}(\tau)$ (resp. $\mathcal{N}(\tau)$), y se define como sigue:

$$
\begin{array}{rcl@{\qquad}rcl}
\mathcal{P}(\alpha) & \stackrel{\text{def}}{=} & \{\alpha\} & \mathcal{N}(\alpha) & \stackrel{\text{def}}{=} & \varnothing \\
\mathcal{P}(\mathcal{M} \to \tau) & \stackrel{\text{def}}{=} & \mathcal{N}(\mathcal{M}) \cup \mathcal{P}(\tau) \cup \{\mathcal{M} \to \tau\} & \mathcal{N}(\mathcal{M} \to \tau) & \stackrel{\text{def}}{=} & \mathcal{P}(\mathcal{M}) \cup \mathcal{N}(\tau) \\
\mathcal{P}(\mathbf{c}) & \stackrel{\text{def}}{=} & \{\mathbf{c}\} & \mathcal{N}(\mathbf{c}) & \stackrel{\text{def}}{=} & \varnothing \\
\mathcal{P}(D\,\mathcal{M}) & \stackrel{\text{def}}{=} & \mathcal{P}(D) \cup \mathcal{P}(\mathcal{M}) \cup \{D\,\mathcal{M}\} & \mathcal{N}(D\,\mathcal{M}) & \stackrel{\text{def}}{=} & \mathcal{N}(D) \cup \mathcal{N}(\mathcal{M}) \\
\mathcal{P}(E\,\tau) & \stackrel{\text{def}}{=} & \mathcal{P}(E) \cup \mathcal{P}(\tau) \cup \{E\,\tau\} & \mathcal{N}(E\,\tau) & \stackrel{\text{def}}{=} & \mathcal{N}(E) \cup \mathcal{N}(\tau) \\
\mathcal{P}(\langle G \rangle) & \stackrel{\text{def}}{=} & \mathcal{P}(G) \cup \{G\} & \mathcal{N}(\langle G \rangle) & \stackrel{\text{def}}{=} & \mathcal{N}(G) \\
\mathcal{P}(G\,\tau) & \stackrel{\text{def}}{=} & \mathcal{P}(G) \cup \mathcal{P}(\tau) \cup \{G\,\tau\} & \mathcal{N}(G\,\tau) & \stackrel{\text{def}}{=} & \mathcal{N}(G) \cup \mathcal{N}(\tau) \\
\mathcal{P}(\mathbb{E}\,\tau\,\bar{B}) & \stackrel{\text{def}}{=} & \mathcal{P}(\tau) \cup \mathcal{P}(\bar{B}) \cup \{\mathbb{E}\,\tau\,\bar{B}\} & \mathcal{N}(\mathbb{E}\,\tau\,\bar{B}) & \stackrel{\text{def}}{=} & \mathcal{N}(\tau) \cup \mathcal{N}(\bar{B}) \\
\mathcal{P}(\mathcal{M}_1, \ldots, \mathcal{M}_n \Rightarrow \tau) & \stackrel{\text{def}}{=} & \bigcup_{i \in 1..n} \mathcal{N}(\mathcal{M}_i) \cup \mathcal{P}(\tau) \cup \{\bar{\mathcal{M}} \Rightarrow \tau\} & \mathcal{N}(\mathcal{M}_1, \ldots, \mathcal{M}_n \Rightarrow \tau) & \stackrel{\text{def}}{=} & \bigcup_{i \in 1..n} \mathcal{P}(\mathcal{M}_i) \cup \mathcal{N}(\tau) \\
\mathcal{P}(\mathcal{M}) & \stackrel{\text{def}}{=} & \bigcup_{\tau \in \mathcal{M}} \mathcal{P}(\tau) \cup \{\mathcal{M}\} & \mathcal{N}(\mathcal{M}) & \stackrel{\text{def}}{=} & \bigcup_{\tau \in \mathcal{M}} \mathcal{N}(\tau) \\
\mathcal{P}(\Gamma; \Sigma \vdash \tau) & \stackrel{\text{def}}{=} & \mathcal{N}(\Gamma) \cup \mathcal{P}(\Sigma) \cup \mathcal{P}(\tau) & \mathcal{N}(\Gamma; \Sigma \vdash \tau) & \stackrel{\text{def}}{=} & \mathcal{P}(\Gamma) \cup \mathcal{N}(\Sigma) \cup \mathcal{N}(\tau) \\
\mathcal{P}(\Gamma) & \stackrel{\text{def}}{=} & \bigcup_{(x \in \mathsf{dom}\Gamma)} \mathcal{P}(\Gamma(x)) & \mathcal{N}(\Gamma) & \stackrel{\text{def}}{=} & \bigcup_{(x \in \mathsf{dom}\Gamma)} \mathcal{N}(\Gamma(x))
\end{array}
$$

Sea además $\mathtt{X}$ un tipo (resp. tipo de datos, tipo de pre-error, tipo de error, tipo de rama, contexto de tipado). Decimos entonces que $\mathtt{X}$ está *cubierto* por un registro de errores Σ, y lo escribimos $\mathsf{covered}_\Sigma(\mathtt{X})$, si para cada tipo de error E tal que E es subfórmula de $\mathtt{X}$, es decir, ocurre en cualquier lugar del árbol sintáctico de $\mathtt{X}$, se tiene que $E \in \Sigma$.

Definición 4.11 (Tipos y juicios de tipado buenos)**.** Un tipo τ es *bueno* si $\mathbf{c} \notin \mathcal{P}(\tau)$ y $[] \notin \mathcal{N}(\tau)$. Decimos que $\mathcal{M}$ es bueno si cada $\tau \in \mathcal{M}$ es bueno. Un contexto de tipado Γ es *bueno* si se puede escribir como $\Gamma = \Gamma_g\Gamma_e$ de tal forma que $\Gamma_g(x)$ es bueno para todo $x \in \mathsf{dom}\Gamma_g$, y $\Gamma_e(x)$ es un tipo de error para todo $x \in \mathsf{dom}\Gamma_e$. Un juicio de tipado $\Gamma; \Sigma \vdash t : \tau$ es *bueno* si se cumplen todas las condiciones siguientes:

1. Γ es bueno;
2. $[] \notin \mathcal{P}(\Sigma)$ y $[] \notin \mathcal{P}(\tau)$;
3. $\mathbf{c} \notin \mathcal{N}(\Sigma)$ y $\mathbf{c} \notin \mathcal{N}(\tau)$ para todo constructor $\mathbf{c}$;
4. $\mathsf{covered}_\Sigma(\Gamma)$ y $\mathsf{covered}_\Sigma(\tau)$.

Abajo enunciamos los dos resultados principales de esta sección, que relacionan tipabilidad y normalización. Observar que:

- El Teo. 4.12 relaciona las formas normals en λ^{e} con la tipabilidad en $\mathcal{HW}^{\mathsf{e}}$, extendiendo el Teo. 3.30 del capítulo anterior (que relaciona las formas normales en el cálculo-λ con la tipabilidad en $\mathcal{HW}$).
- El Teo. 4.13 relaciona las formas normales en $\lambda^{\mathsf{e}}_{\mathtt{sh}}$ con la tipabilidad en $\mathcal{HW}^{\mathsf{e}}$, extendiendo el Teo. 3.40 del capítulo anterior (que relaciona las formas normales en la teoría de *sharing* con la tipabilidad en $\mathcal{HW}$).

Teorema 4.12 (Los términos débilmente normalizantes en λ^{e} son tipables)**.** *Sea t débilmente normalizante en λ^{e}. Entonces existen un contexto Γ, un registro de errores Σ y un tipo τ tales que $\Gamma; \Sigma \vdash t : \tau$ es derivable y bueno.*

Teorema 4.13 (Los términos tipables son débilmente normalizantes en $\lambda^{\mathsf{e}}_{\mathtt{sh}}$)**.** *Si $\Gamma; \Sigma \vdash t : \tau$ es derivable y bueno, entonces t es débilmente normalizante en $\lambda^{\mathsf{e}}_{\mathtt{sh}}$.*

4.4. Extensión de la estrategia call-by-need fuerte

En esta sección extendemos la estrategia call-by-need fuerte $\overset{\mathsf{S}}{\rightsquigarrow}$ para la teoría de *sharing*, presentada en el Capítulo 3, a la estrategia call-by-need fuerte $\rightsquigarrow^{\mathsf{e}}$ para la teoría de *sharing* extendida. Además, demostramos que la estrategia es completa con respecto al cálculo-λ extendido λ^{e}.

4.4.1. La estrategia call-by-need fuerte extendida

Al igual que en el capítulo anterior, la *estrategia* call-by-need fuerte $\overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}}$ es una relación binaria sobre el conjunto de términos extendidos $\mathcal{T}^{\mathsf{e}}_{\mathtt{sh}}$, y está parametrizada por un conjunto ϑ de variables congeladas. Las reglas de reducción son instancias de las reglas de reescritura de la teoría de *sharing* extendida (Def. 4.5), con dos diferencias: (1) no hay regla de *garbage collection*, (2) la reducción no está cerrada bajo contextos arbitrarios, sino bajo *contextos de evaluación*. Tal como en la Sección 3.2.2, para poder definir el conjunto de contextos de evaluación, comenzamos definiendo (sintácticamente) el conjunto de formas normales de la estrategia y a continuación describimos los contextos de evaluación.

Definición 4.14 (Formas normales de la estrategia call-by-need fuerte)**.** El conjunto de *variables no basura* de un término t se escribe $\mathsf{ngv}(t)$ y se define como $\mathsf{fv}(\downarrow_{\mathsf{gc}}(t))$, donde $\downarrow_{\mathsf{gc}}(t)$ es la gc-forma normal de t.

Para cada conjunto de variables ϑ, los conjuntos de *constantes normales* ($\mathcal{K}_\vartheta$), *estructuras normales* ($\mathcal{S}_\vartheta$), *formas normales de error* ($\mathcal{E}_\vartheta$), *abstracciones normales* ($\mathcal{L}_\vartheta$) y *formas normales* a secas ($\mathcal{N}_\vartheta$) se definen mutuamente inductivamente por medio de los siguientes juicios. En las reglas para sustituciones explícitas, escribimos $\mathbb{X}_\vartheta$ para representar cualquiera de los conjuntos $\mathcal{K}_\vartheta$, $\mathcal{S}_\vartheta$, $\mathcal{E}_\vartheta$ o $\mathcal{L}_\vartheta$:

$$\frac{}{\mathbf{c} \in \mathcal{K}_\vartheta}\ \textsc{cNfCons} \qquad \frac{t \in \mathcal{K}_\vartheta \quad s \in \mathcal{N}_\vartheta}{t\,s \in \mathcal{K}_\vartheta}\ \textsc{cNfApp} \qquad \frac{x \in \vartheta}{x \in \mathcal{S}_\vartheta}\ \textsc{sNfVar} \qquad \frac{t \in \mathcal{S}_\vartheta \quad s \in \mathcal{N}_\vartheta}{t\,s \in \mathcal{S}_\vartheta}\ \textsc{sNfApp}$$

$$\frac{t \in \mathcal{K}_\vartheta \cup \mathcal{L}_\vartheta \cup \mathcal{S}_\vartheta \quad t \not> (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} \quad (s_i \in \mathcal{N}_{\vartheta\cup\bar{x}_i})_{i\in I}}{\texttt{case}\ t\ \texttt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} \in \mathcal{E}_\vartheta}\ \textsc{eNfStrt}$$

$$\frac{t \in \mathcal{E}_\vartheta \quad s \in \mathcal{N}_\vartheta}{t\,s \in \mathcal{E}_\vartheta}\ \textsc{eNfApp} \qquad \frac{t \in \mathcal{E}_\vartheta \quad (s_i \in \mathcal{N}_{\vartheta\cup\bar{x}_i})_{i\in I}}{\texttt{case}\ t\ \texttt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} \in \mathcal{E}_\vartheta}\ \textsc{eNfCase} \qquad \frac{t \in \mathcal{N}_{\vartheta\cup\{x\}}}{\lambda x.t \in \mathcal{L}_\vartheta}\ \textsc{lNfLam}$$

$$\frac{t \in \mathbb{X}_{\vartheta\cup\{x\}} \quad s \in \mathcal{S}_\vartheta \cup \mathcal{E}_\vartheta \quad x \in \mathsf{ngv}(t)}{t[x\backslash s] \in \mathbb{X}_\vartheta}\ \textsc{nfSubNG} \qquad \frac{t \in \mathbb{X}_\vartheta \quad x \notin \mathsf{ngv}(t)}{t[x\backslash s] \in \mathbb{X}_\vartheta}\ \textsc{nfSubG}$$

$$\frac{t \in \mathcal{K}_\vartheta}{t \in \mathcal{N}_\vartheta}\ \textsc{nfCons} \qquad \frac{t \in \mathcal{S}_\vartheta}{t \in \mathcal{N}_\vartheta}\ \textsc{nfStruct} \qquad \frac{t \in \mathcal{L}_\vartheta}{t \in \mathcal{N}_\vartheta}\ \textsc{nfLam} \qquad \frac{t \in \mathcal{E}_\vartheta}{t \in \mathcal{N}_\vartheta}\ \textsc{nfError}$$

La definición sintáctica de formas normales dada arriba es similar a la caracterización sintáctica de $\rightarrow^{\mathsf{e}}_{\mathsf{sh}}$-formas normales dada en Def. 4.6, salvo que: (1) el conjunto de variables congeladas se registra explícitamente, (2) la regla NFSUB se refina en la regla NFSUBNG y (3) en una nueva regla NFSUBG que se agrega debido a la ausencia de gc en $\overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}}$.

Definición 4.15 (Contextos de evaluación extendidos)**.** Los juicios que definen los conjuntos de contextos de evaluación son de la forma $\mathtt{C} \in \mathsf{E}^h_\vartheta$ donde $\mathtt{C}$ es un contexto arbitrario, ϑ es un conjunto de variables y h es un símbolo llamado *discriminador* del contexto. Este símbolo puede tomar uno de los siguientes valores: '$\circ$', 'λ' o cualquier constructor $\mathbf{c}, \mathbf{d}, \ldots$ y su función es discriminar el constructor en la cabeza del contexto. Observar que las reglas de formación de contextos de evaluación imponen restricciones sobre los discriminadores. Un *contexto de evaluación* es un contexto $\mathtt{C}$ tal que el juicio $\mathtt{C} \in \mathsf{E}^h_\vartheta$ es derivable para algún conjunto de variables ϑ y algún discriminador h, usando las siguientes reglas de inferencia:

$$\frac{}{\square \in \mathsf{E}^\circ_\vartheta}\ \textsc{eBox}$$

$$\frac{\mathtt{C} \in \mathsf{E}^h_\vartheta \quad h \neq \lambda}{\mathtt{C}\,t \in \mathsf{E}^h_\vartheta}\ \textsc{eAppL} \qquad \frac{t \in \mathcal{S}_\vartheta \cup \mathcal{E}_\vartheta \quad \mathtt{C} \in \mathsf{E}^h_\vartheta}{t\,\mathtt{C} \in \mathsf{E}^\circ_\vartheta}\ \textsc{eAppRStruct} \qquad \frac{t \in \mathcal{K}_\vartheta \quad \mathtt{C} \in \mathsf{E}^h_\vartheta}{t\,\mathtt{C} \in \mathsf{E}^{\mathsf{hc}(t)}_\vartheta}\ \textsc{eAppRCons}$$

$$\frac{\mathtt{C} \in \mathsf{E}^h_\vartheta \quad t \notin \mathcal{S}_\vartheta \cup \mathcal{E}_\vartheta \quad x \notin \vartheta}{\mathtt{C}[x\backslash t] \in \mathsf{E}^h_\vartheta}\ \textsc{eSubsLNonStruct} \qquad \frac{\mathtt{C} \in \mathsf{E}^h_{\vartheta\cup\{x\}} \quad t \in \mathcal{S}_\vartheta \cup \mathcal{E}_\vartheta}{\mathtt{C}[x\backslash t] \in \mathsf{E}^h_\vartheta}\ \textsc{eSubsLStruct}$$

$$\frac{\mathtt{C}_1 \in \mathsf{E}^h_\vartheta \quad \mathtt{C}_2 \in \mathsf{E}^\circ_\vartheta}{\mathtt{C}_1\langle\!\langle x\rangle\!\rangle[x\backslash \mathtt{C}_2] \in \mathsf{E}^h_\vartheta}\ \textsc{eSubsR} \qquad \frac{\mathtt{C} \in \mathsf{E}^h_{\vartheta\cup\{x\}}}{\lambda x.\mathtt{C} \in \mathsf{E}^\lambda_\vartheta}\ \textsc{eLam}$$

$$\frac{\mathtt{C} \in \mathsf{E}^h_\vartheta \quad h \notin \{\mathbf{c}_i\}_{i\in I} \text{ ó } h = \mathbf{c}_j \in \{\mathbf{c}_i\}_{i\in I} \text{ y } |\mathtt{C}\langle y\rangle| \neq |\bar{x}_j|}{\texttt{case}\ \mathtt{C}\ \texttt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} \in \mathsf{E}^\circ_\vartheta}\ \textsc{eCase1}$$

$$\frac{t \in \mathcal{N}_\vartheta \quad t \not> (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I} \quad t_k \in \mathcal{N}_{\vartheta\cup\bar{x}_k} \text{ para todo } k < j \quad \mathtt{C} \in \mathsf{E}^h_{\vartheta\cup\bar{x}_i}}{\texttt{case}\ t\ \texttt{of}\ (\mathbf{c}_1\bar{x}_1 \Rightarrow t_1)\ldots(\mathbf{c}_j\bar{x}_j \Rightarrow \mathtt{C})\ldots(\mathbf{c}_n\bar{x}_n \Rightarrow t_n) \in \mathsf{E}^\circ_\vartheta}\ \textsc{eCase2}$$

La función $\mathsf{hc}(-)$ usada en la regla eAppRCons se define como sigue, por inducción en la derivación de $t \in \mathcal{K}_\vartheta$:

$$\mathsf{hc}(\mathbf{c}) \stackrel{\text{def}}{=} \mathbf{c} \qquad \mathsf{hc}(t\,s) \stackrel{\text{def}}{=} \mathsf{hc}(t) \qquad \mathsf{hc}(t[x\backslash s]) \stackrel{\text{def}}{=} \mathsf{hc}(t)$$

Notar en particular que $\mathsf{hc}(\mathtt{A}\langle\mathbf{c}\rangle\mathtt{L}) = \mathbf{c}$. La notación $|\mathtt{C}\langle y\rangle|$ usada en la regla eCase1 cuenta el número de argumentos en la columna vertebral del término $\mathtt{C}\langle y\rangle$, más precisamente:

$$\begin{array}{rcl} |x| & \stackrel{\text{def}}{=} & 0 \\ |\mathbf{c}| & \stackrel{\text{def}}{=} & 0 \\ |\lambda x.t| & \stackrel{\text{def}}{=} & 0 \\ |t\,s| & \stackrel{\text{def}}{=} & 1 + |t| \end{array} \qquad \begin{array}{rcl} |\mathtt{fix}(x.t)| & \stackrel{\text{def}}{=} & 0 \\ |t[x\backslash s]| & \stackrel{\text{def}}{=} & |t| \\ |\mathtt{case}\ t\ \mathtt{of}\ \bar{b}| & \stackrel{\text{def}}{=} & 0 \end{array}$$

Definición 4.16 (Estrategia call-by-need fuerte extendida)**.** La estrategia $\overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}}$ se define por las siguientes reglas:

$$\begin{array}{llrcll}
(\mathtt{db}) & & \mathtt{C}\langle(\lambda x.t)\mathtt{L}\,s\rangle & \overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}} & \mathtt{C}\langle t[x\backslash s]\mathtt{L}\rangle & \text{if } \mathtt{C} \in \mathsf{E}^h_\vartheta \\
(\mathtt{lsv}) & & \mathtt{C}_1\langle\mathtt{C}_2\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{vL}]\rangle & \overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}} & \mathtt{C}_1\langle\mathtt{C}_2\langle\mathtt{v}\rangle[x\backslash\mathtt{v}]\mathtt{L}\rangle & \text{if } \mathtt{C}_1\langle\mathtt{C}_2\langle\Box\rangle[x\backslash\mathtt{vL}]\rangle \in \mathsf{E}^h_\vartheta \\
(\mathtt{fix}) & & \mathtt{C}\langle\mathtt{fix}(x.t)\rangle & \overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}} & \mathtt{C}\langle t[x\backslash\mathtt{fix}(x.t)]\rangle & \text{if } \mathtt{C} \in \mathsf{E}^h_\vartheta \\
(\mathtt{case}) & & \mathtt{C}\langle\mathtt{case}\ \mathtt{A}\langle\mathbf{c}_j\rangle\mathtt{L}\ \mathtt{of}\ (\mathbf{c}_i\bar{x}_i \Rightarrow s_i)_{i\in I}\rangle & \overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}} & \mathtt{C}\langle s_j[\bar{x}_j\backslash\mathtt{A}]\mathtt{L}\rangle & \\
 & & & & & \text{if } \mathtt{C} \in \mathsf{E}^h_\vartheta \text{ con } j \in I \text{ and } |\mathtt{A}\langle\Box\rangle| = |\bar{x}_j|
\end{array}$$

Observar que el discriminador h en las condiciones de todas las reglas está cuantificado existencialmente.

Propiedades de la estrategia call-by-need fuerte extendida

La estrategia call-by-need fuerte extendida tiene las siguientes propiedades. Contrastarlas con las propiedades estudiadas en el capítulo anterior (Sección 3.2.3, Sección 3.3).

