Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Semántica denotacional para un cálculo-λ relacional

Tesis de Licenciatura en Ciencias de la Computación

Mariana Milicich

Director: Pablo Barenbaum

Buenos Aires, 2022

RESUMEN

En esta tesis trabajamos con el cálculo-$\lambda^{\mathtt{U}}$, una extensión del cálculo-λ que incorpora las características fundamentales de la programación relacional: alternativa no determinística, secuenciación explícita, unificación de primer orden e introducción de variables frescas. Proponemos un sistema de tipos y formulamos una *semántica denotacional* para su fragmento tipado. Por semántica denotacional entendemos a una función $[\![-]\!]$ que dado un programa devuelve su significado o *denotación*, es decir, un elemento de algún dominio de interpretación apropiado. El objetivo es demostrar que la semántica cumple con propiedades esperables: por un lado, probar la *correctitud* de la semántica operacional con respecto a la denotacional, que asegura que dos programas equivalentes de acuerdo con una teoría sintáctica de igualdad deben tener la misma denotación; por otra parte, la propiedad de *completitud* de la semántica operacional con respecto a la denotacional, que asegura que dos programas con la misma denotación se pueden probar equivalentes en una teoría sintáctica de igualdad. En este trabajo logramos formular una semántica denotacional para la cual la semántica operacional verifica una forma débil de correctitud. Queda como trabajo futuro proponer una semántica denotacional para que la operacional sea correcta y completa.

Palabras claves: Semántica denotacional, cálculo-λ, unificación, programación funcional, programación lógica, programación relacional.

ABSTRACT

In this thesis we work with the $\lambda^{\mathtt{U}}$-calculus, an extension of the λ-calculus that incorporates the fundamental characteristics of relational programming: non-deterministic alternative, explicit secuenciation, first-order unification and introduction of free variables. We propose a type system and formulate a *denotational semantics* for its typed fragment. By denotational semantics we mean a function $[\![-]\!]$ which, given a program, returns its meaning or *denotation*, that is, an element of some appropiate interpretacion domain. The goal of this work is to prove that the semantics meets the expected properties: on the one hand, show the *soundness* of the operational semantics with respect to the denotationali semantics, which ensures that two equivalent programs in accordance with a syntactic theory of equality must have the same denotation; and on the other hand, the property of *completeness* of the operational semantics with respect to the denotational semantics, which assures that two programs with the same denotation can be shown equivalents in a syntactic theory of equivalence. In this work we managed to formulate a denotational semantics for which the operational semantics verifies a weak form of soundness. It remains as future work to propose a denotational semantics for which the operational semantics is both sound and complete.

Keywords: Denotational semantics, λ-calculus, unification, functional programming, logic programming, relational programming.

AGRADECIMIENTOS

Quiero agradecer en primer lugar a Pablo por aceptar dirigir mi tesis, y todo lo que eso conlleva. Agradezco a Jano y a Hernán Melgratti por haber aceptado ser jurado de mi tesis, así como también a Santi Figueira que aceptó, y que por una situación de fuerza mayor no pudo llevar a cabo la tarea. Por fuera de la tesis en sí, estoy agradecida con la comunidad del DC por haberme dado un lugar de pertenencia; instantáneamente desde que me cambié a esta carrera sentí al departamento como mi hogar. En todos los roles que ocupé dentro del DC aprendí muchísimo y me encontré con gente que me ayudó un montón, que son personas que inspiran y que da gusto poder trabajar y/o estudiar a su lado. En particular, me gustaría agradecer nuevamente a Jano por la ayuda que nos dio a Pablo y a mí durante el primer año en la pasantía, y por dirigir y llevar adelante al grupo de Lógica y Reescritura para Lenguajes de programación, es admirable el trabajo que hace por el grupo. A las amistades que fui formando en estos años les agradezco por todo el aguante y los tiempos muertos que siempre llenamos con charlas, meriendas, sesiones de estudio y una infinidad de recuerdos agradables; y también menciono, ya que nos tocó estar tres cuatrimestres así, las reuniones virtuales para estudiar y hacer catarsis durante la cuarentena. La lista no está completa pero me gustaría destacar a Balej, Caro, Gian, Guilla, Lu, Marce, Rozen y Tobi. También quiero agradecerles a quienes conformaron los planteles docentes de Lógica y Computabilidad, PLP y Taller de álgebra cuando las cursé como cuando fui docente: son mis tres materias favoritas, y la gente que estuvo a cargo del dictado influyó en gran parte a esta preferencia. Además quiero agradecerle a mis familiares, amistades por fuera del DC y a mi novio, porque su ayuda y apoyo fueron clave para que pudiera encaminarme hacia esta carrera y poder dedicarme a lo que me gusta hacer realmente.