Lema 4.17 (Caracterización de formas normales)**.** *Sea t un término arbitrario. Son equivalentes:*

1. *t está en $\overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}}$-forma normal.*
2. *$t \in \mathcal{N}_\vartheta$.*

Proposición 4.18 (Reducción fuerte)**.** *Si t está en $\overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}}$-forma normal, entonces su desenrollado $t^\diamond$ está en $\rightarrow^{\mathsf{e}}$-forma normal.*

Proposición 4.19 (Determinismo)**.** *Si $t \overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}} s$ y $t \overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}} u$ entonces $s = u$.*

Proposición 4.20 (Conservatividad)**.** *La estrategia call-by-need fuerte extendida es conservativa con respecto a la estrategia call-by-need fuerte del Capítulo 3, es decir, si $t \overset{\vartheta}{\rightsquigarrow} s$ entonces $t \overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}} s$.*

Proposición 4.21 (Correctitud)**.** *Si $t \overset{\vartheta}{\rightsquigarrow} s$ entonces $t^\diamond \twoheadrightarrow^{\mathsf{e}} s^\diamond$.*

Teorema 4.22 (Completitud)**.** *Sea $\vartheta = \mathsf{fv}(t)$. Si $t \twoheadrightarrow^{\mathsf{e}} s \in \mathtt{NF}(\rightarrow^{\mathsf{e}})$, entonces existe un término $u \in \mathtt{NF}(\overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}})$ tal que $t\ (\overset{\vartheta}{\rightsquigarrow}{}^{\mathsf{e}})^*\ u$ y $u^\diamond = s$.*

Capítulo 5

Etiquetas de Lévy para el LSC

5.1. Introducción

Este capítulo es fruto de la colaboración con Eduardo Bonelli, y se estructura de la siguiente manera. Destacamos en negrita las principales contribuciones:

- En la Sección 5.2, motivamos algunas decisiones de diseño detrás de un cálculo con etiquetas de Lévy, y **definimos una variante del LSC con etiquetas de Lévy**, el LLSC (Def. 5.6).
- En la Sección 5.3, estudiamos las prpoiedades del LLSC. En particular:
  1. En la Sección 5.3.1 estudiamos sus propiedades sintácticas básicas.
  2. En la Sección 5.3.2 mostramos que **el LLSC es un sistema de reescritura axiomático ortogonal** (Prop. 5.32).
  3. En la Sección 5.3.3 demostramos que el LLSC es débilmente normalizante para reducción acotada (Prop. 5.43), es decir, cuando la reducción se restringe a etiquetas de altura acotada.
  4. En Sección 5.3.4 fortalecemos este resultado, y demostramos que **el LLSC es fuertemente normalizante para reducción acotada** (Teo. 5.49).
  5. En la Sección 5.3.5 damos dos demostraciones de que el LLSC es confluente, usando los resultados anteriores.

5.2. El LSC con etiquetas de Lévy

Nuestro objetivo es definir una variante del LSC con *etiquetas de Lévy*. Para ello comenzamos.

5.2.1. Teoría de residuos para el LSC

Recordemos que el LSC se define como la relación de reescritura $\to_{\mathsf{LSC}}$ que se obtiene como la unión de las tres siguientes reglas de reescritura, cerradas por contextos arbitrarios.

$$\begin{array}{lrcll}
\text{Beta a distancia} & (\lambda x.t)\mathtt{L}\, s & \mapsto_{\mathtt{db}} & t[x\backslash s]\mathtt{L} & \\
\text{Sustitución lineal} & \mathtt{C}\langle\!\langle x\rangle\!\rangle[x\backslash t] & \mapsto_{\mathtt{ls}} & \mathtt{C}\langle\!\langle t\rangle\!\rangle[x\backslash t] & \\
\textit{Garbage collection} & t[x\backslash s] & \mapsto_{\mathtt{gc}} & t & \text{if } x \notin \mathsf{fv}(t)
\end{array}$$

Como punto de partida en nuestra misión de definir una variante del LSC con etiquetas de Lévy, recordemos las propiedades fundamentales a las que aspiramos. Querríamos que el cálculo etiquetado le atribuya un *nombre* a cada redex, de tal manera que:

- Si un redex R' es un residuo de un redex R, entonces R y R' tienen el mismo nombre.
- Si un redex R crea un redex S, entonces el nombre de R es un "subnombre" del nombre de S.

Estas afirmaciones presuponen la existencia de una teoría de residuos *a priori*. Afortunadamente, en [2], Accattoli, Bonelli, Kesner y Lombardi dan una definición de residuos para el LSC, y demuestran que da lugar a una teoría de residuos con buenas propiedades. En esta subsección repasamos las definiciones y resultados que son relevantes para nuestro trabajo.

Definición 5.1 (El LSC marcado)**.** Suponemos dado un conjunto infinito numerable de *marcas* $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}, \ldots$. El conjunto de *términos marcados* está dado por la siguiente gramática:

$$\begin{array}{rcll}
t, s, u, \ldots & ::= & x & \text{variable} \\
& \mid & x^{\mathfrak{a}} & \text{variable marcada} \\
& \mid & \lambda x.t & \text{abstracción} \\
& \mid & \lambda x^{\mathfrak{a}}.t & \text{abstracción marcada} \\
& \mid & ts & \text{applicación} \\
& \mid & t[x\backslash s] & \text{sustitución} \\
& \mid & t[x^{\mathfrak{a}}\backslash s] & \text{sustitución marcada}
\end{array}$$

Las notaciones $\mathtt{L}$ para contextos de sustitución y $\mathtt{C}$ para contextos arbitrarios se extienden para permitir la presencia de marcas. Lo mismo para las nociones de variables libres y α-conversión. La *reducción marcada* $\xrightarrow{\mathfrak{a}}$ sobre términos marcados se define como la clausura por contextos arbitrarios de las siguientes reglas de reescritura:

$$\begin{array}{rcll}
(\lambda x^{\mathfrak{a}}.t)\mathtt{L}\, s & \mapsto^{\mathfrak{a}}_{\mathtt{db}} & t[x\backslash s]\mathtt{L} & \\
\mathtt{C}\langle\!\langle x^{\mathfrak{a}}\rangle\!\rangle[x\backslash t] & \mapsto^{\mathfrak{a}}_{\mathtt{ls}} & \mathtt{C}\langle t\rangle[x\backslash t] & \\
t[x^{\mathfrak{a}}\backslash s] & \mapsto^{\mathfrak{a}}_{\mathtt{gc}} & t & \text{if } x \notin \mathsf{fv}(t)
\end{array}$$

Un *redex marcado* es un redex R cuyo patrón es de alguna de las formas $(\lambda x^{\mathfrak{a}}.t)\mathtt{L}\, s$, $\mathtt{C}\langle\!\langle x^{\mathfrak{a}}\rangle\!\rangle[x\backslash t]$, ó $t[x^{\mathfrak{a}}\backslash s]$, y $\mathfrak{a}$ se llama la *marca del redex* R. La *reducción no marcada* $\to$ sobre términos marcados se define como la clausura por contextos arbitrarios de las reglas $\mathtt{db}$, $\mathtt{ls}$ y $\mathtt{gc}$ usuales—en este caso, el redex no está marcado pero se permiten marcas en otros lugares del término. El

ancla de un redex (marcado o no) es la variable que posiblemente lleva la marca. Si t es un término marcado, t° es el término que resulta de eliminar todas las marcas de t. Si $t^\circ = s$, decimos que t es una *variante* de s. En tal caso, identificamos los redexes de t y los redexes de s a través de la biyección obvia.

Definición 5.2 (Residuos en el LSC)**.** Sean R, S pasos coiniciales en el LSC:

$$R : t \to s \quad S : t \to u$$

Considerar una variante marcada t' de t con exactamente una marca $\mathfrak{a}$ sobre el ancla de S. Sea $R' : t' \to s'$ el paso correspondiente a R a través de la biyección obvia. El conjunto de *residuos de S después de R*, escrito S/R, es el conjunto de pasos de la forma $S' : s \to r$ para algún término r, tal que S' está marcado con $\mathfrak{a}$ en la variante marcada s' de s.

Por ejemplo, el siguiente es el grafo de reducción para una variante marcada del término $x[x\backslash y][y\backslash z]$ en el LSC marcado—escribimos $R(\mathfrak{a})$ para enfatizar que $\mathfrak{a}$ es la marca de R:

$$\begin{tikzcd}
x^{\mathfrak{a}}[x\backslash y^{\mathfrak{b}}][y\backslash z] \arrow[rr, "R(\mathfrak{a})"] \arrow[dd, "S(\mathfrak{b})"'] & & y^{\mathfrak{b}}[x\backslash y^{\mathfrak{b}}][y\backslash z] \arrow[dl, "S_1(\mathfrak{b})"'] \arrow[dr, "S_2(\mathfrak{b})"] & \\
& z[x\backslash y^{\mathfrak{b}}][y\backslash z] \arrow[dr, "S'_2(\mathfrak{b})"'] & & y^{\mathfrak{b}}[x\backslash z][y\backslash z] \arrow[dl, "S'_1(\mathfrak{b})"] \\
x^{\mathfrak{a}}[x\backslash z][y\backslash z] \arrow[rr, "R'(\mathfrak{a})"'] & & z[x\backslash z][y\backslash z] &
\end{tikzcd}$$

Además, de acuerdo con la definición de residuo, tenemos:

$$R/S = \{R'\} \quad S/R = \{S_1, S_2\} \quad S_1/S_2 = \{S'_1\} \quad S_2/S_1 = \{S'_1\} \quad R/R = \varnothing$$

Enunciamos dos resultados conocidos de [2]:

Proposición 5.3. *El LSC forma un sistema de reescritura axiomático ortogonal, en el sentido de Melliès [34].*

Proposición 5.4 (Creación de redexes en el LSC)**.** *Sean $t_1 \xrightarrow{R} t_2 \xrightarrow{S} t_3$ dos pasos consecutivos en el LSC tales que R crea S. Entonces S se crea en exactamente una de siete maneras posibles. Damos ejemplos paradigmáticos de cada una de las siete maneras, el enunciado completo se puede encontrar en [2]:*

1. `db` **crea** `db`**.** *Por ejemplo:* $(\lambda x.(\lambda y.t)s)u \to (\lambda y.t)[x\backslash s]\, u \to t[y\backslash u][x\backslash s]$.
2. `db` **crea** `ls`**.** *Por ejemplo:* $(\lambda x.xx)t \to (xx)[x\backslash t] \to (xt)[x\backslash t]$.
3. `db` **crea** `gc`**.** *Por ejemplo:* $(\lambda x.y)t \to y[x\backslash t] \to y$.
4. `ls` **crea** `db` **hacia arriba.** *Por ejemplo:* $x[x\backslash \lambda y.t]s \to (\lambda y.t)[x\backslash \lambda y.t]s \to t[y\backslash s][x\backslash \lambda y.t]$.
5. `ls` **crea** `db` **hacia abajo.** *Por ejemplo:* $(xt)[x\backslash \lambda y.s] \to ((\lambda y.s)t)[x\backslash \lambda y.s] \to s[y\backslash t][x\backslash \lambda y.s]$.
6. `ls` **crea** `gc`**.** *Por ejemplo:* $(yx)[x\backslash y] \to (yy)[x\backslash y] \to yy$.
7. `gc` **crea** `gc`**.** *Por ejemplo:* $y[x\backslash z][z\backslash t] \to y[z\backslash t] \to y$.

5.2.2. Definición del LSC etiquetado sin gc

Comenzamos dando una definición de una variante del LSC con etiquetas de Lévy sin la regla gc. Más adelante discutimos cómo extender esta definición para incorporar la regla gc.

Definición 5.5 (El LSC con etiquetas de Lévy sin gc, LLSC^I)**.** Suponemos dado un conjunto infinito numerable de *etiquetas iniciales* $\mathcal{I} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots\}$. Asumimos la existencia de una etiqueta inicial distinguida $\bullet \in \mathcal{I}$. El conjunto de *etiquetas* $\mathcal{L}^I$ está dado por la siguiente gramática:

$$\alpha, \beta, \gamma, \ldots ::= \mathbf{a} \mid \alpha\beta \mid \lceil\alpha\rceil \mid \lfloor\alpha\rfloor \mid \mathtt{db}(\alpha)$$

Las etiquetas se consideran módulo asociatividad de la operación de yuxtaposición, es decir, dadas etiquetas $\alpha, \beta, \gamma \in \mathcal{L}$ cualesquiera, declaramos que se verifica $(\alpha\beta)\gamma = \alpha(\beta\gamma)$. Las etiquetas que no son de la forma $\alpha\beta$ se llaman *atómicas*. El conjunto de *términos etiquetados* $\mathcal{T}^{\ell I}$ está definido por la siguiente gramática:

$$t, s, \ldots \quad ::= \quad x^\alpha \quad | \quad \lambda^\alpha x.t \quad | \quad @^\alpha(t, s) \quad | \quad t[x\backslash s]$$

La *etiqueta atómica exterior* de una etiqueta α se escribe $\uparrow(\alpha)$ y se define como sigue, por inducción en el número de yuxtaposiciones que se usan para construir la etiqueta α:

$$\uparrow(\alpha) \stackrel{\text{def}}{=} \begin{cases} \uparrow(\alpha_1) & \text{si } \alpha = \alpha_1\alpha_2 \\ \alpha & \text{si } \alpha \text{ es atómica} \end{cases}$$

Por ejemplo $\uparrow(\lceil\mathbf{ab}\rceil\mathbf{ac}) = \lceil\mathbf{ab}\rceil$. Análogamente, la *etiqueta atómica interior* de una etiqueta α, se escribe $\downarrow(\alpha)$ y se define como sigue:

$$\downarrow(\alpha) \stackrel{\text{def}}{=} \begin{cases} \downarrow(\alpha_2) & \text{si } \alpha = \alpha_1\alpha_2 \\ \alpha & \text{si } \alpha \text{ es atómica} \end{cases}$$

La *etiqueta externa* de un término t se escribe $\ell(t)$ y se define como la etiqueta que decora su nodo más externo, salteando sustituciones:

$$\ell(x^\alpha) = \alpha \quad \ell(\lambda^\alpha x.t) = \alpha \quad \ell(@^\alpha(t, s)) = \alpha \quad \ell(t[x\backslash s]) = \ell(t)$$

La *etiqueta atómica exterior* de un término t se escribe $\uparrow(t)$ y se define como $\uparrow(\ell(t))$. Por ejemplo:

$$\ell((\lambda^{\mathbf{abc}}x.x^{\mathbf{d}})[y\backslash y^{\mathbf{d}}]) = \mathbf{abc} \quad \text{y} \quad \uparrow((\lambda^{\mathbf{abc}}x.x^{\mathbf{d}})[y\backslash y^{\mathbf{d}}]) = \mathbf{a}$$

La sintaxis de los contextos se extiende para permitir términos con etiquetas:

$$\mathtt{C} \quad ::= \quad \square \quad | \quad \lambda^\alpha x.\mathtt{C} \quad | \quad @^\alpha(\mathtt{C}, t) \quad | \quad @^\alpha(t, \mathtt{C}) \quad | \quad \mathtt{C}[x\backslash t] \quad | \quad t[x\backslash \mathtt{C}]$$

y lo mismo para contextos de sustitución. Se define una operación para *agregar una etiqueta a un término*, que se escribe $\alpha : t$ y se define por inducción en t, salteando sustituciones:

$$\begin{array}{rcl@{\qquad}rcl} \alpha : x^\beta & \stackrel{\text{def}}{=} & x^{\alpha\beta} & \alpha : \lambda^\beta x.t & \stackrel{\text{def}}{=} & \lambda^{\alpha\beta}x.t \\ \alpha : @^\beta(t, s) & \stackrel{\text{def}}{=} & @^{\alpha\beta}(t, s) & \alpha : (t[x\backslash s]) & \stackrel{\text{def}}{=} & (\alpha : t)[x\backslash s] \end{array}$$

El *LSC con etiquetas de Lévy sin* gc, LLSC^I, es el sistema de reescritura cuyos términos son los términos etiquetados $\mathcal{T}^{\ell I}$ y con la relación de reescritura $\to_{\ell I}$ dada por la unión de las siguientes reglas, cerradas por contextos arbitrarios:

$$\begin{array}{rcl} @^\alpha((\lambda^\beta x.t)\mathtt{L}, s) & \mapsto_{\mathtt{db}} & \alpha\lceil \mathtt{db}(\beta)\rceil : t[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : s]\mathtt{L} \\ \mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle[x\backslash t] & \mapsto_{\mathtt{ls}} & \mathtt{C}\langle\!\langle \alpha \bullet : t \rangle\!\rangle[x\backslash t] \end{array}$$

$$\to_{\ell I\,\mathtt{db}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathtt{db}} \rangle \quad \to_{\ell I\,\mathtt{ls}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathtt{ls}} \rangle \quad \to_{\ell I} \stackrel{\text{def}}{=} \to_{\ell I\,\mathtt{db}} \cup \to_{\ell I\,\mathtt{ls}}$$

Con respecto a los *nombres* de los pasos, el nombre de un paso db como el de la regla $\mapsto_{\mathtt{db}}$ es $\mathtt{db}(\beta)$, mientras que el nombre de un paso ls como el de la regla $\mapsto_{\mathtt{ls}}$ es $\downarrow(\alpha) \bullet \uparrow(t)$. A veces escribimos $t \xrightarrow{\alpha}_{\ell I} s$ cuando $t \to_{\ell I} s$ y el nombre del redex que se contrae es α.

5.2.3. Definición del LSC etiquetado – extensión con gc

Definición 5.6 (El LSC etiquetado con gc, LLSC)**.** Tal como en Def. 5.5, suponemos dado un conjunto de *etiquetas iniciales* $\mathcal{I} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots\}$, y asumimos la existencia de dos etiquetas distinguidas $\bullet \in \mathcal{I}$ y $\otimes \in \mathcal{I}$. El conjunto de etiquetas $\mathcal{L}$ se define por la siguiente gramática:

$$\alpha, \beta, \gamma, \ldots ::= \mathbf{a} \mid \alpha\beta \mid \lceil\alpha\rceil \mid \lfloor\alpha\rfloor \mid \mathtt{db}(\alpha)$$

Las metavariables $\Omega, \Theta, \Psi, \ldots$ varían sobre *conjuntos finitos* de etiquetas iniciales. El conjunto de *términos etiquetados* $\mathcal{T}^\ell$ se define mediante la siguiente gramática:

$$t, s, \ldots \quad ::= \quad x^\alpha \quad | \quad \lambda^\alpha_\Omega x.t \quad | \quad @^\alpha(t, s) \quad | \quad t[x\backslash s]_\Omega$$

Las nociones de *etiqueta atómica exterior* $\uparrow(\alpha)$ de una etiqueta α, *etiqueta atómica interior* $\downarrow(\alpha)$ de una etiqueta α, *etiqueta externa* $\ell(t)$ de un término etiquetado t, la operación para *agregar una etiqueta a un término* $\alpha : t$, y las nociones de *contexto* y *contexto de sustitución* se definen de manera análoga a la de Def. 5.5.

El *LSC con etiquetas de Lévy con* gc, LLSC, es el sistema de reescritura cuyos objetos son los términos $\mathcal{T}^\ell$ y con la relación de reescritura $\to_\ell$ dada por la unión de las siguientes reglas, cerradas por contextos arbitrarios.

$$\begin{array}{rcll} @^\alpha((\lambda^\beta_\Omega x.t)\mathtt{L}, s) & \mapsto_{\mathtt{db}} & \alpha\lceil \mathtt{db}(\beta)\rceil : t[x\backslash\lfloor \mathtt{db}(\beta)\rfloor : s]_\Omega\mathtt{L} & \\ \mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle[x\backslash t]_\Omega & \mapsto_{\mathtt{ls}} & \mathtt{C}\langle\!\langle \alpha \bullet : t \rangle\!\rangle[x\backslash t]_\Omega & \\ t[x\backslash s]_\Omega & \mapsto_{\mathtt{gc}} & t & \text{si } x \notin \mathsf{fv}(t) \end{array}$$

$$\to_{\ell\,\mathtt{db}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathtt{db}} \rangle \quad \to_{\ell\,\mathtt{ls}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathtt{ls}} \rangle \quad \to_{\ell\,\mathtt{gc}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathtt{gc}} \rangle \quad \to_{\ell} \stackrel{\text{def}}{=} \to_{\ell\,\mathtt{db}} \cup \to_{\ell\,\mathtt{ls}} \cup \to_{\ell\,\mathtt{gc}}$$

Con respecto a los *nombres* de los pasos, el nombre de un paso db como el de la regla $\mapsto_{\mathtt{db}}$ es $\mathtt{db}(\beta)$, el nombre de un paso ls como el de la regla $\mapsto_{\mathtt{ls}}$ es $\downarrow(\alpha) \bullet \uparrow(t)$, y el nombre de un paso gc como el de la regla $\mapsto_{\mathtt{gc}}$ es el *conjunto de etiquetas* $\{\mathbf{a} \bullet \uparrow(s) \mid \mathbf{a} \in \Omega\}$. Tal como antes, escribimos $t \xrightarrow{\alpha}_\ell s$ cuando $t \to_\ell s$ y el nombre del redex que se contrae es α.

Observar que los *nombres de redexes* $\mu, \nu, \xi, \ldots$ tienen tres formas posibles, dadas por la gramática de abajo, donde α representa una etiqueta arbitraria en $\mathcal{L}$, y ω, ω' representan etiquetas atómicas:

$$\mu ::= \quad \underbrace{\mathtt{db}(\alpha)}_{\text{nombre de un paso } \mathtt{db}} \quad | \quad \underbrace{\omega \bullet \omega'}_{\text{nombre de un paso } \mathtt{ls}} \quad | \underbrace{\{\omega_1 \bullet \omega'_1, \ldots, \omega_n \bullet \omega'_n\}}_{\text{nombre de un paso } \mathtt{gc},\ n \geqslant 1}$$

En general consideramos a los nombres de redexes como elementos de su propio género (*sort*), pero de manera ocasional identificamos los nombres de pasos `db` y `ls` con la etiqueta subyacente correspondiente—por ejemplo el nombre de redex $\mathtt{db}(\mathbf{a})$ se puede entender como la etiqueta $\mathtt{db}(\mathbf{a})$.

El LSC con gc no es estable

La estabilidad es una propiedad abstracta de los sistemas de reescritura con residuos, que afirma que los pasos de cómputo se crean de manera esencialmente única: si dos pasos tienen un residuo común, deben tener también un ancestro común. Esto significa que la presencia de un paso de cómputo se puede reducir a una única *causa*. La propiedad de estabilidad fue originalmente estudiada por Jean-Jacques Lévy [30, 32], e inpirada por la noción de estabilidad de Gérard Berry en el ámbito de la semántica denotacional [8].

Definición 5.7 (Estabilidad)**.** Un sistema de reescritura axiomático ortogonal verifica la propiedad de *estabilidad* si dados pasos R, S, T_1, T_2, T_3 tales que $T_3 \in T_1/(S/R)$ y $T_3 \in T_2/(R/S)$, entonces existe un paso T_0 tal que $T_1 \in T_0/R$ y $T_2 \in T_0/S$. Gráficamente:

$$\begin{tikzcd}
 & T_0 \arrow[dl, "R"'] \arrow[dr, "S"] & \\
T_1 \arrow[dr, "S/R"'] & & T_2 \arrow[dl, "R/S"] \\
 & T_3 &
\end{tikzcd}$$

No es difícil ver que la propiedad de estabilidad falla en el LSC ante la presencia de la regla gc.

Observación 5.8 (El LSC no es estable). Considerar el diagrama:

$$\begin{tikzcd}
 & x[y\backslash z][z\backslash w] \arrow[dl, "R"'] \arrow[dr, "S"] & \\
x[y\backslash w][z\backslash w] \arrow[d, "T"'] \arrow[dr, "S/R"] & & x[z\backslash w] \arrow[d, "T'"] \arrow[dl, equal, "\varnothing"] \\
x[y\backslash w] & x[z\backslash w] \arrow[d, "T'"'] & x \\
 & x &
\end{tikzcd}$$

Observar que T y T' no tienen un ancestro común, por cuanto el LSC con gc no verifica la propiedad de estabilidad.

5.3. Propiedades del LSC con etiquetas de Lévy

En esta sección se establecen varias propiedades del LLSC:

1. En la Sección 5.3.1 demostramos propiedades básicas de la reducción etiquetada, incluyendo el invariante de términos correctamente etiquetados.
2. En la Sección 5.3.2 estudiamos diagramas de permutación en el LLSC. En particular demostramos que el LLSC es un sistema de reescritura axiomático orthogonal (Prop. 5.32).
3. En la Sección 5.3.3 demostramos que el LLSC es *débilmente* normalizante si la altura de los nombres de los redexes que se contraen está acotada (Prop. 5.43).
4. En la Sección 5.3.4 fortalecemos el resultado anterior, mostrando que el LLSC es *fuertemente* normalizante si la altura de los nombres de los redexes que se contraen está acotada (Teo. 5.49).
5. En la Sección 5.3.5 obtenemos como corolario que el LLSC es confluente (Teo. 5.51).

5.3.1. Propiedades básicas

Comenzamos demostrando algunas propiedades básicas del cálculo etiquetado LLSC definido en Def. 5.6, incluyendo el invariante de *términos correctamente etiquetados.*

Etiquetas y contextos

Lema 5.9 (Propiedades de las etiquetas y contextos)**.**

1. $\alpha : (\beta : t) = (\alpha\beta) : t$
2. *Si* $\mathtt{L}$ *es un contexto de sustitución, entonces* $\alpha : (t\mathtt{L}) = (\alpha : t)\mathtt{L}$.
3. *Si* $\mathtt{C}$ *no es un contexto de sustitución, entonces* $\alpha : \mathtt{C}\langle t\rangle = (\alpha : \mathtt{C})\langle t\rangle$.
4. $\uparrow (\alpha : t) = \uparrow (\alpha)$
5. $\uparrow (\mathtt{C}\langle\!\langle x^{\alpha}\rangle\!\rangle) = \uparrow (\mathtt{C}\langle \alpha \bullet \ : t\rangle)$
6. $\alpha : \mathtt{C}\langle\!\langle x^{\beta}\rangle\!\rangle$ *es de la forma* $\mathtt{C}'\langle\!\langle x^{\beta'}\rangle\!\rangle$*, donde* $\downarrow (\beta) = \downarrow (\beta')$ *y* $\alpha : \mathtt{C}\langle \beta \bullet \ : t\rangle = \mathtt{C}'\langle \beta' \bullet \ : t\rangle$.

Lema 5.10 (El agregado de etiquetas es funtorial)**.** *Si* $t \xrightarrow{\mu}_{\ell} s$ *entonces* $(\alpha : t) \xrightarrow{\mu}_{\ell} (\alpha : s)$.

Lema 5.11 (La reducción preserva la etiqueta exterior)**.** *Si* $t \to_{\ell} s$ *entonces* $\uparrow (t) = \uparrow (s)$.

Términos inicialmente y correctamente etiquetados

En esta subsección damos una definición precisa de *términos inicialmente etiquetados*, y definimos un invariante para *términos correctamente etiquetados*, de tal modo que todos los términos inicialmente etiquetados están correctamente etiquetados y la relación de reescritura ($\to_{\mathsf{LSC}}$) preserva el etiquetado correcto.

Definición 5.12 (Etiquetas hoja)**.** Sea $t \in \mathcal{T}^\ell$. El multiconjunto de *etiquetas hoja* de t se escribe $\mathsf{vl}_x(t)$ y se define como el multiconjunto de etiquetas atómicas de la forma $\downarrow(\alpha)$ para cada ocurrencia de x^α en t. Formalmente:

$$\begin{array}{rcll}
\mathsf{vl}_x(x^\alpha) & \stackrel{\text{def}}{=} & \{\downarrow(\alpha)\} & \\
\mathsf{vl}_x(y^\alpha) & \stackrel{\text{def}}{=} & \varnothing & \text{si } x \neq y \\
\mathsf{vl}_x(\lambda^\alpha_\Omega y.t) & \stackrel{\text{def}}{=} & \mathsf{vl}_x(t) & \text{si } x \neq y \\
\mathsf{vl}_x(@^\alpha(t,s)) & \stackrel{\text{def}}{=} & \mathsf{vl}_x(t) \uplus \mathsf{vl}_x(s) & \\
\mathsf{vl}_x(t[y\backslash s]_\Omega) & \stackrel{\text{def}}{=} & \mathsf{vl}_x(t) \uplus \mathsf{vl}_x(s) & \text{si } x \neq y
\end{array}$$

Extendemos esta operación a contextos definiendo $\mathsf{vl}_x(\Box) \stackrel{\text{def}}{=} \varnothing$. Observar que $\mathsf{vl}_x(\mathtt{C}\langle t\rangle) = \mathsf{vl}_x(\mathtt{C}) \uplus \mathsf{vl}_x(t)$ si $\mathtt{C}$ no liga x. De manera ocasional, tratamos a estos multiconjuntos como conjuntos, cuando la multiplicidad es irrelevante.

Lema 5.13 (Propiedades de las etiquetas hoja)**.**

1. $\mathsf{vl}_x(\alpha : t) = \mathsf{vl}_x(t)$
2. *Si* $t \to_\ell s$ *entonces* $\mathsf{vl}_x(t) \supseteq \mathsf{vl}_x(s)$ *para toda variable* x *(donde "$\supseteq$" denota la inclusión de* **conjuntos***).*

Definición 5.14 (Términos inicialmente etiquetados)**.** Un término $t \in \mathcal{T}^\ell$ está *inicialmente etiquetado*, lo que se escribe $\textsc{init}(t)$, si:

1. Para cada subtérmino s, la etiqueta externa $\ell(s)$ es inicial y $\ell(s) \notin \{\bullet, \otimes\}$.
2. Para cada par de subtérminos s_1, s_2 en posiciones diferentes, se tiene que $\ell(s_1) \neq \ell(s_2)$.
3. Para cada subtérmino de t que es un ligador, es decir, de la forma $(\lambda^{\mathbf{a}}_\Omega x.s)$, o de la forma $s[x\backslash u]_\Omega$, se tiene que $\Omega = \begin{cases} \{\otimes\} & \text{si } \mathsf{vl}_x(s) = \varnothing \\ \mathsf{vl}_x(s) & \text{en caso contrario} \end{cases}$

Observación 5.15. Dado un término sin etiquetas t, siempre existe una variante inicialmente etiquetada t^ℓ de t.

Ejemplo 5.16 (Términos inicialmente etiquetados)**.** *El término etiquetado* $(\lambda^{\mathbf{a}}_{\{\mathbf{c}\}} x.@^{\mathbf{b}}(x^{\mathbf{c}}, y^{\mathbf{d}}))[y\backslash z^{\mathbf{e}}]_{\{\mathbf{e}\}}$ *es una variante inicialmente etiquetada de* $(\lambda x.xy)[y\backslash z]$.

Los términos etiquetados $\lambda^{\mathbf{a}}_{\{\mathbf{c},\mathbf{d}\}} x.@^{\mathbf{b}}(x^{\mathbf{c}}, x^{\mathbf{d}})$ *y* $y^{\mathbf{a}}[x\backslash z^{\mathbf{b}}]_{\{\otimes\}}$ *están inicialmente etiquetados.*

Los términos etiquetados $x^\bullet$ *y* $x^\otimes$ *no están inicialmente etiquetados porque las etiquetas iniciales distinguidas* $\bullet$ *y* $\otimes$ *no pueden decorar subtérminos.*

Los términos etiquetados $@^{\mathbf{a}}(x^{\mathbf{b}}, x^{\mathbf{b}})$ *y* $@^{\mathbf{a}}(x^{\mathbf{b}}, x^{\mathbf{a}})$ *no están inicialmente etiquetados porque diferentes subtérminos deben tener diferentes etiquetas.*

Los términos etiquetados $\lambda^{\mathbf{a}}_{\{\mathbf{b},\mathbf{c}\}} x.x^{\mathbf{b}}$ *y* $y^{\mathbf{a}}[x \backslash z^{\mathbf{b}}]_{\{\otimes\}}$ *no están inicialmente etiquetados porque los conjuntos de etiquetas sobre los ligadores deberían coincidir con el conjunto de etiquetas hoja de la variable ligada.*

El siguiente lema sencillo (Lem. 5.18) afirma que los nombres de los pasos que salen de un término inicialmente etiquetado tienen una forma particular.

Definición 5.17 (Nombres de redexes iniciales)**.** Un nombre de redex μ se dice *inicial* de acuerdo con la siguiente definición por casos:

1. Un nombre de redex `db` es inicial si es de la forma $\mathtt{db}(\mathbf{a})$ con $\mathbf{a} \in \mathcal{I}$.
2. Un nombre de redex `ls` es inicial si es de la forma $\mathbf{a} \bullet \mathbf{b}$ con $\mathbf{a}, \mathbf{b} \in \mathcal{I}$.
3. Un nombre de redex `gc` es inicial si es de la forma $\{\otimes \bullet \mathbf{a}\}$.

Lema 5.18 (Los nombres de redexes de un término inicialmente etiquetado son iniciales)**.** *Sea* $t \in \mathcal{T}^{\ell}$ *un término inicialmente etiquetado. Sea* μ *el nombre de un redex de* t*. Entonces* μ *es inicial.*

Lema 5.19 (Propiedad **Inicial**)**.** *Sea* $t \in \mathcal{T}^{\ell}$ *un término inicialmente etiquetado. Si* $R : t \xrightarrow{\mu}_{\ell} s$ *y* $S : t \xrightarrow{\nu}_{\ell} u$ *son diferentes pasos, entonces* $\mu \neq \nu$*.*

A continuación definimos la noción de *término correctamente etiquetado*, invariante que verifican los términos inicialmente etiquetados después de aplicar pasos de reducción etiquetada.

Definición 5.20 (Términos correctamente etiquetados)**.** Una etiqueta $\alpha \in \mathcal{L}$ es *buena*, lo que escribimos $\checkmark(\alpha)$, de acuerdo con la siguiente definición inductiva:

$$\begin{array}{rcl}
\mathbf{a} \notin \{\bullet, \otimes\} & \Longrightarrow & \checkmark(\mathbf{a}) \\
\checkmark(\alpha) \wedge \checkmark(\beta) & \Longrightarrow & \checkmark(\alpha\,\beta) \\
\checkmark(\alpha) \wedge \checkmark(\beta) & \Longrightarrow & \checkmark(\alpha \bullet \beta) \\
\checkmark(\alpha) & \Longrightarrow & \checkmark([\alpha]) \\
\checkmark(\alpha) & \Longrightarrow & \checkmark(\lfloor\alpha\rfloor) \\
\checkmark(\alpha) & \Longrightarrow & \checkmark(\mathtt{db}(\alpha))
\end{array}$$

Un conjunto de variables iniciales Ω es bueno, lo que escribimos $\checkmark(\Omega)$, si es no vacío, no contiene ocurrencias de $\bullet$, y no contiene ocurrencias de $\otimes$ a menos que sea precisamente $\{\otimes\}$. Formalmente:

$$\checkmark(\Omega) \overset{\text{def}}{\Longleftrightarrow} (\Omega \neq \varnothing) \wedge (\bullet \notin \Omega) \wedge (\otimes \notin \Omega \vee \Omega = \{\otimes\})$$

Un término $t \in \mathcal{T}^{\ell}$ es bueno, lo que escribimos $\checkmark(t)$, si todas las etiquetas y conjuntos de etiquetas son buenos. Más precisamente:

$$\begin{array}{lcl}
\checkmark(\alpha) & \Longrightarrow & \checkmark(x^{\alpha}) \\
\checkmark(\alpha) \wedge \checkmark(\Omega) \wedge \checkmark(t) & \Longrightarrow & \checkmark(\lambda^{\alpha}_{\Omega} x.t) \\
\checkmark(\alpha) \wedge \checkmark(t) \wedge \checkmark(s) & \Longrightarrow & \checkmark(@^{\alpha}(t, s)) \\
\checkmark(t) \wedge \checkmark(s) \wedge \checkmark(\Omega) & \Longrightarrow & \checkmark(t[x \backslash s]_{\Omega})
\end{array}$$

También extendemos la noción de *bueno* a contextos, declarando que vale $\checkmark(\square)$. Observar que $\checkmark(\mathtt{C}\langle t\rangle)$ se verifica si y sólo si tanto $\checkmark(\mathtt{C})$ como $\checkmark(t)$ se verifican.

Un término $t \in \mathcal{T}^\ell$ se dice *correctamente etiquetado* si y sólo si verifica todas las condiciones siguientes:

1. *Bueno:* se verifica $\checkmark(t)$.
2. *Abstracciones correctas:* para cualquier subtérmino $\lambda^\alpha_\Omega x.t'$ se tiene que $\mathsf{vl}_x(t') \subseteq \Omega$.
3. *Sustituciones correctas:* para cualquier subtérmino $t'[x\backslash s']_\Omega$ se tiene que $\mathsf{vl}_x(t') \subseteq \Omega$.

Para los ítems 2 y 3, observar que las inclusiones son entre conjuntos, es decir, sólo nos interesa el conjunto subyacente al multiconjunto $\mathsf{vl}_x(t)$.

Ejemplo 5.21 (Términos correctamente etiquetados)**.** *El término etiquetado* $\lambda^{\mathbf{a}\bullet\mathbf{b}}_{\{\lceil\mathtt{db}(\mathbf{d})\rceil,\lfloor\mathtt{db}(\mathbf{e})\rfloor\}} x.x^{\mathbf{c}\lceil\mathtt{db}(\mathbf{d})\rceil}$ *es una variante correctamente etiquetada de* $\lambda x.x$.

Los términos etiquetados $x^{\mathbf{a}\bullet}$ *y* $y^{\mathbf{a}\otimes\mathbf{b}}$ *no están correctamente etiquetados porque* $\mathbf{a}\bullet$ *y* $\mathbf{a}\otimes\mathbf{b}$ *no son etiquetas buenas.*

El término etiquetado $\lambda^{\mathbf{a}}_{\{\otimes\}} x.x^{\mathbf{bcd}}$ *no está correctamente etiquetado porque* $\{\mathbf{d}\}$ *no es subconjunto de* $\{\otimes\}$.

La definición de términos inicialmente y correctamente etiquetados se extiende también a derivaciones. Una derivación $\rho : t \twoheadrightarrow_\ell s$ se dice inicialmente (resp. correctamente) etiquetada si t está inicialmente (resp. correctamente) etiqueada. Por Rem. 5.15, toda derivación $\rho : t \twoheadrightarrow_{\mathsf{LSC}} s$ en el LSC sin etiquetas tiene una variante etiquetada $\rho' : t' \twoheadrightarrow_\ell s'$.

A continuación demostramos que la noción de término correctamente etiquetado es invariante por la relación de reescritura $(\to_\ell)$.

Observación 5.22. Todos los términos inicialmente etiquetados también están correctamente etiquetados.

Lema 5.23 (La reducción preserva el correcto etiquetado)**.** *Sea* $t \in \mathcal{T}^\ell$ *un término correctamente etiquetado y* $t \to_\ell s$. *Entonces* s *está correctamente etiquetado.*

Definición 5.24 (Términos inicialmente alcanzables)**.** Un término t se dice *inicialmente alcanzable* si hay un término inicialmente etiquetado t_0 tal que $t_0 \twoheadrightarrow_\ell t$.

Observación 5.25 (Los términos inicialmente etiquetados están correctamente etiquetados). Consecuencia inmediata de Lem. 5.23.

Morfismos de etiquetado

A veces resulta útil o necesario *renombrar* etiquetas. Por ejemplo, la siguiente derivación en el LLSC:

$$(x^{\mathbf{a}}\, x^{\mathbf{b}})[x\backslash z^{\mathbf{c}}]_{\{\mathbf{a},\mathbf{b}\}} \xrightarrow{\mathbf{a}\bullet\mathbf{c}}_\ell (z^{\mathbf{a}\bullet\mathbf{c}}\, x^{\mathbf{b}})[x\backslash z^{\mathbf{c}}]_{\{\mathbf{a},\mathbf{b}\}}$$

Se puede renombrar, mandando la etiqueta **a** a **d** • **e**, la etiqueta **b** a **d** • **e** y la etiqueta **c** a **f** • **g**, obteniendo:

$$(x^{\mathbf{d}\bullet\mathbf{e}}\, x^{\mathbf{d}\bullet\mathbf{e}})[x\backslash z^{\mathbf{f}\bullet\mathbf{g}}]_{\{\mathbf{d}\}} \xrightarrow{\mathbf{e}\bullet\mathbf{f}}_{\ell} (z^{\mathbf{d}\bullet\mathbf{e}\bullet\mathbf{f}\bullet\mathbf{g}}\, x^{\mathbf{d}\bullet\mathbf{e}})[x\backslash z^{\mathbf{f}\bullet\mathbf{g}}]_{\{\mathbf{d}\}}$$

Este mecanismo se formaliza con la siguiente noción de morfismo:

Definición 5.26 (Morfismo de etiquetado)**.** Un *morfismo de etiquetado* ϕ es una función $\phi : \mathcal{L} \to \mathcal{L}$ homomórfica en todos los constructores de etiquetas, salvo por las etiquetas iniciales:

$$\begin{array}{rclrclrcl}
\phi(\ \bullet\ ) & = & \bullet & \phi(\otimes) & = & \otimes & \phi(\mathtt{db}(\alpha)) & = & \mathtt{db}(\phi(\alpha)) \\
\phi(\lceil\alpha\rceil) & = & \lceil\phi(\alpha)\rceil & \phi(\lfloor\alpha\rfloor) & = & \lfloor\phi(\alpha)\rfloor & \phi(\alpha\,\beta) & = & \phi(\alpha)\,\phi(\beta)
\end{array}$$

Si Ω es un conjunto de etiquetas, escribimos $\phi(\Omega)$ para denotar $\{\downarrow(\phi(\Omega)) \mid \alpha \in \Omega\}$. El dominio de los morfismos de etiquetado se extiende de tal modo que se puedan aplicar a términos, como sigue:

$$\begin{array}{rclrcl}
\phi(x^{\alpha}) & = & x^{\phi(\alpha)} & \phi(\lambda^{\alpha}_{\Omega} x.t) & = & \lambda^{\phi(\alpha)}_{\phi(\Omega)} x.\phi(t) \\
\phi(@^{\alpha}(t,s)) & = & @^{\phi(\alpha)}(\phi(t),\phi(s)) & \phi(t[x\backslash s]_{\Omega}) & = & \phi(t)[x\backslash\phi(s)]_{\phi(\Omega)}
\end{array}$$

Los morfismos de etiquetado se pueden aplicar también a contextos, declarando que $\phi(\Box) = \Box$, y sobre nombres de redexes, como sigue:

$$\begin{array}{rcl}
\phi(\mathtt{db}(\alpha)) & = & \mathtt{db}(\phi(\alpha)) \\
\phi(\alpha\ \bullet\ \beta) & = & \downarrow(\phi(\alpha))\ \bullet\ \uparrow(\phi(\beta)) \\
\phi(\{\mathbf{a}\ \bullet\ \beta \mid \mathbf{a} \in \Omega\}) & = & \{\mathbf{a}\ \bullet\ \uparrow(\phi(\beta)) \mid \mathbf{a} \in \phi(\Omega)\}
\end{array}$$

Observación 5.27. Un morfismo de etiquetado queda unívocamente determinado por su valor en el conjunto de etiquetas iniciales $\mathcal{I}$.

Lema 5.28. *Si ϕ es un morfismo de etiquetado, valen las siguientes propiedades para cualquier etiqueta $\alpha \in \mathcal{L}$ y cualquier término $t \in \mathcal{T}^{\ell}$:*

1. $\downarrow(\phi(\alpha)) = \downarrow(\phi(\downarrow(\alpha)))$
2. $\uparrow(\phi(\alpha)) = \uparrow(\phi(\uparrow(\alpha)))$
3. $\uparrow(\phi(t)) = \uparrow(\phi(\uparrow(t)))$

Proposición 5.29 (Los morfismos de etiquetado son funtoriales)**.** *Sea ϕ un morfismo de etiquetado. Entonces para cada paso $R : t \xrightarrow{\mu}_{\ell} s$ hay un paso $\phi(R) : \phi(t) \xrightarrow{\phi(\mu)}_{\ell} \phi(s)$.*

Como consecuencia, se pueden aplicar morfismos de etiquetado a derivaciones, definiendo $\phi(R_1 \ldots R_n) = \phi(R_1) \ldots \phi(R_n)$.

5.3.2. Ortogonalidad

En esta sección demostramos que el LLSC es confluente. A decir verdad, demostramos una propiedad más fuerte, la de que el LLSC forma un sistema de reescritura axiomático ortogonal en el sentido de Melliès.

Comenzamos demostrando que el LLSC es débilmente Church–Rosser, es decir, que todo diagrama $\leftarrow\rightarrow$ formado por dos pasos se puede cerrar con cero o más pasos $\twoheadrightarrow\twoheadleftarrow$. En el cálculo etiquetado vale el siguiente resultado más fuerte, porque afirma que el diagrama se cierra *con pasos del mismo nombre*:

Proposición 5.30 (Permutación fuerte)**.** *Sean $R : t \xrightarrow{\mu}_\ell s$ y $S : t \xrightarrow{\nu}_\ell u$ pasos en el LLSC. Entonces existen un término r y dos derivaciones $\sigma : s \overset{\nu}{\twoheadrightarrow}_\ell r$ y $\rho : u \overset{\mu}{\twoheadrightarrow}_\ell r$. Gráficamente:*

$$\begin{tikzcd} t \arrow[r, "\mu"] \arrow[d, "\nu"'] & s \arrow[d, dotted, two heads, "\nu"] \\ u \arrow[r, dotted, two heads, "\mu"] & r \end{tikzcd}$$

Además, la demostración es constructiva y:

1. *Si R es un paso* `db`*, σ consta de exactamente un paso.*
2. *Si R es un paso* `ls`*, σ puede constar de uno o dos pasos.*
3. *Si R es un paso* `gc`*, σ puede constar de cero o un pasos.*

Y simétricamente para S y ρ.

Recordar que el LSC con su relación de residuo usual forma un sistema de reescritura axiomático ortogonal (Prop. 5.3). Dotamos al LLSC de una relación de residuo usando la relación de residuo del LSC, como se indica a continuación.

Definición 5.31 (Residuos para el LLSC)**.** Si $t \in \mathcal{T}^\ell$ es un término etiquetado, notamos $|t| \in \mathcal{T}$ para el término sin etiquetas que resulta de eliminar todas las etiquetas de t. Análogamente, si $R : t \to_\ell s$ es un paso etiquetado en el LLSC, escribimos $|R| : |t| \to_{\mathsf{LSC}} |s|$ para el correspondiente paso en el LSC, a través de la biyección obvia.

Sea $R : t \to_\ell s$ un paso etiquetado y sean $S_1 : t \to_\ell u$ y $S_2 : s \to_\ell r$ dos pasos etiquetados. Declaramos que se verifica la relación de residuos $S_1 \langle R \rangle S_2$ en el LLSC si y sólo si la relación de residuos usual $|S_1| \langle |R| \rangle |S_2|$ se verifica en el LSC.

Proposición 5.32 (Ortogonalidad)**.** *Con la noción de residuo dada arriba, el LLSC forma un sistema de reescritura axiomático ortogonal.*

Un corolario importante del resutlado de ortogonalidad es que el etiquetado es consistente con la equivalencia por permutación.

Proposición 5.33 (Las derivaciones equivalentes por permutación dan lugar a las mismas etiquetas)**.** *Sean ρ_1 y ρ_2 derivaciones equivalentes por permutación, es decir, $\rho_1 \equiv \rho_2$. Sean ρ_1^ℓ y ρ_2^ℓ variantes etiquetadas de ρ_1 y ρ_2 respectivamente tales que $\mathsf{src}(\rho_1^\ell) = \mathsf{src}(\rho_2^\ell)$, es decir, comienzan en el mismo término etiquetado. Entonces $\mathsf{tgt}(\rho_1^\ell) = \mathsf{tgt}(\rho_2^\ell)$, es decir, terminan en el mismo término etiquetado.*

5.3.3. Normalización débil de la reducción acotada

En esta sección demostramos que si la relación de reescritura del LLSC se restringe de tal forma que la *altura* de los nombres de los pasos esté acotada, la relación de reescritura obtenida resulta ser débilmente normalizante.

Debajo introducimos los cálculos auxiliares LLSC^P y LLSC^{PI}, que sólo permiten contraer un paso R si el nombre de R verifica un predicado P. Introducimos también la noción de *predicado acotado.*

Definición 5.34 (El LLSCP-restringido)**.** Sea P un predicado sobre los nombres de los redexes. Definimos dos cálculos, LLSC^P y LLSC^{PI}. El conjunto de términos es $\mathcal{T}^\ell$ en ambos casos. La relación de reducción $\to_{\ell P}$ se define tal como en el LLSC, restringida a contraer sólo pasos cuyos nombres verifican el predicado P. La relación $\to_{\ell PI}$ se define de manera similar, pero se restringe a contraer pasos `db` y `ls`:

$$\begin{array}{rcl} t \xrightarrow{\mu}_{\ell P} s & \overset{\text{def}}{\iff} & \text{se verifica } t \xrightarrow{\mu}_{\ell} s \ \wedge \ P(\mu) \\ t \xrightarrow{\mu}_{\ell PI} s & \overset{\text{def}}{\iff} & \text{se verifica } (t \xrightarrow{\mu}_{\ell\,\mathtt{db}} s \ \vee \ t \xrightarrow{\mu}_{\ell\,\mathtt{ls}} s) \ \wedge \ P(\mu) \end{array}$$

Observar que, dado que no hay pasos `gc`, el nombre de un paso en el cálculo LLSC^{PI} se puede entender siempre como una etiqueta. Escribimos $t \to_{\ell P} s$ si hay un paso $t \xrightarrow{\mu}_{\ell P} s$ para algún nombre de redex μ, y análogamente para $\to_{\ell PI}$.

Definición 5.35 (Altura de etiquetas y nombres de redexes)**.** Definimos la altura de una etiqueta como sigue:

$$\begin{array}{rcl} h(\mathbf{a}) & \overset{\text{def}}{=} & 1 \\ h(\lceil\alpha\rceil) = h(\lfloor\alpha\rfloor) = h(\mathtt{db}(\alpha)) & \overset{\text{def}}{=} & 1 + h(\alpha) \\ h(\alpha\beta) & \overset{\text{def}}{=} & \text{máx}\{h(\alpha), h(\beta)\} \end{array}$$

De manera similar, definimos la altura de un nombre de redex como sigue:

$$\begin{array}{llrcll} \text{Redex } \mathtt{db}: & & h(\mathtt{db}(\alpha)) & \overset{\text{def}}{=} & 1 + h(\alpha) & \\ \text{Redex } \mathtt{ls}: & & h(\alpha \bullet \beta) & \overset{\text{def}}{=} & \text{máx}\{h(\alpha), h(\beta)\} & \text{donde } \alpha, \beta \text{ son atómicas} \\ \text{Redex } \mathtt{gc}: & & h(\{\mathbf{a} \bullet \beta \mid \mathbf{a} \in \Omega\}) & \overset{\text{def}}{=} & \text{máx}\{h(\mathbf{a} \bullet \beta) \mid \mathbf{a} \in \Omega\} & \text{donde } \beta \text{ es atómica} \end{array}$$

Observar que en el caso de redexes `db` y `ls`, la altura del nombre del redex coincide con su altura si se lo entiende como una etiqueta.

Definición 5.36 (Predicado acotado)**.** Un predicado P sobre nombres de redexes se dice *acotado* si y sólo si existe una cota $H \in \mathbb{N}$ tal que para todo nombre de redex μ, si vale $P(\mu)$ entonces $h(\mu) < H$.

Por el momento ignoramos los pasos `gc` y trabajamos únicamente con el cálculo LLSC^{PI}. Nuestro objetivo es demostrar que la relación de reescritura $\xrightarrow{\mu}_{\ell PI}$ es débilmente normalizante si P es acotado. Este es el contenido de la Prop. 5.43 de abajo. Esbozamos la estructura de la demostración:

1. En Prop. 5.39, demostramos que el LLSC sin gc verifica el principio de **creación**, es decir, si un redex R crea a otro redex S, el nombre de R es un "subnombre" de S.

2. En Lem. 5.41, demostramos que entre los pasos LLSC^{PI} que salen de un término dado, hay al menos un paso *no duplicante*.

3. En Lem. 5.42, demostramos que contraer un paso a no duplicante tiene el siguiente efecto:

   3.1 Preserva a todos los pasos existentes (es decir, tienen exactamente un residuo).

   3.2 Los redexes creados tienen nombres *más altos* que el redex que se contrajo, es decir, la altura de la etiqueta se incrementa.

Con estos resultados es posible definir una medida sobre los términos que siempre decrece cuando se elige contraer un redex no duplicante.

Principio de creación

La siguiente relación de *contribución entre nombres* corresponde a la idea informal de que el nombre de un redex es un "subnombre", o sea, está contenido, en el nombre de otro redex.

Definición 5.37 (Contribución de nombres)**.** Se dice que un nombre de redex μ *contribuye directamente* a un nombre de redex ν, y se escribe $\mu \overset{\mathsf{Name}}{\hookrightarrow}_1 \nu$, si se da alguno de los tres casos siguientes:

$$\begin{array}{rcll} \mathtt{db}(\beta) & \overset{\mathsf{Name}}{\hookrightarrow}_1 & \mathtt{db}(\alpha\,\lceil\mathtt{db}(\beta)\rceil\,\gamma) & \\ \mathtt{db}(\beta) & \overset{\mathsf{Name}}{\hookrightarrow}_1 & \alpha \bullet \lfloor\mathtt{db}(\beta)\rfloor & \text{donde } \alpha \text{ es cualquier etiqueta atómica} \\ \downarrow(\alpha) \bullet \uparrow(\beta) & \overset{\mathsf{Name}}{\hookrightarrow}_1 & \mathtt{db}(\alpha \bullet \beta) & \end{array}$$

Se dice que un nombre de redex μ *contribuye* (indirectamente) a un nombre de redex ν, y se escribe $\mu \overset{\mathsf{Name}}{\hookrightarrow} \nu$, si $\mu(\overset{\mathsf{Name}}{\hookrightarrow}_1)^*\nu$.

Observación 5.38. Si $\mu \overset{\mathsf{Name}}{\hookrightarrow}_1 \nu$ entonces $h(\mu) < h(\nu)$.

Proposición 5.39 (Propiedad de **Creación** para el LLSC sin gc)**.** *Sean $t \xrightarrow{\mu}_\ell s \xrightarrow{\nu}_\ell u$ dos pasos consecutivos, cada uno de los cuales puede ser un paso* db *o* ls*, pero no* gc*. Si el primer paso crea al segundo, entonces $\mu \overset{\mathsf{Name}}{\hookrightarrow}_1 \nu$.*

Pasos no duplicantes

Definición 5.40 (Paso no duplicante)**.** Dado un sistema de reescritura axiomático con una noción de residuo, un paso $R : t \to s$ se dice *no duplicante* si todo paso coinicial $S : t \to u$ tiene a lo sumo un residuo después de R, es decir, $\#(S/R) \leqslant 1$.

Lema 5.41 (Existencia de pasos $\to_{\ell\,PI}$ no duplicantes)**.** *Sea $t \in \mathcal{T}^\ell$ un término tal que no está en $\to_{\ell\,PI}$-forma normal. Entonces t tiene al menos un redex $\to_{\ell\,PI}$ no duplicante.*

Lema 5.42 (Efecto de contraer un paso no duplicante)**.** *Sea* $\mathsf{names}_{PI}(t)$ *el multiconjunto de nombres de pasos* $\to_{\ell\ PI}$ *que salen de* t*. Sea* $R : t \xrightarrow{\alpha}_{\ell\ PI} s$ *un paso no duplicante. Entonces existen multiconjuntos de etiquetas* $\mathfrak{m}$ *y* $\mathfrak{n}$ *tales que:*

$$\begin{array}{rcl} \mathsf{names}_{PI}(t) & = & \mathfrak{m} \uplus \{\alpha\} \\ \mathsf{names}_{PI}(s) & = & \mathfrak{m} \uplus \mathfrak{n} \end{array}$$

y además $h(\alpha) < h(\beta)$ *para toda etiqueta* $\beta \in \mathfrak{n}$*. Observar que* α *es el nombre del paso que se contrae y* $\mathfrak{n}$ *son los nombres de los pasos que se crean.*

Normalización débil para reducción acotada sin gc

El siguiente es el resultado principal de esta sección:

Proposición 5.43 (La reducción acotada es débilmente normalizante)**.** *Si* P *es un predicado acotado, entonces* $\to_{\ell\ PI}$ *es* WN.

5.3.4. Normalización fuerte de la reducción acotada

En esta sección fortalecemos el resultado de normalización de Prop. 5.43, probando que el LLSC en su totalidad (incluyendo la regla gc) es *fuertemente* normalizante, en tanto que se restrinja la reducción a redexes cuyo nombre tenga altura acotada. La estructura de la demostración es la siguiente:

1. Primero demostramos que $\to_{\ell\ PI}$ es *increasing* (Lem. 5.46). Recordar que una relación de reescritura $\to \subseteq X^2$ se dice *increasing* si existe una función $f : X \to \mathbb{N}$ tal que $x \to y$ implica $f(x) < f(y)$.

2. Usando la propiedad anterior, concluimos en Lem. 5.47 que $\to_{\ell\ PI}$ es fuertemente normalizante si P es un predicado acotado.

3. Por último, usando un lema técnico que permite postponer los pasos gc (Lem. 5.48), obtenemos el resultado principal (Teo. 5.49), que afirma que la relación de reducción para el LLSC es SN, para nombres redexes de altura acotada.

Normalización fuerte para reducción acotada sin gc

Definición 5.44 (Medida de un término etiquetado)**.** Definimos el *tamaño de una etiqueta* de la siguiente manera:

$$\begin{array}{rcl} \|\mathbf{a}\| & \stackrel{\text{def}}{=} & 1 \\ \|\lceil\alpha\rceil\| = \|\lfloor\alpha\rfloor\| = \|\mathtt{db}(\alpha)\| & \stackrel{\text{def}}{=} & 1 + \|\alpha\| \\ \|\alpha\,\beta\| & \stackrel{\text{def}}{=} & \|\alpha\| + \|\beta\| \end{array}$$

Dado un término etiquetado, su *medida* $\|t\|$ se define como la suma de los tamaños de todas sus etiquetas:

$$\begin{array}{rcl} \|x^\alpha\| & \stackrel{\text{def}}{=} & \|\alpha\| \\ \|\lambda^\alpha_\Omega x.t\| & \stackrel{\text{def}}{=} & \|\alpha\| + \|t\| \\ \|@^\alpha(t,s)\| & \stackrel{\text{def}}{=} & \|\alpha\| + \|t\| + \|s\| \\ \|t[x\backslash s]_\Omega\| & \stackrel{\text{def}}{=} & \|t\| + \|s\| \end{array}$$

La medida de un contexto $\|\mathtt{C}\|$ se define de manera similar, declarando $\|\Box\| \stackrel{\text{def}}{=} 0$.

Lema 5.45 (Propiedades de la medida de un término)**.**

1. $\|\mathtt{C}\langle t\rangle\| = \|\mathtt{C}\| + \|t\|$
2. $\|\alpha : t\| = \|\alpha\| + \|t\|$

Lema 5.46 (La reducción etiquetada sin gc es *increasing*)**.** *Si* $t \to_{\ell\, PI} s$ *entonces* $\|t\| < \|s\|$.

La demostración del siguiente lema se basa en el lema de Klop-Nederpelt, que afirma que WCR $\wedge$ WN $\wedge$ Inc $\implies$ SN:

Lema 5.47 (La reducción acotada en el cálculo sin gc es fuertemente normalizante)**.** *Sea* P *un predicado acotado. Entonces* $\to_{\ell\, PI}$ *es* SN.

Normalización fuerte para reducción acotada con gc

Para extender el resultado de normalización fuerte a todo el cálculo, incluyendo la regla gc, necesitamos el siguiente lema ténico que permite *postponer* pasos gc.

Lema 5.48 (Postposición de gc en el cálculo LLSC)**.** *Sea* $\rho : t \twoheadrightarrow_\ell s$ *una reducción. Entonces existen un término* u *y una reducción* $\sigma : t \twoheadrightarrow_{\ell\, \mathtt{db} \cup \mathtt{ls}} u \twoheadrightarrow_{\ell\, \mathtt{gc}} s$. *Además, escribimos* $\#_\mu(\rho)$ *para denotar el número de redexes con nombre* μ *que se contraen a lo largo de una reducción* ρ. *Con esta notación, tenemos que:*

1. *El número de pasos* db *y* ls *se preserva:*

$$\#_\mu(\rho) = \#_\mu(\sigma) \text{ si } \mu \text{ es un nombre de redex } \mathtt{db} \text{ o } \mathtt{ls}$$

2. *El número de pasos* gc *puede aumentar:*

$$\#_\mu(\rho) \leqslant \#_\mu(\sigma) \text{ si } \mu \text{ es un nombre de redex } \mathtt{gc}$$

3. *La reducción* σ *contrae los mismos nombres que* ρ*:*

$$\#_\mu(\sigma) > 0 \implies \#_\mu(\rho) > 0 \text{ para cualquier nombre de redex } \mu$$

El siguiente es el resultado principal de esta sección:

Teorema 5.49 (La reducción acotada en el cálculo LLSC^P es fuertemente normalizante)**.** *Sea* P *un predicado acotado. Entonces* $\to_{\ell\, P}$ *es* SN.

5.3.5. Confluencia

En esta sección damos dos demostraciones de que el LLSC es confluente. Las dos demostraciones son consecuencia de hechos ya previamente establecidos. La primera demostración se basa en métodos puramente sintácticos, en tanto que la segunda se basa en teoría de residuos.

Lema 5.50 (La reducción acotada en LLSC^P es confluente)**.** *Sea P un predicado acotado. Entonces la relación de reescritura $\to_{\ell\, P}$ es Church–Rosser.*

Teorema 5.51 (El LLSC es confluente)**.** *La relación de reescritura $\to_\ell$ es Church–Rosser.*

Demostración.

Primera demostración. Si $\rho : t \twoheadrightarrow_\ell s$ y $\sigma : t \twoheadrightarrow_\ell u$, definimos $P(\mu)$ de tal manera que valga si μ es el nombre de un redex que se contrae en ρ o σ. Como el número de etiquetas es finito, P es acotado y por el resultado anterior (Lem. 5.50) concluímos.

Segunda demostración. Ya hemos demostrado que el LLSC es un sistema de reescritura axiomático ortogonal (Prop. 5.32). Además, los sistemas de reescritura axiomáticos ortogonales tienen la propiedad de *confluencia algebraica*, que a su vez implica confluencia. □

Capítulo 6

Aplicaciones del LSC etiquetado

6.1. Introducción

Este capítulo es fruto de la colaboración con Eduardo Bonelli, y se estructura de la siguiente manera. Destacamos en negrita las principales contribuciones:

1. En el capítulo anterior (Rem. 5.8), vimos que el LSC con la regla gc no cumple con la propiedad conocida como *estabilidad*. En la Sección 6.2, usando el LLSC como herramienta, demostramos que, en cambio, **el LSC sin la regla** gc **cumple con la propiedad de estabilidad** (Prop. 6.1).

2. En la Sección 6.3 recordamos la noción de Deterministic Family Structure (DFS) de Glauert y Khasidashvili. A continuación, usando el LLSC como herramienta, demostramos que **el LSC sin** gc **forma una Deterministic Family Structure** (Teo. 6.13). En particular, la propiedad de normalización fuerte para reducción acotada del LLSC corresponde a la propiedad conocida como *Finite Family Developments* para el LSC sin gc.

3. En la Sección 6.4 recordamos el resultado de *optimalidad* de Lévy para el cálculo-λ-calculus (Teo. 6.17), repasamos el resultado de optimalidad abstracta de Glauert y Khasidashvili (Teo. 6.24), y, como corolario, obtenemos un resultado de optimalidad para el LSC sin gc.

4. En la Sección 6.5 recordamos en qué consiste el problema de *estandarización*, y proponemos un **procedimiento de estandarización para Deterministic Family Structures** (Prop. 6.39), inspirado en un resultado de estandarización de Klop. Como corolario, obtenemos un resultado de estandarización para el LSC sin gc (Coro. 6.43).

5. En la Sección 6.6 recordamos la noción de *normalización*, y demostramos un **resultado de normalización para Deterministic Family Structures** (Prop. 6.54): damos condiciones suficientes bajo las cuales una estrategia de reducción es normalizante. Como corolario, concluimos que, en el LSC sin gc, la estrategia call-by-name (Coro. 6.56) y una variante de la estrategia call-by-need (Coro. 6.59) son normalizantes.

6.2. Estabilidad

Recordar que, dado un sistema de reescritura axiomático ortogonal, se dice que verifica la propiedad de estabilidad (Def. 5.7) si dos pasos que tienen un residuo común siempre tienen también un ancestro común. Gráficamente:

$$\begin{tikzcd}
 & T_0 \arrow[dl, "R"'] \arrow[dr, "S"] & \\
T_1 \arrow[dr, "S/R"'] & & T_2 \arrow[dl, "R/S"] \\
 & T_3 \arrow[d] & \\
 & {} &
\end{tikzcd}$$

En Rem. 5.8 observamos que el LSC con la regla gc no cumple con la propiedad de estabilidad. Por otra parte:

Proposición 6.1. *El LLSC sin* gc *cumple con la propiedad de estabilidad.*

6.3. Familias de redexes

En esta sección estudiamos *familias de redexes* en el LSC sin gc. Comenzamos recordando notación y algunas definiciones:

- Un sistema de reescritura axiomático tiene la propiedad de *ancestro único* (UA) si un paso tiene a lo sumo un ancestro, es decir, cada vez que $R_1 \langle S \rangle R$ y $R_2 \langle S \rangle R$ se tiene que $R_1 = R_2$.
- Un sistema de reescritura axiomático tiene la propiedad de *aciclicidad* si dos pasos no pueden borrarse mutuamente, es decir, cada vez que $R \neq S$ y $R/S = \varnothing$ se tiene que $S/R \neq \varnothing$.
- En un sistema de reescritura, un *redex con historia* o *hredex* es una derivación no vacía. Generalmente nos interesa el último paso del hredex, de modo que los hredexes típicamente se escriben como de la forma ρR donde ρ es una derivación posiblemente vacía y R es un paso componible. El conjunto de hredexes cuyo origen es un objeto x se nota $\mathsf{Hist}(x)$.
- En un sistema de reescritura axiomático ortogonal escribimos $\rho \equiv \sigma$ si ρ y σ son derivaciones equivalentes por permutación.

Damos también una definición formal de *copia*:

Definición 6.2 (Relación de copia)**.** Sean ρR y σS hredexes coiniciales en un sistema de reescritura axiomático ortogonal arbitrario. Decimos que σS es una *copia* de ρR, y escribimos $\rho R \leqslant \sigma S$, si existe una derivación τ tal que $\rho\tau \equiv \sigma$ y $R \langle \tau \rangle S$. Gráficamente:

$$\begin{tikzcd}
 & {} \arrow[dl, "\rho"'] \arrow[dr, "\sigma"] & \\
{} \arrow[dl, "R"'] & & {} \arrow[dr, "S"] \\
 & {} \arrow[rr, dotted, "\tau"] & & {}
\end{tikzcd}$$

Definición 6.3 (Deterministic Family Structure)**.** Una *Deterministic Residual Structure* (DRS) es un sistema de reescritura axiomático ortogonal que verifica además las propiedades de ancestro único (UA) y aciclicidad.

Una *Deterministic Family Structure* (DFS) es una tripla $\langle \mathcal{A}, \simeq, \hookrightarrow \rangle$, donde $\mathcal{A}$ es una Deterministic Residual Structure, $\simeq$ es una relación de equivalencia entre hredexes coiniciales cuyas clases de equivalencia se llaman *familias*, y $\hookrightarrow$ es una relación binaria de *contribución* entre familias coiniciales. Dos familias se dicen coiniciales si sus representantes lo son. La familia de un hredex ρR se escribe $\mathsf{Fam}_{\simeq}(\rho R)$. Además, se deben verificar los siguientes axiomas:

1. Inicial. Si R, S son pasos coiniciales distintos, entonces $\mathsf{Fam}_{\simeq}(R) \neq \mathsf{Fam}_{\simeq}(S)$.

2. Copia. Se verifica la inclusión $(\leqslant) \subseteq (\simeq)$.

3. Finite Family Developments (FFD). Cualquier derivación que contrae hredexes de un número finito de familias es finita. Más precisamente, no puede existir una derivación infinita $R_1 R_2 \ldots R_n \ldots$ tal que el conjunto $\{\mathsf{Fam}_{\simeq}(R_1 \ldots R_n) \mid n \in \mathbb{N}\}$ es finito.

4. Creación. Si ρR es un hredex y R crea S, entonces $\mathsf{Fam}_{\simeq}(\rho R) \hookrightarrow \mathsf{Fam}_{\simeq}(\rho R S)$.

5. Contribución. Dadas dos familias coiniciales $\phi_1, \phi_2 \in \mathsf{Hist}(t)/\simeq$, la relación $\phi_1 \hookrightarrow \phi_2$ se verifica si y sólo si para todo hredex $\sigma S \in \phi_2$, existe un hredex $\rho R \in \phi_1$ tal que ρR es un prefijo de σ (es decir, $\sigma = \rho R \sigma'$).

En términos prácticos, una vez que se cuenta con un etiquetado de Lévy para un cálculo, la demostración de que el cálculo verifica los axiomas Inicial, Copia y Creación en la definición de DFS debería ser una demostración técnica pero directa. En contraste, los axiomas Finite Family Developments y Contribution tienen una demostración típicamente no trivial.

El LSC con gc no forma una Deterministic Family Structure

Hemos visto en Rem. 5.8 que el LSC (con gc) no cumple con la propiedad de estabilidad. Una consecuencia de este hecho es que el LSC con gc no forma tampoco una Deterministic Family Structure. Para ver esto, alcanza con demostrar la siguiente proposición, que se puede encontrar en el trabajo de Glauert y Khasidashvili [19, Lemma 4.1].

Proposición 6.4. *Si $\langle \mathcal{A}, \simeq, \hookrightarrow \rangle$ es una DFS, entonces $\mathcal{A}$ tiene la propiedad de estabilidad.*

El LSC sin gc forma una Deterministic Family Structure

Esta sección se aboca a la demostración de que el LSC sin gc forma una Deterministic Family Structure. Por definición, una DFS es una tripla $\langle \mathcal{A}, \simeq, \hookrightarrow \rangle$ donde $\mathcal{A}$ es una Deterministic Residual Structure. En efecto se tiene:

Proposición 6.5. *El LSC sin* gc *forma una Deterministic Residual Structure.*

Para demostrar que el LSC sin gc forma una DFS, necesitamos algunos lemas preliminares. Recordemos que $(\overset{\mathsf{Name}}{\hookrightarrow})$ representa la relación de *contribución de nombres* definida en Def. 5.37.

Lema 6.6. *Sea ϕ un morfismo de etiquetado y sean μ y ν dos nombres de redexes (no* gc*). Entonces $\mu \overset{\mathsf{Name}}{\hookrightarrow} \nu$ implica $\phi(\mu) \overset{\mathsf{Name}}{\hookrightarrow} \phi(\nu)$.*

Veremos que el LSC sin gc forma una DFS con las siguientes nociones de familia de redexes y contribución:

Definición 6.7 (Familias de redexes en el LSC sin gc)**.** Sean ρR y σS hredexes coiniciales en el LSC sin gc. Sean $\rho^\ell R^\ell$ y $\sigma^\ell S^\ell$ dos variantes inicialmente etiquetadas de ρR y σS respectivamente, comenzando a partir del mismo término inicialmente etiquetado. Sea μ el nombre de R^ℓ y sea ν el nombre de S^ℓ. Entonces:

- **Relación de famila.** Declaramos que vale $\rho R \overset{\mathsf{Fam}}{\simeq} \sigma S$ si y sólo si $\mu = \nu$.
- **Relación de contribution.** Declaramos que vale $\rho R \overset{\mathsf{Fam}}{\hookrightarrow} \sigma S$ si y sólo si $\mu \overset{\mathsf{Name}}{\hookrightarrow} \nu$.

Ejemplo 6.8 (Familias de redexes y contribución)**.** *En el siguiente diagrama:*

$$\begin{tikzcd}
((\lambda x.x)\, y)[y\backslash z] \arrow[r, "R"] \arrow[d, "T_1"'] & x[x\backslash y][y\backslash z] \arrow[r, "S"] \arrow[d, "T_2"'] & y[x\backslash y][y\backslash z] \\
((\lambda x.x)\, z)[y\backslash z] & x[x\backslash z][y\backslash z] &
\end{tikzcd}$$

tenemos que $T_1 \overset{\mathsf{Fam}}{\simeq} RT_2$ y que $R \overset{\mathsf{Fam}}{\hookrightarrow} RS$. Esto se puede justificar comenzando desde una variante inicialmente etiquetada del término $((\lambda x.x)\, y)[y\backslash z]$ y observando que los nombres de T_1 y RT_2 son ambos $\mathbf{d} \bullet \mathbf{e}$, y que el nombre de R contribuye al nombre de RS, es decir, $\mathtt{db}(\mathbf{b}) \overset{\mathsf{Name}}{\hookrightarrow} \mathbf{c} \bullet \lfloor \mathtt{db}(\mathbf{b}) \rfloor$*:*

$$\begin{tikzcd}
@^{\mathbf{a}}(\lambda^{\mathbf{b}} x.x^{\mathbf{c}}, y^{\mathbf{d}})[y\backslash z^{\mathbf{e}}] \arrow[r, "\mathtt{db}(\mathbf{b})"] \arrow[d, "\mathbf{d}\bullet\mathbf{e}"'] & x^{\mathbf{a}\lceil \mathtt{db}(\mathbf{b})\rceil \mathbf{c}}[x\backslash y^{\lfloor \mathtt{db}(\mathbf{b})\rfloor \mathbf{d}}][y\backslash z^{\mathbf{e}}] \arrow[r, "\mathbf{c}\bullet\lfloor \mathtt{db}(\mathbf{b})\rfloor"] \arrow[d, "\mathbf{d}\bullet\mathbf{e}"'] & y^{\mathbf{a}\lceil \mathtt{db}(\mathbf{b})\rceil \mathbf{c}\bullet\lfloor \mathtt{db}(\mathbf{b})\rfloor \mathbf{d}}[x\backslash y^{\lfloor \mathtt{db}(\mathbf{b})\rfloor \mathbf{d}}][y\backslash z^{\mathbf{e}}] \\
@^{\mathbf{a}}(\lambda^{\mathbf{b}} x.x^{\mathbf{c}}, z^{\mathbf{d}\bullet\mathbf{e}})[y\backslash z^{\mathbf{e}}] & x^{\mathbf{a}\lceil \mathtt{db}(\mathbf{b})\rceil \mathbf{c}}[x\backslash z^{\lfloor \mathtt{db}(\mathbf{b})\rfloor \mathbf{d}\bullet\mathbf{e}}][y\backslash z^{\mathbf{e}}] &
\end{tikzcd}$$

Proposición 6.9 (Las familias de redexes están bien definidas)**.** *Las relaciones $(\overset{\mathsf{Fam}}{\simeq})$ y $(\overset{\mathsf{Fam}}{\hookrightarrow})$ están bien definidas, en el sentido de que no dependen de la elección del etiquetado inicial.*

Es inmediato comprobar que la relación de familia $(\overset{\mathsf{Fam}}{\simeq})$ es una relación de equivalencia. Esto se reduce al hecho de que la igualdad de nombres de redexes es a su vez una relación de equivalencia. Tal como en la definición abstracta de DFS, las clases de equivalencia de $\overset{\mathsf{Fam}}{\simeq}$ se llaman *familias*, y $\mathsf{Fam}_{\simeq}(\rho R)$ representa la familia del hredex ρR.

Definición 6.10 (Relación de contribución entre familias)**.** Dadas dos familias coiniciales ϕ_1, ϕ_2, decimos que ϕ_1 *contribuye* a ϕ_2, y escribimos $\phi_1 \overset{\mathsf{Fam}}{\hookrightarrow} \phi_2$, si y sólo si dados $\rho R \in \phi_1$ y $\sigma S \in \phi_2$ se verifica la condición $\rho R \overset{\mathsf{Fam}}{\hookrightarrow} \sigma S$.

Observar que, por abuso de notación, escribimos $\overset{\mathsf{Fam}}{\hookrightarrow}$ tanto para la relación de contribución entre hredexes y para la relación de contribución entre familias.

Proposición 6.11 (La contribución está bien definida)**.** *La relación de contribución* $\overset{\mathsf{Fam}}{\hookrightarrow}$ *entre familias está bien definida, en el sentido de que no depende de la elección de los representantes.*

En la siguiente proposición enunciamos y demostramos el axioma de Contribución para el LSC sin gc. La demostración se basa en varios lemas técnicos que se omiten:

Proposición 6.12 (Contribution axiom for the LSC without gc)**.** *Sean* $\phi_1, \phi_2 \in \mathsf{Hist}(t)/\overset{\mathsf{Fam}}{\simeq}$ *familias coiniciales en el LSC sin* gc*. Entonces las siguientes propiedades son lógicamente equivalentes:*

1. **Contribución sintáctica.** $\phi_1 \overset{\mathsf{Fam}}{\hookrightarrow} \phi_2$.
2. **Contribución semántica.** *Para cada hredex* $\sigma S \in \phi_2$*, existe un hredex* $\rho R \in \phi_1$ *tal que* ρR *es un prefijo de* σ*.*

Por último, estamos en condiciones de demostrar el teorema principal de esta sección:

Teorema 6.13 (El LSC sin gc forma una DFS)**.** *La tripla* $(\mathcal{A}, \overset{\mathsf{Fam}}{\simeq}, \overset{\mathsf{Fam}}{\hookrightarrow})$ *forma una Deterministic Family Structure, donde* $\mathcal{A}$ *es el DRS construido en Prop. 6.5,* $\overset{\mathsf{Fam}}{\simeq}$ *es la relación de familia ente hredexes coiniciales (Def. 6.7) y* $\overset{\mathsf{Fam}}{\hookrightarrow}$ *es la relación de contribución entre familias (Def. 6.10).*

6.4. Reducción optimal

En las secciones anteriores, dotamos al LSC de una noción de etiquetas de Lévy (Def. 5.6) y usamos esta noción de etiquetado para definir una noción de *familia de redexes* para el LSC sin gc (Def. 6.7): dos redexes están en la misma familia si el esquema de etiquetado les atribuye el mismo nombre. Vimos también que esta noción de familia tiene buenas propiedades, en el sentido de que el LSC sin gc forma una Deterministic Family Structure (Teo. 6.13).

En esta sección, enunciamos el resultado de optimalidad de Lévy en el marco del cálculo-λ. Esto no es estrictamente necesario pero sirve para clarificar el resto de la exposición. En segundo lugar, enunciamos una generalización, debida a Glauert and Khasidashvili, del teorema de optimalidad de Lévy para una Deterministic Family Structure arbitraria. Por último, usando el hecho de que el LSC sin gc forma una Deterministic Family Structure, obtenemos un teorema de optimalidad para el LSC sin gc, lo que significa que ciertos tipos de reducciones son optimales. Para ello estudiamos la noción de forma normal *salvo aplicación de la regla* gc.

Optimalidad en el cálculo-λ

Para enunciar más precisamente el resultado de optimalidad de Lévy, necesitamos dar primero algunas definiciones. Recordar que si $\mathcal{M}$ es un *multipaso*, escribimos $\mathcal{M}$ para referirnos a su development completo canónico, que existe y es único módulo equivalencia por permutación. Las definiciones y resultados en esta subsección se remontan al trabajo de Lévy y se pueden encontrar claramente presentadas en el libro de Asperti y Guerrini [7].

Definición 6.14 (Reducción por familias)**.** Sea $(\mathcal{A}, \simeq, \hookrightarrow)$ una DFS. Una *reducción por familias* es una multiderivación $\mathcal{M}_1 \dots \mathcal{M}_n$ en $\mathcal{A}$ tal que para cada $i \in \{1, \dots, n\}$ todos los pasos en $\mathcal{M}_i$ pertenecen a la misma familia. Más precisamente, para todo $i \in \{1, \dots, n\}$ y para dos pasos $R, S \in \mathcal{M}_i$ cualesquiera se tiene que $\mathcal{M}_1 \dots \mathcal{M}_{i-1} R \simeq \mathcal{M}_1 \dots \mathcal{M}_{i-1} S$. Además, una reducción por familias es *completa* si cada $\mathcal{M}_i$ es un conjunto maximal de pasos que tienen al objeto $\mathsf{src}(\mathcal{M}_i)$ como origen y pertenecen a la misma familia.

La motivación detrás de la definición de Lévy de reducción por familias completa es la idea de que una implementación optimal nunca debería duplicar el trabajo computacional. En cambio, debería *compartir* el trabajo de contraer todas las copias de un mismo redex. Ejecutar un paso de cómputo en una implementación optimal debería corresponder a contraer todos y solamente los redexes de una misma familia.

Ejemplo 6.15 (Reducciones por familias)**.** *Considerar el siguiente diagrama en el LSC sin* gc*:*

$$\begin{tikzcd}
(xx)[x\backslash y][y\backslash z] \arrow[r, "R"] \arrow[d, "S"'] & (yx)[x\backslash y][y\backslash z] \arrow[d, "S'"'] & (zy)[x\backslash y][y\backslash z] \\
(xy)[x\backslash y][y\backslash z] \arrow[r, "R'"] & (yy)[x\backslash y][y\backslash z] \arrow[ur, "T_1"] \arrow[r, "T_2"] \arrow[dr, "T_3"] & (yz)[x\backslash y][y\backslash z] \\
& & (yy)[x\backslash z][y\backslash z]
\end{tikzcd}$$

La multiderivación $\{R, S\}$ (que consta de una secuencia de exactamente un multipaso) no es una reducción por familias, porque R y S no están en la misma familia, en tanto que $\{R\}\{S'\}$ y $\{S\}\{R'\}$ son ambas reducciones por familia completas. La multiderivación $\{R\}\{S'\}\{T_1, T_2\}$ es una reducción por familias, pero no es completa porque el conjunto $\{T_1, T_2\}$ no es un conjunto maximal de pasos coiniciales en la misma familia. La multiderivación $\{R\}\{S'\}\{T_1, T_2, T_3\}$ es una reducción por familias completa.

Comenzando desde un término t, noes interesa encontrar la reducción *optimal*, es decir, la más *corta*.

Definición 6.16 (Reducción optimal)**.** Sea $x \in \mathcal{A}$ un objeto en una DFS. Una reducción por familias que tiene como origen al objeto x y como destino a la forma normal de x de dice *optimal* si su longitud es mínima entre todas las reducciones por familias con dichas características.

Al requerir que una multiderivación sea una reducción por familias completa, aseguramos que nunca se duplica el trabajo computacional. A pesar de ello, una reducción por familias completa podría no ser optimal, porque podría efectuar trabajo computacional innecesario. Por ejemplo, en el cálculo-λ, dado el diagrama:

$$\begin{tikzcd}
(\lambda x.y)\,(I z) \arrow[r, "R"] \arrow[d, "S_1"'] & (\lambda x.y)\, z \arrow[dl, "S_2"] \\
y &
\end{tikzcd}$$

la multiderivación $\{R\}\{S_2\}$ es una reducción por familias completa que alcanza la forma normal. Sin embargo, no es optimal, dado que $\{S_1\}$ es una reducción por familias más corta que también alcanza la forma normal.

Para definir formalmente qué significa que una multiderivación efectúe solamente trabajo computacional necesario, Lévy trabaja con la siguiente definición: un paso $R : t \to s$ es *necesario* si toda derivación coinicial $\sigma : t \twoheadrightarrow u$ que alcanza la forma normal de t contrae al menos un residuo de R. Una reducción por familias $\mathcal{M}_1 \dots \mathcal{M}_n$ se dice *necesaria* si todo multipaso $\mathcal{M}_i$ contiene al menos un paso necesario. El resultado de optimalidad de Lévy afirma entonces lo siguiente:

Teorema 6.17 (Optimalidad — Lévy, 1978)**.** *En el cálculo-λ, cualquier reducción por familias completa y necesaria que alcanza una forma normal es optimal.*

Optimalidad en Deterministic Family Structures

En [19], Glauert y Khasidashvili proponen una generalización del resultado de optimalidad de Lévy. Este resultado generaliza Teo. 6.17 en dos dimensiones. En primer lugar, el resultado no sólo aplica al cálculo-λ, sino en general a cualquier Deterministic Family Structure, de las que el cálculo-λ es un caso particular. En segundo lugar, el resultado no sólo aplica a reducciones que alcanzan una forma normal, sino más en general a reducciones que alcanzan una *respuesta*, donde la noción de respuesta es un parámetro adicional para el teorema de optimalidad generalizado. La noción de respuesta se especifica a través de un conjunto de términos que pueden variar en distintos contextos. Por ejemplo, en el cálculo-λ cualquiera de los siguientes conjuntos de formas normales (f.n.'s) se puede considerar como un conjunto de respuestas:

$$\begin{array}{rl}
\{t \in \mathcal{T} \mid \nexists s \in \mathcal{T}.\, t \to s\} & \text{(f.n.'s)} \\
\{\lambda x.t \mid x \in \mathcal{V},\ t \in \mathcal{T}\} & \text{(abstracciones)} \\
\{\lambda x_1 \dots x_n.y\, t_1 \dots t_m \mid n, m \geqslant 0,\ x_1, \dots, x_n, y \in \mathcal{V},\ t_1, \dots, t_m \in \mathcal{T}\} & \text{(f.n.'s a la cabeza)} \\
\{\lambda x.t \mid x \in \mathcal{V}, t \in \mathcal{T}\} \cup \{x t_1 \dots t_n \mid n \geqslant 0,\ x \in \mathcal{V},\ t_1, \dots, t_n \in \mathcal{T}\} & \text{(f.n.'s a la cabeza débiles)}
\end{array}$$

donde $\mathcal{V}$ es el conjunto de variables y $\mathcal{T}$ el conjunto de todos los términos. El conjunto de respuestas se escribe $\mathcal{X}$. Las definiciones y resultados de esta subsección se remontan al trabajo de Lévy y corresponden a la generalización hecha por Glauert y Khasidashvili para DFSs arbitrarias [19, 7].

Definición 6.18 ($\mathcal{X}$-necesario)**.** Sea $\mathcal{A}$ un sistema de reescritura axiomático ortogonal y sea $\mathcal{X}$ un conjnuto de objetos. Un paso $R : x \to y$ es *$\mathcal{X}$-necesario* si toda derivación $\sigma : x \twoheadrightarrow z \in \mathcal{X}$ contrae al menos un residuo de R. Un multipaso $\mathcal{M}$ es $\mathcal{X}$-necesario si contiene al menos un paso $\mathcal{X}$-necesario. Una multiderivación $\mathcal{M}_1 \dots \mathcal{M}_n$ es $\mathcal{X}$-necesaria si el multipaso $\mathcal{M}_i$ es $\mathcal{X}$-necesario para todo $i \in 1..n$.

Por motivos técnicos, el conjunto de respuestas no puede ser un conjunto arbitrario de objetos. Debe ser un conjunto estable:

Definición 6.19 (Conjunto estable)**.** Un conjunto $\mathcal{X}$ de objetos es *estable* si:

1. $\mathcal{X}$ es cerrado por *parallel moves*, es decir, para todo $x \notin \mathcal{X}$, para todo $\rho : x \twoheadrightarrow y \in \mathcal{X}$, y para toda reducción $\sigma : x \twoheadrightarrow z$ que no contiene objetos en $\mathcal{X}$, el destino de ρ/σ está en $\mathcal{X}$.

2. $\mathcal{X}$ es cerrado bajo expansión no necesaria, es decir, para todo $R : x \to y$ tal que $x \notin \mathcal{X}$ y $y \in \mathcal{X}$, el paso R es $\mathcal{X}$-necesario.

Ejemplo 6.20. *Es fácil verificar que, en el cálculo-λ, el conjunto de abstracciones $\{\lambda x.t \mid t \in \mathcal{T}\}$ es estable.*

Definición 6.21 (Reducción $\mathcal{X}$-optimal)**.** Sea $x \in \mathcal{A}$ un objeto en un DFS y sea $\mathcal{X}$ un conjunto estable en $\mathcal{A}$. Una reducción por familias $D : x \twoheadrightarrow y \in \mathcal{X}$ es *$\mathcal{X}$-optimal* si su longitud es mínima entre todas las reducciones por familias de la forma $x \twoheadrightarrow y \in \mathcal{X}$ (donde x está fijo y y varía).

Notemos $\mathsf{FAM}(D)$ para el conjunto de familias de una multiderivación. Más precisamente:

$$\mathsf{FAM}(\mathcal{M}_1 \dots \mathcal{M}_n) \stackrel{\text{def}}{=} \{\mathsf{Fam}_{\simeq}(\mathcal{M}_1 \dots \mathcal{M}_{i-1} R) \mid 1 \leqslant i \leqslant n,\ R \in \mathcal{M}_i\}$$

Entonces podemos demostrar el siguiente resultado auxiliar.

Lema 6.22. *Sea $\mathcal{X}$ un conjunto estable de términos en una DFS. Si $D : x \twoheadrightarrow y \in \mathcal{X}$ es una reducción por familias, entonces $\#\mathsf{FAM}(D) \leqslant |D|$.*

Lema 6.23. *Sea $\mathcal{X}$ un conjunto estable de términos en una DFS. Si $D : x \twoheadrightarrow y \in \mathcal{X}$ es una reducción por familias completa y $\mathcal{X}$-necesaria, entonces $|D| = \#\mathsf{FAM}(D)$.*

Teorema 6.24 (Optimalidad generaliza — Glauert y Khasidashvili, 1996)**.** *Sea $\mathcal{X}$ un conjunto estable de términos en un DFS. Entonces cualquier reducción por familias completa y $\mathcal{X}$-necesaria $D : x \twoheadrightarrow y \in \mathcal{X}$ es $\mathcal{X}$-optimal.*

Ejemplo 6.25 (Reducción optimal en el cálculo-λ)**.** *Sea Δ un término cualquiera tal que $\Delta \to \Delta'$, y considerar el siguiente diagrama:*

$$\begin{tikzcd}
 & & (\lambda x.xx)\,((\lambda x.y)\,\Delta) \arrow[dl, "R"'] \arrow[dr, "S"] \arrow[rr, "T"] & & (\lambda x.xx)\,((\lambda x.y)\,\Delta') \\
 & (\lambda x.y)\,\Delta\,((\lambda x.y)\,\Delta) \arrow[dl, "S_1"'] \arrow[dr, "S_2"] & & (\lambda x.xx)\,y \arrow[ddll, bend left, "R'"] & \\
y\,((\lambda x.y)\,\Delta) \arrow[dr, "S'_2"'] & & (\lambda x.y)\,\Delta\,y \arrow[dl, "S'_1"] & & \\
 & y\,y & & &
\end{tikzcd}$$

Entonces las reducciones por familias $\{R\}\{S_1, S_2\}$ y $\{S\}\{R'\}$ son ambas reducciones optimales a forma normal. Las reducciones por familias $\{R\}\{S_1\}\{S'_2\}$ y $\{R\}\{S_2\}\{S'_1\}$ no son completas. Cualquier reducción por familias que comience con el multipaso $\{T\}$ no puede ser necesaria, porque el paso T no es necesario para alcanzar la forma normal.

Optimalidad en el LSC sin gc

Combinando el hecho de que el LSC sin gc es una Deterministic Family Structure (Teo. 6.13) con el resultado de optimalidad generalizado para DFSs (Teo. 6.24), se obtiene un resultado de optimalidad para el LSC. Sin embargo, el resultado de optimalidad generalizado depende de la elección de un conjunto estable $\mathcal{X}$ que capture la noción de respuesta buscada.

Uno podría estar interesado en el conjunto de respuestas dado por las formas normales del LSC sin gc, es decir, en el conjunto:

$$\mathtt{NF}_{\mathtt{db,ls}} \stackrel{\text{def}}{=} \{t \in \mathcal{T} \mid \nexists s \in \mathcal{T}.\ t \to_{\mathtt{db,ls}} s\}$$

es fácil comprobar que $\mathtt{NF}_{\mathtt{db,ls}}$ es un conjunto estable. Sin embargo, la noción de $\mathtt{NF}_{\mathtt{db,ls}}$-optimalidad que uno obtiene en este caso no es muy interesante, por dos razones. Un motivo es que en el LSC sin gc no hay borrado, lo que implica que todos los pasos son siempre $\mathtt{NF}_{\mathtt{db,ls}}$-necesarios. Otro motivo es que en el LSC sin gc uno no busca obtener realmente la forma normal de un término. Por ejemplo, sea $\Omega = (\lambda x.xx)\,\lambda x.xx$ y considerar la siguiente derivación:

$$\begin{array}{lll}
(\lambda x.\lambda y.x)\,z\,\Omega & \to_{\mathtt{db}} & (\lambda y.x)[x\backslash z]\,\Omega \\
 & \to_{\mathtt{db}} & x[y\backslash\Omega][x\backslash z] \\
 & \to_{\mathtt{ls}} & z[y\backslash\Omega][x\backslash z] \\
 & \to_{\mathtt{db}} & z[y\backslash(xx)[x\backslash\lambda x.xx]][x\backslash z] \\
 & \to & \ldots
\end{array}$$

en este ejemplo, la reducción sigue indefinidamente sin alcanzar nunca una forma normal, prosiguiendo con la evaluación del término dentro de la sustitución $[y\backslash\ldots]$, a pesar de que esta sustitución no se utiliza. La noción interesante es la de formas normales *salvo aplicación de la regla de garbage collection* que elimina las sustituciones que no se utilizan. Esta es la noción de *forma normal alcanzable* que se define a continuación.

Definición 6.26 (Forma normal alcanzable)**.** Notemos $\mathsf{nf}_{\mathtt{gc}}(t)$ a la gc-forma normal de un término t dado. El conjunto RNF de *formas normales alcanzables* es el conjunto de términos:

$$\mathsf{RNF} \stackrel{\text{def}}{=} \{t \in \mathcal{T} \mid \mathsf{nf}_{\mathtt{gc}}(t) \in \mathtt{NF}_{\mathtt{db,ls}}\}$$

La siguiente proposición justifica que el Teo. 6.24 se puede aplicar a la noción de reducción RNF-optimal.

Proposición 6.27 (El conjunto RNF es estable)**.**

Ejemplo 6.28 (Optimal RNF-reduction in the LSC without gc)**.** *Sea Δ un término arbitrario tal que $\Delta \to \Delta'$ y considerar el siguiente diagrama, en el cual los términos en* RNF *se encuentran*

subrayados:

$$
\begin{tikzcd}
 & & x[x\backslash(\lambda y.z)\,\Delta] \arrow[dl, "R"'] \arrow[r, "T"] \arrow[dr, "S"] & x[x\backslash(\lambda y.z)\,\Delta'] & \\
 & ((\lambda y.z)\,\Delta)[x\backslash(\lambda y.z)\,\Delta] \arrow[dl, "S_1"'] \arrow[dr, "S_2"] & & & x[x\backslash y[z\backslash\Delta]] \arrow[dddll, bend left] \\
\underline{y[z\backslash\Delta][x\backslash(\lambda y.z)\,\Delta]} \arrow[dr, "S_2'"'] & & ((\lambda y.z)\,\Delta)[x\backslash z[y\backslash\Delta]] \arrow[dl, "S_1'"] & & \\
 & \underline{y[z\backslash\Delta][x\backslash y[z\backslash\Delta]]} & & &
\end{tikzcd}
$$

Entonces las reducciones por familias $\{R\}\{S_1, S_2\}$ *y* $\{S\}\{R'\}$ *son* RNF*-optimales por Teo. 6.24. Cualquier reducción por familias que comience con el multipaso* $\{T\}\ldots$ *no es* RNF*-necesaria, porque* T *no es no es necesario para alcanzar un término en* RNF*.*

Observar que la reducción por familias $\{R\}\{S_1\}$ *alcanza un término en* RNF *en el menor número posible de multipasos, pero no es completa porque* $\{S_1\}$ *no es maximal, de modo que el Teo. 6.24 no asegura que sea* RNF*-optimal.*

6.5. Estandarización

El problema de *estandardización* consiste, en general, en hallar, para cada derivación $\rho : x \twoheadrightarrow y$ una derivación $\rho' : x' \twoheadrightarrow y'$ *estándar* y *equivalente* a ρ.

En principio uno podría estar interesado en varias nociones distintas de *equivalencia* entre derivaciones. En la literatura, la palabra *estandarización* se refiere comúnmente a la estandarización con respecto a la relación de *equivalencia por permutación.*

Dada una noción de equivalencia fija $\sim$ entre derivaciones, a veces es posible demostrar un resultado de estandarización, que involucra a la clase de derivaciones S, cuyos elementos se denominan *derivaciones estándar.* Un resultado de estandarización afirma que siempre es posible hallar, para cada derivación ρ, una derivación estándar equivalente $\rho' \in S$. Un resultado de estandarización más fuerte aseguraría además que para cada derivación ρ hay una *única* derivación equivalente $\rho' \in S$, es decir, que el conjunto de clases de equivalencia módulo $\sim$ está en correspondencia 1–1 con el conjunto S. Además, el resultado de estandarización se demuestra constructivamente, dando un procedimiento que otorga, para cada derivación ρ el representante estándar ρ' de su clase de $\sim$-equivalencia.

En esta sección definimos un procedimiento de estandarización para Deterministic Family Structures, requiriendo algunos axiomas adicionales. La demostración de que el procedimiento de estandarización termina se apoya en la propiedad de Finite Family Developments. Como corolario, obtenemos un teorema de estandarización para el LSC sin gc.

Se han formulado muchos teoremas de estandarización abstracta. El resultado que presentamos en esta sección es una adaptación del teorema de estandarización en paralelo de Klop ([37, Proposition 8.5.19]) al marco de las Deterministic Family Structures.

Observar que en [3, Theorem 3, Theorem 4], Accattoli, Bonelli, Lombardi y Kesner ya han

propuesto un procedimiento de estandarización para el LSC. Nuestro procedimiento difiere de aquél en los siguientes aspectos:

1. Nuestro procedimiento se basa en el teorema de Finite Family Developments, mientras que [3] se basa en el hecho de que el LSC cumple con una serie de axiomas propuestos por Melliès en [34, Chapter 4].

2. Nuestro procedimiento de estandarización está inspirado por el de Klop [37, Section 8.5.2], y se basa en un procedimiento de *selection*, que remite al algoritmo de ordenamiento por selección, mientras que [3] se basa en un procedimiento de *permutación* de pares anti–standard, que remite al algoritmo de ordenamiento por burbujeo.

3. Nuestro procedimiento no trata la regla gc, mientras [3] sí la trata.

4. Nuestro procedimiento impone un orden fijo entre los redexes de modo tal que la reducción estándar es sintácticamente única, en tanto que [3] considera las formas estándar módulo permutación de redexes disjuntos, de modo tal que la reducción estándar es única *salvo equivalencia por permutación de cuadrados.*

Estandarización en Deterministic Family Structures

En esta subsección demostramos un resultado de estandarización para Deterministic Family Structures que verifican algunas restricciones adicionales. El resultado principal de esta sección es el resultado de estandarización para DFSs (Prop. 6.39). Empezamos demostrando un resultado técnico sencillo.

Proposición 6.29 (La proyección no crea familias)**.** *Sea $\mathcal{A}$ una DFS, sea $\phi : t \twoheadrightarrow t'$ una derivación en $\mathcal{A}$, y sean ρ y σ derivaciones coiniciales en $\mathcal{A}$ comenzando desde un término t'. Entonces el conjunto de familias de redexes que se contraen a lo largo de ρ/σ está incluido en el conjunto de familias que se contraen a lo largo de ρ, relativos a la historia ϕ. Más precisamente, si ρ/σ se escribe como $\tau_1 T \tau_2$ entonces ρ se puede escribir como $\upsilon_1 U \upsilon_2$ de tal modo que $\mathsf{Fam}_{\simeq}(\phi\upsilon_1 U) = \mathsf{Fam}_{\simeq}(\phi\sigma\tau_1 T)$.*

Para demostrar el resultado de estandarización, enunciamos algunas definiciones auxiliares, incluyendo la noción clave de *estrategia de multi-selección uniforme*. Recordar que en un sistema de reescritura axiomático ortogonal las letras $\mathcal{M}, \mathcal{N}, \ldots$ denotan multipasos, $D, E, \ldots$ denotan multiderivaciones y Multistep denota el conjunto de todos los multipasos.

Definición 6.30 (Pertenencia)**.** En un sistema de reescritura axiomático ortogonal $\mathcal{A}$, un paso R *pertenece* a una derivación ρ, lo que se escribe $R \lhd \rho$, si y sólo si ρ se puede escribir como de la forma $\rho = \rho_1 R' \rho_2$ donde $R' \in R/\rho_1$. Un multipaso $\mathcal{M}$ *pertenece* a una derivación ρ, lo que se escribe $\mathcal{M} \lhd \rho$, si y sólo si $R \lhd \rho$ para cada $R \in \mathcal{M}$.

Definición 6.31 (Estrategia de multi-selección)**.** En un sistema de reescritura axiomático ortogonal $\mathcal{A}$, una *estrategia de multi-selección* es una función $\mathbb{M}$ que le asocia, a cada derivación ρ no vacía, un multipaso $\mathcal{M} \in$ Multistep coinicial tal que $\mathcal{M} \lhd \rho$ y $\mathcal{M}/\rho = \varnothing$.

Definición 6.32 (Estrategia de multi-selección uniforme)**.** Una estrategia de multi-selección $\mathbb{M}$ se dice *uniforme* si $\rho \equiv \sigma$ implica $\mathbb{M}(\rho) = \mathbb{M}(\sigma)$ para derivaciones ρ, σ no vacías cualesquiera.

Ejemplo 6.33. *En el cálculo-λ, considerar la estrategia de multi-selección trivial $\mathbb{M}_{\text{Triv}}$ que siempre elige el primer paso de una derivación dada. Más precisamente, sea $\mathbb{M}_{\text{Triv}}(R\rho) \stackrel{\text{def}}{=} \{R\}$. Entonces $\mathbb{M}_{\text{Triv}}$ es una estrategia de multi-selección porque para toda derivación $R\rho$ no vacía se tiene que $R \lhd R\rho$ y que $R/R\rho = \varnothing$.*

Sin embargo, $\mathbb{M}_{\text{Triv}}$ no es uniforme. Por ejemplo, si $RS' \equiv SR'$, tal como en el siguiente diagrama, se tiene que $\mathbb{M}_{\text{Triv}}(RS') = \{R\} \neq \{S\} = \mathbb{M}_{\text{Triv}}(SR')$.

$$\begin{array}{ccc} (\lambda x.(\lambda y.z)\,x\,x)\,t & \xrightarrow{R} & (\lambda y.z)\,t\,t \\ {\scriptstyle S}\downarrow & & {\scriptstyle S'}\downarrow \\ (\lambda x.z\,x)\,t & \xrightarrow{R'} & z\,t \end{array}$$

En lo que resta de esta subsección, demostramos que cualquier estrategia de multi-selección $\mathbb{M}$ induce, para una derivación ρ dada, una derivación ρ^* equivalente por permutación. Esto da lugar a un resultado de estandarización, paramétrico en la estrategia $\mathbb{M}$. El conjunto de derivaciones estándar es el conjunto $\{\rho^* \mid \rho \text{ es una derivación}\}$. Además, demostramos que la derivación ρ^* inducida es única, salvo equivalencia por permutación.

Definición 6.34 (Multiderivación inducida)**.** En un sistema de reescritura axiomático ortogonal, sea $\mathbb{M}$ una estrategia de multi-selección y sea ρ una derivación arbitraria. La *secuencia inducida por $\mathbb{M}$ sobre ρ* se escribe $\mathbb{M}^\star(\rho)$ y es una secuencia posiblemente infinita de multipasos, definida por las siguientes ecuaciones recursivas:

$$\mathbb{M}^\star(\rho) \stackrel{\text{def}}{=} \begin{cases} \epsilon & \text{si } \rho = \epsilon \\ \mathbb{M}(\rho) \cdot \mathbb{M}^\star(\rho/\mathbb{M}(\rho)) & \text{en caso contrario} \end{cases}$$

Si la recursión termina, la secuencia es finita y la llamamos la *multiderivación inducida por $\mathbb{M}$ sobre ρ*.

En un sistema de reescritura arbitrario, esta definición recursiva podría no terminar. El siguiente lema provee condiciones suficientes para que $\mathbb{M}^\star(\rho)$ quede bien definida. Concretamente, en una Deterministic Family Structure la definición recursiva de $\mathbb{M}^\star(\rho)$ está bien fundada, como consecuencia de la propiedad de Finite Family Developments.

Lema 6.35. *Sea $\mathbb{M}$ una estrategia de multi-selección en una DFS. Si ρ es cualquier derivación (finita), entonces $\mathbb{M}^\star(\rho)$ es finita.*

Por definición, una estrategia de multi-selección $\mathbb{M}$, cuando recibe dos derivaciones equivalentes por permutación, siempre elige el mismo multpaso. Y, de hecho, da lugar a la misma multiderivación.

Lema 6.36. *Sea $\mathbb{M}$ una estrategia de multi-selección uniforme en una DFS, y sean ρ, σ derivaciones finitas. Si $\rho \equiv \sigma$ entonces $\mathbb{M}^\star(\rho) = \mathbb{M}^\star(\sigma)$.*

Lema 6.37. *Sea $\mathbb{M}$ una estrategia de multi-selección en una DFS, y sea ρ una derivación finita. Entonces $\rho \equiv \partial\mathbb{M}^\star(\rho)$, donde ∂D denota el development completo canónico de D.*

Definición 6.38 (Multiderivación estándar)**.** Una multiderivación D se dice $\mathbb{M}$-*estándar* si $\mathbb{M}^\star(\partial D) = D$.

Proposición 6.39 (Estandarización para DFSs)**.** *Sea $\mathbb{M}$ una estrategia de multi-selección uniforme en una DFS. Para cualquier derivación finita ρ, existe una única multiderivación D tal que $\rho \equiv \partial D$ y D es $\mathbb{M}$-estándar. Concretamente, $D = \mathbb{M}^\star(\rho)$.*

Ejemplo 6.40 (Estandarización en el cálculo-λ)**.** *En el cálculo-λ, sea $\mathbb{M}_{\mathsf{Left}}(\rho) := \{R\}$ donde R es el paso más a la izquierda entre aquellos pasos tales que $R/\rho = \varnothing$. Sea además $\mathbb{M}_{\mathsf{Par}}(\rho) := \{R \mid R/\rho = \varnothing\}$. Se puede verificar que $\mathbb{M}_{\mathsf{Left}}$ y $\mathbb{M}_{\mathsf{Par}}$ son estrategias de multi-selección uniformes. Además, sea $\Delta \to \Delta'$ y sea ρ la derivación:*

$$\rho : (\lambda x.yxx)\,((\lambda x.z)\Delta) \to (\lambda x.yxx)\,((\lambda x.z)\Delta') \to (\lambda x.yxx)\,z \to yzz$$

Entonces la forma estándar (más a la izquierda) *de ρ es:*

$$\mathbb{M}^\star_{\mathsf{Left}}(\rho) : (\lambda x.yxx)\,((\lambda x.z)\Delta) \to y((\lambda x.z)\,\Delta)((\lambda x.z)\Delta) \to yz((\lambda x.z)\Delta) \to yzz$$

La forma estándar paralela *de ρ consta de un único multipaso:*

$$\mathbb{M}^\star_{\mathsf{Par}}(\rho) : (\lambda x.yxx)\,((\lambda x.z)\Delta) \Rightarrow yzz$$

Estandarización en el LSC sin gc

En esta subsección aplicamos el resultado de estandarización anterior (Prop. 6.39) al LSC sin gc.

Definición 6.41 (Selector arbitrario)**.** Sea $\mathsf{Out}(t)$ el conjunto de pasos con origen en un término t del LSC sin gc, y sea $<_t$ un orden parcial estricto sobre $\mathsf{Out}(t)$. Escribimos $<$ para la función tal que, dado un término $t \in \mathcal{T}$, otorga un orden parcial $<_t \subseteq \mathsf{Out}(t) \times \mathsf{Out}(t)$.

El *selector arbitrario sobre* $<$ se escribe $\mathbb{M}_<$ y se define como la siguiente función, que recibe una derivación no vacía y devuelve un conjunto finito de pasos coiniciales:

$$\mathbb{M}_<(\rho) \stackrel{\text{def}}{=} \{R \mid R/\rho = \varnothing \text{ y } R \text{ es minimal}\}$$

Por *minimal* nos referimos a que no existe un paso R' tal que $R'/\rho = \varnothing$ y $R' <_{\mathsf{src}(\rho)} R$.

Observar que $\mathbb{M}_<(\rho)$ es un conjunto finito no vacío. Para ver esto, notar que el conjunto $X = \{R \mid R/\rho = \varnothing\}$ es no vacío, pues $R/\rho = \varnothing$ si R es el primer paso de la derivación ρ. Además, el conjunto X es finito, porque el LSC es *finitely branching*. Por lo tanto X debe contener al menos un elemento minimal. Además:

Lema 6.42. *$\mathbb{M}_<$ es una estrategia de multi-selección uniforme.*

Corolario 6.43 (Estandarización por selección arbitraria para el LSC sin gc)**.** *Sea $\mathbb{M}_<$ el selector arbitrario sobre $<$. Para cada secuencia finita ρ en el LSC sin* gc*, hay una única multiderivación D tal que $\rho \equiv \partial D$ y D es $\mathbb{M}_<$-estándar. Además, si la función $<$ es computable, entonces D es computable a partir ρ, a saber: $D = \mathbb{M}^\star_<(\rho)$.*

Ejemplo 6.44 (Estandarización en el LSC sin gc)**.** *En el LSC sin* gc*, sea $\rho : x[x\backslash t] \to x[x\backslash t'] \to t'[x\backslash t'] \to t''[x\backslash t']$, donde $t \to t' \to t''$.*

1. *Si $<^1$ es el orden parcial trivial en el que todo par de pasos es incomparable, es decir, $R <^1_t S$ nunca se verifica, entonces: $\mathbb{M}^\star_{<^1}(\rho) : x[x\backslash t] \Rightarrow t'[x\backslash t'] \to t''[x\backslash t']$. El primer paso es un multipaso propiamente dicho.*

2. *Sea $<^2$ el orden total de izquierda a derecha, definido de tal modo que $R <^2_t S$ se verifica cuando R está a la izquierda de S. Entonces: $\mathbb{M}^\star_{<^2}(\rho) : x[x\backslash t] \to t[x\backslash t] \to t'[x\backslash t] \to t'[x\backslash t'] \to t''[x\backslash t']$.*

3. *Sea $<^3$ es el orden total de derecha a izquierda, definido de tal modo que $R <^3_t S$ se verifica si R está a la derecha de S. Entonces: $\mathbb{M}^\star_{<^3}(\rho) = \rho : x[x\backslash t] \to x[x\backslash t'] \to t'[x\backslash t'] \to t''[x\backslash t']$.*

6.6. Normalización de estrategias

Una *estrategia de reducción* es, informalmente, una restricción sobre los pasos de cómputo que se pueden ejecutar en un sistema de reescritura. Por ejemplo, en el cálculo-λ, la *reducción a la cabeza* es la restricción de la regla de β-reducción que solamente permite contraer *redexes a la cabeza*, esto es, redexes que están bajo un contexto de la forma $\lambda x_1 \ldots x_n.\Box\, u_1 \ldots u_m$. Más precisamente, la reducción a la cabeza se define por medio de la siguiente regla de reescritura:

$$\lambda x_1 \ldots x_n.(\lambda y.t)s\, u_1 \ldots u_m \to_{\mathsf{head}} \lambda x_1 \ldots x_n.t\{y := s\}\, u_1 \ldots u_m$$

Por ejemplo, la siguiente es una secuencia de pasos de reducción a la cabeza. Subrayamos el redex que se contrae:

$$\lambda x.\underline{(\lambda y.y)((\lambda y.x)\Omega)} \to_{\mathsf{head}} \lambda x.\underline{(\lambda y.x)\Omega} \to_{\mathsf{head}} \lambda x.x$$

en tanto que la siguiente no es una secuencia de pasos de reducción a la cabeza, porque el primer paso no contrae un redex a la cabeza:

$$\lambda x.(\lambda y.y)(\underline{(\lambda y.x)\Omega}) \to \lambda x.\underline{(\lambda y.y)x} \to_{\mathsf{head}} \lambda x.x$$

Se puede demostrar que un término tiene a lo sumo un redex a la cabeza. Un término sin redex a la cabeza es una *forma normal a la cabeza*. Es un hecho conocido el de que, al contraer sucesivamente el redex a la cabeza, se alcanza la forma normal a la cabeza, de ser posible. Más precisamente, se tiene el siguiente resultado—ver por ejemplo [36, Corollary 1.5.12 (i)] para su demostración:

Proposición 6.45 (La reducción a la cabeza es normalizante a la cabeza en el cálculo-λ)**.** *Sea t un término que tiene forma normal a la cabeza, es decir, existe una forma normal a la cabeza s tal que $t \leftrightarrow_\beta^* s$. Entonces no existe una reducción a la cabeza infinita $t \to_{\mathsf{head}} t_1 \to_{\mathsf{head}} t_2 \dots$.*

Observar que un término, tal como $\Omega = (\lambda x.xx)(\lambda x.xx)$, podría no tener una forma normal a la cabeza, en cuyo caso es imposible que una estrategia alcance una forma normal a la cabeza.

El resultado de Prop. 6.45 se conoce como el hecho de que la reducción a la cabeza es *normalizante a la cabeza*. En general, si $\mathcal{X}$ es un conjunto de respuestas, una estrategia se dice $\mathcal{X}$-normalizante si contraer sucesivamente un término de acuerdo con lo indicado por la estrategia lleva a un término en el conjunto $\mathcal{X}$, siempre que sea posible.

En esta sección damos condiciones suficientes bajo las cuales ciertas estrategias de reducción son $\mathcal{X}$-normalizantes en Deterministic Family Structures. La demostración de normalización se basa fuertemente en la propiedad de Finite Family Developments. Como consecuencia, tenemos que dos estrategias específicas, *call-by-name* y *call-by-need lineal*, son normalizantes en el LSC sin gc.

Se han estudiado muchos resultados de normalización. En particular, mencionamos que Glauert y Khasidashvili demuestran un resultado de normalización relativa [19, Theorem 4.1] para Deterministic Family Structures, que asegura que contraer pasos $\mathcal{X}$-necesarios (*cf.* Def. 6.18) alcanza un término en $\mathcal{X}$ de esr posible. El resultado de normalización para DFSs que enunciamos y demostramos es un caso particular, es decir, es más débil que el resultado de normalización relativa de Glauert y Khasidashvili. La ventaja es que nuestro resultado sólo requiere verificar un número de condiciones sintácticas locales sobre diagramas de reescritura para poder asegurar que una estrategia es $\mathcal{X}$-normalizante.

Normalización en Deterministic Family Structures

En esta subsección demostramos un resultado de normalización para Deterministic Family Structures. El resultado principal de esta subsección es Prop. 6.54, en el que damos condiciones suficientes para que una estrategia sea $\mathcal{X}$-normalizante. Comenzamos dando definiciones formales de todas las nociones asociadas.

Definición 6.46 (Sub-ARS)**.** Un *sub-ARS* de un ARS $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ es un ARS $\mathcal{B} = (\mathsf{Obj}', \mathsf{Stp}', \mathsf{src}', \mathsf{tgt}')$ tal que $\mathsf{Obj}' \subseteq \mathsf{Obj}$, $\mathsf{Stp}' \subseteq \mathsf{Stp}$, y además las funciones $\mathsf{src}', \mathsf{tgt}'$ son las restricciones de $\mathsf{src}, \mathsf{tgt}$ a Stp'. Un sub-ARS $\mathcal{B}$ es *cerrado* si el conjunto $\mathsf{NF}(\mathcal{B})$ es cerrado por reducción, es decir si $x \to_{\mathcal{A}} y$ y $x \in \mathsf{NF}(\mathcal{B})$ entonces $y \in \mathsf{NF}(\mathcal{B})$.

Definición 6.47 (Estrategia)**.** Una *estrategia* en un ARS $\mathcal{A} = (\mathsf{Obj}, \mathsf{Stp}, \mathsf{src}, \mathsf{tgt})$ es un sub-ARS $\mathcal{B} = (\mathsf{Obj}', \mathsf{Stp}', \mathsf{src}', \mathsf{tgt}')$ que cuenta con los mismos objetos, es decir, $\mathsf{Obj} = \mathsf{Obj}'$, y el mismo conjunto de formas normales, es decir, $\mathsf{NF}(\mathcal{A}) = \mathsf{NF}(\mathcal{B})$.

Observación 6.48. Cualquier sub-ARS $\mathcal{B}$ se puede extender a una estrategia $\mathbb{S}_{\mathcal{B}}$ agregando los pasos que salen de las formas normales, es decir, declarando $\mathsf{Stp}(\mathbb{S}_{\mathcal{B}}) := \mathsf{Stp}(\mathcal{B}) \cup \{R \in \mathsf{Stp}(\mathcal{A}) \mid \mathsf{src}(R) \in \mathsf{NF}(\mathcal{B})\}$. Observar en particular que si $\mathcal{B}$ es una estrategia entonces $\mathbb{S}_{\mathcal{B}} = \mathcal{B}$.

Ejemplo 6.49. *En el cálculo-λ, la noción de reducción a la cabeza $\to_{\mathsf{head}}$ no es una estrategia estrictamente hablando, porque el conjunto de β-formas normales no coincide con el conjunto de formas normales a la cabeza.*

La reducción a la cabeza se puede extender a una estrategia $\mathbb{S}_{\mathsf{head}}$ de tal modo que un paso β arbitrario $R : t \to_\beta s$ esté en la estrategia $\mathbb{S}_{\mathsf{head}}$ si R contrae un redex a la cabeza o, alternativamente, t es una forma normal a la cabeza.

Definición 6.50 (Estrategia $\mathcal{X}$-normalizante)**.** Sea $\mathcal{X}$ un superconjunto de las formas normales de $\mathcal{A}$. Una estrategia $\mathbb{S}$ se dice *$\mathcal{X}$-normalizante* si para todo objeto x tal que existe una reducción $x \twoheadrightarrow_{\mathcal{A}} y \in \mathcal{X}$, toda reducción maximal desde x en la estrategia $\mathbb{S}$ contiene un objeto en $\mathcal{X}$.

La siguiente noción de *invarianza por residuos* es la noción clave para dar una condición suficiente para que una estrategia sea $\mathcal{X}$-normalizante.

Definición 6.51 (Invarianza por residuos)**.** Sea $\mathcal{A}$ un sistema de reescritura axiomático. Un sub-ARS $\mathcal{B}$ de $\mathcal{A}$ es *invariante por residuos* si dados pasos R y S tales que $R \in \mathcal{B}$ y $S \neq R$, existe un paso $R' \in \mathbb{S}$ tal que $R' \in R/S$.

Ejemplo 6.52. *En el cálculo-λ, la estrategia* leftmost outermost $\mathbb{S}_{\mathsf{LO}}$ *es la estrategia que sólo permite contraer el paso* leftmost outermost*, es decir, el paso que contrae el redex cuya λ está más a la izquierda. Es fácil verificar que $\mathbb{S}_{\mathsf{LO}}$ es invariante por residuos, porque el residuo de un paso leftmost outermost es también leftmost outermost.*

El siguiente es un lema sencillo sobre la noción de invarianza por residuos:

Lema 6.53 (Los pasos de un sub-ARS invariante por residuos se preservan en DFSs)**.** *Sea $F = (\mathcal{A}, \simeq, \hookrightarrow)$ una DFS y sea $\mathcal{B}$ un sub-ARS invariante por residuos de $\mathcal{A}$. Sea ρR un redex con historia tal que R está en $\mathcal{B}$, y sea σ cualquier reducción finita coinicial al paso R. Supongamos también que σ no contrae redexes en la familia de ρR. Más precisamente, supongamos que cada vez que σ se puede escribir como $\sigma_1 S \sigma_2$ se tiene que $\rho R \not\simeq \rho\sigma_1 S$. Entonces R tiene un residuo $R' \in R/\sigma$ en $\mathcal{B}$.*

Pasamos al resultado principal de esta subsección:

Proposición 6.54 (Normalización para DFSs)**.** *Sea $\mathcal{B}$ un sub-ARS cerrado e invariante por residuos en una Deterministic Family Structure. Entonces la estrategia $\mathbb{S}_{\mathcal{B}}$ es $\mathsf{NF}(\mathcal{B})$-normalizante.*

Normalización en el LSC sin gc

En las subsecciones que signe, definimos dos estrategias en el LSC sin gc y demostramos que son normalizantes.

En primer lugar, estudiamos la noción de *reducción lineal a la cabeza*, que corresponde a la reducción *call-by-name* en el LSC sin gc. Como vimos en el Capítulo 2, esta estrategia se corresponde íntimamente con la evaluación de λ-términos en la máquina abstracta de Krivine. Además, esta estrategia se corresponde también con la noción de reducción lineal a la cabeza de Vincent Danos y Laurent Regnier en el cálculo-λ [13].

En segundo lugar, definimos una nueva estrategia que denominamos *reducción call-by-need lineal*. La reducción call-by-need lineal es similar a la estrategia call-by-need (débil) que estudiamos en los Capítulos 2 y 3, con la diferencia ligera diferencia de que la regla de sustitución lineal que estudiamos tiene la forma (6.1) en lugar de la forma (6.2).

$$\mathtt{C}\langle x\rangle[x\backslash\mathtt{vL}] \to \mathtt{C}\langle \mathtt{vL}\rangle[x\backslash\mathtt{vL}] \tag{6.1}$$

$$\mathtt{C}\langle x\rangle[x\backslash\mathtt{vL}] \to \mathtt{C}\langle \mathtt{v}\rangle[x\backslash\mathtt{v}]\mathtt{L} \tag{6.2}$$

La ventaja de la estrategia call-by-need débil, dada por (6.2), es que sólo copia el subtérmino v, y comparte el contexto de sustitución L. Además, (6.2) se corresponde con la noción de call-by-need de Ariola *et al.* [5, 33]. Desafortunadamente, la estrategia call-by-need no es un sub-ARS del LSC, de modo que no sería posible aplicar nuestros resultados directamente a la variante de call-by-need débil dada por (6.2) sin antes rehacer parte del trabajo que hemos realizado en las secciones precedentes. Por ejemplo, habría que demostrar que una variante del LSC adaptada con la nueva regla de sustitución lineal también forma una Deterministic Family Structure. Dejamos esto pendiente como trabajo futuro y restringimos nuestra atención solamente a la variante de call-by-need dada por (6.1).

Normalización de la reducción lineal a la cabeza

En esta subsección, recordamos la definición de la reducción lineal a la cabeza (call-by-name) y demostramos que es normalizante.

Definición 6.55 (Reducción lineal a la cabeza y formas normales lineales a la cabeza)**.** La reducción lineal a la cabeza es el sub-ARS $\mathbb{HL}$ del LSC sin gc que selecciona el (único) paso db o ls cuya ancla se encuentre bajo un contexto a la cabeza débil. Los contextos a la cabeza débiles están definidos por la gramática:

$$H ::= \Box \mid H\,t \mid H[x\backslash t]$$

El conjunto de formas normales lineales a la cabeza HLNF se define como el conjunto de términos generados por la gramática:

$$\begin{array}{rcl} A & ::= & (\lambda x.t)\mathtt{L} \\ & \mid & H\langle\!\langle x\rangle\!\rangle \quad \text{donde } H \text{ no liga } x \end{array}$$

Los términos de la forma $(\lambda x.t)\mathtt{L}$ se llaman *respuestas*, en tanto que los términos de la forma $H\langle\!\langle x\rangle\!\rangle$ se llaman *estructuras*. La variable x se llama la the *variable a la cabeza* de una estructura $H\langle\!\langle x\rangle\!\rangle$.

Corolario 6.56 (La reducción lineal a la cabeza es HLNF-normalizante)**.** *La estrategia* $\mathbb{S}_{HL}$ *asociada al sub-ARS* $\mathbb{HL}$ *es* HLNF*-normalizante.*

Ejemplo 6.57. *La siguiente es una reducción lineal a la cabeza que alcanza un término en* HLNF.

$$\begin{array}{rcl}
(\lambda x.xx)((\lambda y.y)(\lambda z.z)) & \to & (xx)[x\backslash(\lambda y.y)(\lambda z.z)] \\
& \to & ((\lambda y.y)(\lambda z.z)x)[x\backslash(\lambda y.y)(\lambda z.z)] \\
& \to & (y[y\backslash\lambda z.z]x)[x\backslash(\lambda y.y)(\lambda z.z)] \\
& \to & ((\lambda z.z)[y\backslash\lambda z.z]x)[x\backslash(\lambda y.y)(\lambda z.z)] \\
& \to & z[z\backslash x][y\backslash\lambda z.z][x\backslash(\lambda y.y)(\lambda z.z)] \\
& \to & x[z\backslash x][y\backslash\lambda z.z][x\backslash(\lambda y.y)(\lambda z.z)] \\
& \to & ((\lambda y.y)(\lambda z.z))[z\backslash x][y\backslash\lambda z.z][x\backslash(\lambda y.y)(\lambda z.z)] \\
& \to & y[y\backslash\lambda z.z][z\backslash x][y\backslash\lambda z.z][x\backslash(\lambda y.y)(\lambda z.z)] \\
& \to & (\lambda z.z)[y\backslash\lambda z.z][z\backslash x][y\backslash\lambda z.z][x\backslash(\lambda y.y)(\lambda z.z)]
\end{array}$$

Normalización de la reducción call-by-need lineal

En esta subsección, definimos una variante de call-by-need que denominamos *reducción call-by-need lineal* y demostramos que es normalizante.

Definición 6.58 (Reducción call-by-need lineal y formas normales call-by-need lineales)**.** La reducción call-by-need lineal es el sub-ARS $\mathbb{NL}$ del LSC sin gc definido como sigue. Los contextos de evaluación call-by-need lineales están dados por la gramática:

$$\mathtt{N} ::= \Box \mid \mathtt{N}\, t \mid \mathtt{N}[x\backslash t] \mid \mathtt{N}\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{N}]$$

La regla de reducción $\to_{\mathbb{NL}}$ es una unión de la regla db usual y la regla lsnl:

$$\mathtt{N}\langle\!\langle x\rangle\!\rangle[x\backslash\mathtt{vL}] \mapsto_{\mathtt{lsnl}} \mathtt{N}\langle\mathtt{vL}\rangle[x\backslash\mathtt{vL}]$$

las dos reglas se clausuran por contextos call-by-need lineales.

El conjunto de *formas normales call-by-need lineales* NLNF está definida por la gramática $A ::= (\lambda x.t)\mathtt{L} \mid \mathtt{N}\langle\!\langle x\rangle\!\rangle$. Los términos de la forma $(\lambda x.t)\mathtt{L}$ se llaman *respuestas* y los términos de la forma $\mathtt{N}\langle\!\langle x\rangle\!\rangle$ se llaman *estructuras*. En las estructuras, $\mathtt{N}$ no liga x, dicha variable se llama la *variable necesaria*.

Corolario 6.59 (La reducción call-by-need lineal es NLNF-normalizante)**.** *La estrategia* $\mathbb{S}_{\mathbb{NL}}$ *asociada al sub-ARS* $\mathbb{NL}$ *es* NLNF*-normalizante.*

Ejemplo 6.60. *La siguiente es una reducción call-by-need lineal que alcanza un término en* NLNF.

$$\begin{array}{rcl}
(\lambda x.xx)((\lambda y.y)(\lambda z.z)) & \to & (xx)[x\backslash(\lambda y.y)(\lambda z.z)] \\
& \to & (xx)[x\backslash y[y\backslash\lambda z.z]] \\
& \to & (xx)[x\backslash(\lambda z.z)[y\backslash\lambda z.z]] \\
& \to & ((\lambda z.z)x)[x\backslash\lambda z.z][y\backslash\lambda z.z] \\
& \to & z[z\backslash x][x\backslash\lambda z.z][y\backslash\lambda z.z] \\
& \to & z[z\backslash\lambda z.z][x\backslash\lambda z.z][y\backslash\lambda z.z] \\
& \to & (\lambda z.z)[z\backslash\lambda z.z][x\backslash\lambda z.z][y\backslash\lambda z.z]
\end{array}$$

Capítulo 7

Conclusión

En esta tesis hemos usado cálculos con sustituciones explícitas a distancia, y en particular el Cálculo de Sustituciones Lineales, para estudiar estrategias de evaluación. Los temas principales que se abordaron son tres:

1. Demostramos que algunas de estas estrategias de evaluación—call-by-name, call-by-value, call-by-need y call-by-name fuerte—destilan el comportamiento de máquinas abstractas y que son *razonables* en términos de complejidad temporal. Esta metodología nos permitió revisitar algunas máquinas abstractas conocidas de la literatura (tales como la máquina abstracta de Krivine o la máquina ZINC de Leroy), como así también concebir nuevas máquinas abstractas.

2. Extendimos la estrategia de evaluación call-by-need al marco de la evaluación fuerte. Nuestro resultado principal es la *completitud* del call-by-need fuerte, que se basa en una técnica reciente de Kesner, utilizando sistemas de tipos intersección no idempotente para caracterizar la normalización débil.

3. Estudiamos la teoría de familias de redexes en el LSC. Para ello, propusimos una variante del LSC con etiquetas, siguiendo el trabajo de Lévy sobre el cálculo-λ. Esta teoría proporciona resultados sobre la estrategia de evaluación optimal, y otorga nuevas demostraciones de estandarización en el LSC y normalización de la estrategia call-by-need.

En las secciones siguientes se describen posibles líneas de trabajo futuro.

7.1. Una máquina abstracta para reducción call-by-need fuerte

En esta sección, proponemos una máquina abstracta para reducción call-by-need fuerte.

Definición 7.1. Los estados de la máquina son 4-uplas $\langle\ \pi \mid \varphi\ t \mid E\ \rangle$ comprendidas por: una *pila* π, una *fase* φ, un *término* t, y un *entorno* E. Los términos son los usuales del LSC. La fase es un valor lógico que puede tomar dos valores: $\Uparrow$ o $\Downarrow$. La pila representa el contexto de

evaluación actual, de manera similar a las estructuras conocidas como *zippers* [21]. Las fases, pilas y entornos se definen por medio de la siguiente gramática:

φ	$::=$	$\Uparrow$	Fase de retroceso
	$\mid$	$\Downarrow$	Fase de evaluación
π	$::=$	ϵ	Pila vacía (foco en la raíz)
	$\mid$	$\mathsf{a}(t) : \pi$	Argumento (foco a la izquierda de una aplicación)
	$\mid$	$\mathsf{d}(t) : \pi$	Estructura (foco a la derecha de una aplicación)
	$\mid$	$\mathsf{h}(E_1,\, x) : \pi$	*Heap* (foco a la derecha de una sustitución)
	$\mid$	$\lambda(x) : \pi$	Lambda (foco bajo una abstracción)
E	$::=$	ϵ	Entorno vacío
	$\mid$	$E : [x \mapsto t]$	Asociación
	$\mid$	$E : [\![x \mapsto t]\!]$	Asociación congelada
	$\mid$	$E : \mathcal{K}(x)$	Delimitador de alcance

Las reglas de transición de la máquina abstracta se dan a continuación:

$\pi \mid \Downarrow t[x\backslash s] \mid E$	$\rightsquigarrow$	$\pi \mid \Downarrow t \mid [x \mapsto s]E$			D-Migración
$\pi \mid \Downarrow ts \mid E$	$\rightsquigarrow$	$\pi\mathsf{a}(s) \mid \Downarrow t \mid E$			D-Aplicación
$\pi\mathsf{a}(s) \mid \Downarrow \lambda x.t \mid E$	$\rightsquigarrow$	$\pi \mid \Downarrow t \mid [x \mapsto s]E$			Beta
$\pi \mid \Downarrow \lambda x.t \mid E$	$\rightsquigarrow$	$\pi\lambda(x) \mid \Downarrow t \mid \mathcal{K}(x)E$			D-Lambda
		si π no termina con $\mathsf{a}(.)$ *o* $\mathsf{h}(.,\, .)$			
$\pi \mid \Downarrow x \mid E_1[x \mapsto s]E_2$	$\rightsquigarrow$	$\pi\mathsf{h}(E_1,\, x) \mid \Downarrow s \mid E_2$			Búsqueda
$\pi \mid \Downarrow x \mid E_1[\![x \mapsto s]\!]E_2$	$\rightsquigarrow$	$\pi \mid \Uparrow x \mid E_1[\![x \mapsto s]\!]E_2$			Búsqueda congelada
$\pi\mathsf{h}(E_1,\, x) \mid \Downarrow v \mid E_2$	$\rightsquigarrow$	$\pi \mid \Downarrow v' \mid E_1[x \mapsto v]E_2$			LSV
$\pi \mid \Downarrow y \mid E$	$\rightsquigarrow$	$\pi \mid \Uparrow y \mid E$			Retroceso
		si y no tiene un valor asociado en E			
$\pi\mathsf{a}(s) \mid \Uparrow t \mid E$	$\rightsquigarrow$	$\pi\mathsf{d}(t) \mid \Downarrow s \mid E$			U-Argumento
$\pi\mathsf{h}(E_1,\, x) \mid \Uparrow t \mid E_2$	$\rightsquigarrow$	$\pi \mid \Uparrow x \mid E_1[\![x \mapsto t]\!]E_2$			U-Update
$\pi\mathsf{d}(t) \mid \Uparrow s \mid E$	$\rightsquigarrow$	$\pi \mid \Uparrow ts \mid E$			U-Aplicación
$\pi\lambda(x) \mid \Uparrow t \mid [y \mapsto s]E$	$\rightsquigarrow$	$\pi\lambda(x) \mid \Uparrow t[y\backslash s] \mid E$			U-Migración
$\pi\lambda(x) \mid \Uparrow t \mid \mathcal{K}(x)E$	$\rightsquigarrow$	$\pi \mid \Uparrow \lambda x.t \mid E$			U-Lambda

Ejemplo 7.2. *The following is an execution in the strong call-by-need abstract machine. In each step we underline the focus of evaluation,* i.e. *the pattern of the* db *redex or the variable contracted*

by the `ls` *redex:*

		ϵ	$\Downarrow$	$(\lambda x.xx)(\lambda y.z(Iz)y)$		ϵ
D-Aplicación	$\rightsquigarrow$	$\mathsf{a}(\lambda y.z(Iz)y)$	$\Downarrow$	$\lambda x.xx$		ϵ
Beta	$\rightsquigarrow$	ϵ	$\Downarrow$	xx		$[x \mapsto \lambda y.z(Iz)y]$
D-Aplicación	$\rightsquigarrow$	$\mathsf{a}(x)$	$\Downarrow$	x		$[x \mapsto \lambda y.z(Iz)y]$
Búsqueda	$\rightsquigarrow$	$\mathsf{h}(\epsilon,\, x)$	$\Downarrow$	$\lambda y.z(Iz)y$		ϵ
LSV	$\rightsquigarrow$	ϵ	$\Downarrow$	$\lambda y.z(Iz)y$		$[x \mapsto \lambda y.z(Iz)y]$
D-Lambda	$\rightsquigarrow$	$\lambda(y)$	$\Downarrow$	$z(Iz)y$		$\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
D-Aplicación	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)$	$\Downarrow$	$z(Iz)$		$\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
D-Aplicación	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{a}(Iz)$	$\Downarrow$	z		$\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
Retroceso	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{a}(Iz)$	$\Uparrow$	z		$\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
U-Argumento	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)$	$\Downarrow$	Iz		$\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
D-Aplicación	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)\,\mathsf{a}(z)$	$\Downarrow$	I		$\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
Beta	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)$	$\Downarrow$	w		$[w \mapsto z]\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
Búsqueda	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)\,\mathsf{h}(\epsilon,\, w)$	$\Downarrow$	z		$\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
Retroceso	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)\,\mathsf{h}(\epsilon,\, w)$	$\Uparrow$	z		$\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
U-Update	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)\,\mathsf{d}(z)$	$\Uparrow$	w		$[\![w \mapsto z]\!]\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
U-Aplicación	$\rightsquigarrow$	$\lambda(y)\,\mathsf{a}(y)$	$\Uparrow$	zw		$[\![w \mapsto z]\!]\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
U-Argumento	$\rightsquigarrow$	$\lambda(y)\,\mathsf{d}(zw)$	$\Downarrow$	y		$[\![w \mapsto z]\!]\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
Retroceso	$\rightsquigarrow$	$\lambda(y)\,\mathsf{d}(zw)$	$\Uparrow$	y		$[\![w \mapsto z]\!]\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
U-Aplicación	$\rightsquigarrow$	$\lambda(y)$	$\Uparrow$	zwy		$[\![w \mapsto z]\!]\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
U-Migración	$\rightsquigarrow$	$\lambda(y)$	$\Uparrow$	$(zwy)[w\backslash z]$		$\mathcal{K}(y)[x \mapsto \lambda y.z(Iz)y]$
U-Lambda	$\rightsquigarrow$	ϵ	$\Uparrow$	$\lambda y.(zwy)[w\backslash z]$		$[x \mapsto \lambda y.z(Iz)y]$

Se omite la propuesta para la decodificación de los estados de la máquina como términos. Formular una noción adecuada de bisimulación fuerte $\equiv$ entre términos para demostrar que esta máquina simula la estrategia call-by-need fuerte queda pendiente como trabajo futuro. Al momento de escribir esta tesis, la pregunta de si la estrategia call-by-need fuerte se puede implementar de forma *razonable* se encuentra abierta.

7.2. Definición de un procedimiento de extracción

En esta sección describimos un problema, actualmente no resuelto, que hallamos al tratar de caracterizar las familias de radicales en el LSC. En particular, proponemos un procedimiento de extracción, pero dejamos abierta la pregunta sobre si dicho procedimiento cuenta con todas las propiedades deseadas.

Definición 7.3 (No duplicación)**.** Escribimos $\rho \# S$ si ρ no duplica S, es decir, $\#(S/\rho) = 1$. Decimos que ρ no duplica σ, y escribimos $\rho \# \sigma$, de acuerdo con la siguiente definición inductiva:

$$\frac{}{\rho \# \epsilon} \qquad \frac{\rho \# S \quad \rho/S \# \sigma}{\rho \# S\sigma}$$

Observación 7.4. Si $\rho \# \sigma$ entonces σ/ρ tiene la misma longitud que σ.

Definición 7.5 (Derivación interna)**.** Decimos que un paso R es *interno a un contexto* $\mathtt{C}$, y escribimos $\mathtt{C} \prec R$, siempre que el origen de R es de la forma $\mathtt{C}\langle t\rangle$ y, además:

- Si R es un radical $\mathtt{db}$, la posición del agujero de $\mathtt{C}$ es un prefijo de la posición del patrón del radical $\mathtt{db}$.
- Si R es un radical $\mathtt{ls}$, la posición del agujero de $\mathtt{C}$ es un prefijo de la posición de la variable contraída por el radical $\mathtt{ls}$.

Además, una derivación ρ es *interna a un contexto* $\mathtt{C}$, de acuerdo con la siguiente definición inductiva:

$$\frac{}{\mathtt{C} \prec \epsilon} \qquad \frac{\mathtt{C} \prec R \quad \mathtt{C} \prec \rho}{\mathtt{C} \prec R\rho}$$

Si R es un radical $\mathtt{ls}$ y σ es una derivación componible, es decir, $\mathsf{tgt}(R) = \mathsf{src}(\sigma)$, decimos que la derivación σ es *interna al sujeto (respectivamente, al argumento)* de R, y escribimos $R \prec_{\mathtt{sbj}} \sigma$ (respectivamente, $R \prec_{\mathtt{arg}} \sigma$) cuando el radical R es de la forma $\mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x \rangle\!\rangle[x\backslash t]\rangle \to \mathtt{C}_1\langle \mathtt{C}_2\langle t\rangle[x\backslash t]\rangle$ y $\mathtt{C}_1\langle \mathtt{C}_2\langle \Box\rangle[x\backslash t]\rangle \prec \sigma$ (respectivamente, $\mathtt{C}_1\langle \mathtt{C}_2\langle t\rangle[x\backslash \Box]\rangle \prec \sigma$).

Ejemplo 7.6 (Derivaciones internas)**.** *Por ejemplo, considerar el siguiente diagrama:*

$$\begin{tikzcd}
 & (zy)[x\backslash yy][y\backslash z] \arrow[r, "T_1"] & (zz)[x\backslash yy][y\backslash z] \\
(yy)[x\backslash yy][y\backslash z] \arrow[ur, "S_1"] \arrow[dr, "S_2"] & & \\
 & (yy)[x\backslash zy][y\backslash z] \arrow[r, "T_2"] & (yy)[x\backslash zz][y\backslash z] \\
 & x[x\backslash zy][y\backslash z] \arrow[dr, "T"] & \\
x[x\backslash yy][y\backslash z] \arrow[uu, "R"] \arrow[ur, "S"] \arrow[dr, "T'"] & & x[x\backslash zz][y\backslash z] \\
 & x[x\backslash yz][y\backslash z] \arrow[ur, "S'"] &
\end{tikzcd}$$

Entonces se tiene que $\Box[x\backslash yy][y\backslash z] \prec S_1T_1$*, por lo tanto* $R \prec_{\mathtt{sbj}} S_1T_1$*, y* $(yy)[x\backslash\Box][y\backslash z] \prec S_2T_2$ *de modo que* $R \prec_{\mathtt{arg}} S_2T_2$*.*

Observar que si $R \prec_i S_i$ para $i \in \{\mathtt{sbj}, \mathtt{arg}\}$, entonces S_i tiene un ancestro S_0, es decir $S_i \in S_0/R$. Además, S_0/R consta de exactamente dos radicales, $S_{\mathtt{sbj}}$ y $S_{\mathtt{arg}}$, tales que S_i es interno a i. Observar además que S_0 no duplica el radical R. Esto se puede justificar con el siguiente diagrama:

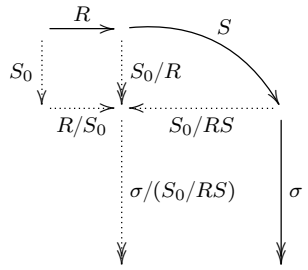
$$\begin{tikzcd}
\mathtt{C}_1\langle \mathtt{C}_2\langle\!\langle x \rangle\!\rangle[x\backslash \underline{t}]\rangle \arrow[r, "R"] \arrow[d, "S_0"] & \mathtt{C}_1\langle \mathtt{C}_2\langle \underline{t}\rangle[x\backslash \underline{t}]\rangle \arrow[d, "S_{\mathtt{sbj}}"] \arrow[d, shift left=4, "S_{\mathtt{arg}}"] \\
 &
\end{tikzcd}$$

Definición 7.7 (Retracción)**.** Si $R \prec_i S_i$ para $i \in \{\mathtt{sbj}, \mathtt{arg}\}$, escribimos $S_i \leftarrowtail R$ para el (único) ancestro de S_i, correspondiente a S_0 en el diagrama de arriba. Llamamos a $(S_i \leftarrowtail R)$ el *retracto* de S_i antes de R.

La noción de retracto se extiende a derivaciones. Si $R \prec_i \sigma$ para $i \in \{\mathtt{sbj}, \mathtt{arg}\}$, el *retracto* de σ antes R se escribe $\sigma \leftarrowtail R$ y se define inductivamente como sigue:

$$\begin{aligned}
\epsilon \leftarrowtail R &\stackrel{\text{def}}{=} \epsilon \\
S\sigma \leftarrowtail R &\stackrel{\text{def}}{=} S_0\,(\sigma/(S_0/RS) \leftarrowtail R/S_0) \quad \text{donde } S_0 = S \leftarrowtail R
\end{aligned}$$

La operación de retracción está bien definida porque R/S_0 es un conjunto que consta de un solo radical, dado que, como ya se mencionó, S_0 no duplica a R. Para ver que la definición inductiva está efectivamente bien definida, se puede verificar que $R/S_0 \prec_i \sigma/(S_0/RS)$ y, además, que la longitud de $\sigma/(S_0/RS)$ coincide con la longitud de σ, de tal modo que las ecuaciones recursivas están bien fundadas. El diagrama siguiente ilustra la situación:



Ejemplo 7.8 (Retracción)**.** *En la situación de Ej. 7.6, tenemos que* $(S_1T_1 \leftarrowtail R) = (S_2T_2 \leftarrowtail R) = ST$.

Por último, podemos dar la definición principal de esta sección:

Definición 7.9 (Procedimiento de extracción)**.** El procedimiento de extracción se formula como un sistema de reescritura cuyos objetos son radicales con historia. Los pasos de reescritura están dados por las dos reglas siguientes:

$$\begin{array}{lcll} \rho R(\sigma/R) & \rhd & \rho\sigma & \text{si } \sigma \neq \epsilon \text{ y } R \# \sigma \\ \rho R\sigma & \rhd & \rho(\sigma \leftarrowtail R) & \text{si } \sigma \neq \epsilon \text{ y } R \prec_i \sigma \text{ para algún } i \in \{\mathtt{sbj}, \mathtt{arg}\} \end{array}$$

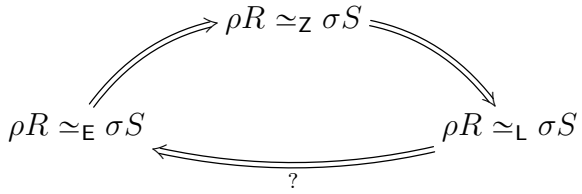
Ejemplo 7.10. *En la situación de Ej. 7.6, tenemos que* $RS_1 \rhd S$, $RS_2 \rhd S$, $RS_1T_1 \rhd ST \rhd T'$ *y* $RS_2T_2 \rhd ST \rhd T'$.

No es difícil demostrar que $\rhd$ es fuertemente normalizante. En efecto, se puede probar que si $\rho \rhd \sigma$, entonces la longitud de σ es estrictamente más chica que la longitud de ρ. Además, se puede definir una relación de *equivalencia de familias por extracción*, declarando que la relación $\rho R \simeq_{\mathsf{E}} \sigma S$ se verifica si y sólo si $\rho R\ (\rhd \cup \rhd^{-1})^*\ \sigma S$. Con esta definición, se puede además ver que $(\rho S \simeq_{\mathsf{E}} \sigma S) \implies (\rho S \simeq_{\mathsf{Z}} \sigma S)$.

Por otra parte, los dos problemas siguientes parecen ser no triviales, y los dejamos expresados en forma de conjeturas:

Conjetura 7.11. *El procedimiento de extracción* $\rhd$ *es confluente.*

Conjetura 7.12. *El procedimiento de extracción* $\rhd$ *caracteriza las familias de radicales. Más precisamente, se verifica la implicación* $(\rho S \simeq_{\mathsf{L}} \sigma S) \implies (\rho S \simeq_{\mathsf{E}} \sigma S)$*, cerrando el círculo de implicaciones:*



Bibliografía

[1] Beniamino Accattoli. An abstract factorization theorem for explicit substitutions. In *23rd International Conference on Rewriting Techniques and Applications (RTA'12), May 28 - June 2, 2012, Nagoya, Japan*, pages 6–21, 2012.

[2] Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 659–670. ACM, 2014.

[3] Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. *ACM SIGPLAN Notices*, 49(1):659–670, 2014.

[4] Beniamino Accattoli and Ugo Dal Lago. Beta reduction is invariant, indeed. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 8:1–8:10. ACM, 2014.

[5] Zena M Ariola and Matthias Felleisen. The call-by-need lambda calculus. *Journal of functional programming*, 7(3):265–301, 1997.

[6] Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. A call-by-need lambda calculus. In *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, pages 233–246, 1995.

[7] Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages.* Cambridge Tracts in Theoretical Computer Science. CUP, 1999.

[8] Gérard Berry. Stable models of typed λ-calculi. In *International Colloquium on Automata, Languages, and Programming*, pages 72–89. Springer, 1978.

[9] Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017.

[10] Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul.* PhD thesis, Ecole Doctorale Physique et Sciences de la Matière (Marseille), 2007.

[11] Stephen Chang and Matthias Felleisen. The call-by-need lambda calculus, revisited. In *European Symposium on Programming*, pages 128–147. Springer, 2012.

[12] Pierre Crégut. Strongly reducing variants of the krivine abstract machine. *Higher-Order and Symbolic Computation*, 20(3):209–230, 2007.

[13] Vincent Danos and Laurent Regnier. Head linear reduction. Technical report, 2004.

[14] Olivier Danvy. A rational deconstruction of landin's secd machine. In *IFL*, pages 52–71, 2004.

[15] Olivier Danvy and Ian Zerny. A synthetic operational account of call-by-need evaluation. In *PPDP*, pages 97–108, 2013.

[16] Matthias Felleisen and Daniel P. Friedman. Control operators, the SECD-machine, and the lambda-calculus. In *3rd Working Conference on the Formal Description of Programming Concepts*, August 1986.

[17] Mattias Felleisen and Daniel P Friedman. A calculus for assignments in higher-order languages. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, page 314. ACM, 1987.

[18] Philippa Gardner. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software*, pages 555–574. Springer, 1994.

[19] John R. W. Glauert and Zurab Khasidashvili. Relative normalization in deterministic residual structures. In Hélène Kirchner, editor, *Trees in Algebra and Programming - CAAP'96, 21st International Colloquium, Linköping, Sweden, April, 22-24, 1996, Proceedings*, volume 1059 of *Lecture Notes in Computer Science*, pages 180–195. Springer, 1996.

[20] Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In *ICFP '02, Pittsburgh, Pennsylvania, USA, October 4-6, 2002.*, pages 235–246, 2002.

[21] Gérard Huet. The zipper. *Journal of functional programming*, 7(5):549–554, 1997.

[22] Delia Kesner. Reasoning about call-by-need by means of types. In *International Conference on Foundations of Software Science and Computation Structures*, pages 424–441. Springer, 2016.

[23] Delia Kesner. Reasoning about call-by-need by means of types. In *Proceedings of the 19th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, pages 424–441. Springer-Verlag, 2016.

[24] Delia Kesner and Daniel Ventura. Quantitative types for the linear substitution calculus. In *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference (TCS'14), Rome, Italy, September 1-3, 2014.*, pages 296–310, 2014.

[25] Donald E Knuth. Ancient babylonian algorithms. *Communications of the ACM*, 15(7):671–677, 1972.

[26] Jean-Louis Krivine. *Lambda-calculus, types and models.* Ellis Horwood, 1993.

[27] Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-order and symbolic computation*, 20(3):199–207, 2007.

[28] Peter J Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

[29] Xavier Leroy. *The ZINC experiment: an economical implementation of the ML language.* PhD thesis, INRIA, 1990.

[30] Jean-Jacques Lévy. *Réductions correctes et optimales dans le lambda-calcul.* PhD thesis, Université de Paris 7, 1978.

[31] Jean-Jacques Lévy. Optimal reductions in the lambda calculus. *Essays on Combinatory Logic, Lambda Calculus and Formalism*, 1980.

[32] Jean-Jacques Levy. Redexes are stable in the λ-calculus. 27:1–13, 07 2015.

[33] John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. *Journal of functional programming*, 8(3):275–317, 1998.

[34] Paul-André Melliès. *Description Abstraite des Systèmes de Réécriture.* PhD thesis, Université Paris 7, december 1996.

[35] Peter Sestoft. Deriving a lazy abstract machine. *Journal of Functional Programming*, 7(3):231–264, 1997.

[36] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*, volume 149. Elsevier, 2006.

[37] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 2003.