Índice general

1.. Introducción . . . 1
- 1.1. Semántica operacional y denotacional . . . 1
- 1.2. Trabajo relacionado . . . 4
  - 1.2.1. Lenguajes relacionales . . . 4
  - 1.2.2. Lenguajes lógico–funcionales . . . 4
  - 1.2.3. Cálculos con elección no determinística . . . 5
  - 1.2.4. Cálculos con *pattern matching* . . . 5
- 1.3. Descripción informal del cálculo-$\lambda^{\mathtt{U}}$ . . . 5
- 1.4. Objetivo de la tesis . . . 6
- 1.5. Contribuciones . . . 7
- 1.6. Estructura de la tesis . . . 7

2.. Preliminares . . . 9
- 2.1. Sintaxis del cálculo-$\lambda^{\mathtt{U}}$ . . . 9
- 2.2. Algoritmo de unificación . . . 12
  - 2.2.1. Ejemplos . . . 13
- 2.3. Semántica operacional del cálculo-$\lambda^{\mathtt{U}}$ . . . 14

3.. Tipado del cálculo-$\lambda^{\mathtt{U}}$ . . . 17
- 3.1. Propiedades básicas del sistema de tipos . . . 18
  - 3.1.1. Tipado de problemas de unificación . . . 23
  - 3.1.2. Preservación de tipos . . . 25

4.. Semántica denotacional para el cálculo-$\lambda^{\mathtt{U}}$ . . . 29
- 4.1. Interpretaciones de tipos y términos . . . 29
- 4.2. Propiedades básicas de la semántica denotacional . . . 32

5.. Limitaciones de la semántica denotacional propuesta . . . 43
- 5.1. Variante 1: semántica denotacional con memoria . . . 44
- 5.2. Variante 2: revisión del cálculo para incorporar clausuras . . . 47
  - 5.2.1. Sintaxis y semántica operacional del cálculo-$\lambda^{\mathtt{UC}}$ . . . 48
  - 5.2.2. Algoritmo de unificación . . . 50
  - 5.2.3. Semántica operacional del cálculo $\lambda^{\mathtt{UC}}$ . . . 52
  - 5.2.4. Dificultades . . . 53

6.. Conclusiones y trabajo futuro . . . 55

Bibliografía . . . 57

1. INTRODUCCIÓN

Los lenguajes de programación cuentan con características que suelen agruparse por *paradigmas*, entre los que se destacan la programación funcional y la programación lógica. La programación funcional se caracteriza por la presencia de *funciones de orden superior* —capaces de recibir otras funciones como argumentos— y *tipos de datos algebraicos*, sobre los que se pueden dar construcciones inductivas utilizando *pattern matching*. La programación lógica se caracteriza por la presencia de variables simbólicas que se instancian a través del mecanismo de *unificación*, como así también por el comportamiento no determinístico implementado a través del *backtracking*, y por la concepción de los programas como relaciones.

Además de lenguajes de programación puramente funcionales, como Haskell y Ocaml, y los puramente lógicos como Prolog y Mercury, existen lenguajes funcionales–lógicos como λ-Prolog y Curry, que incorporan características de ambos paradigmas. La teoría de los lenguajes puramente funcionales y la de los puramente lógicos se encuentran plenamente desarrolladas, pero no se ha desarrollado, a nuestro saber y entender, un modelo formal que combine conjuntamente las características de ambos paradigmas y cuente con una metateoría satisfactoria a la par de la que, por ejemplo, sí se conoce para el cálculo-λ[1]. La importancia de contar con este tipo de resultados teóricos —tales como confluencia, terminación del fragmento tipado, estandarización, etc.— es que estos permiten razonar rigurosamente sobre el comportamiento de los programas, y justifica la corrección de técnicas de implementación de compiladores, intérpretes, optimizadores y otros componentes.

Recientemente [3] se ha propuesto una extensión del cálculo-λ que incorpora, además de características funcionales, las características fundamentales del paradigma lógico: alternativa no determinística, secuenciación explícita y unificación de primer orden. El sistema está formulado mediante una semántica de reducción *small-step*, es decir, con reglas de reescritura que indican cómo una expresión puede simplificarse paso a paso hasta llegar a un resultado. Además, el sistema es confluente y dispone de un sistema de tipos.

1.1. Semántica operacional y denotacional

En esta sección repasamos las distintas nociones de semántica que se pueden definir para el cálculo-λ simplemente tipado, y en particular la semántica operacional y la semántica denotacional clásicas.

Comenzamos definiendo la sintaxis del cálculo-λ simplemente tipado. Supongamos dado un conjunto infinito numerable de *variables*, $\mathsf{Var} = \{x, y, z, \ldots\}$ y un conjunto infinito numerable de tipos base, $\{\alpha, \beta, \gamma, \ldots\}$. Los conjuntos de *tipos* ($\mathsf{Type} = \{A, B, \ldots\}$), *términos* ($\mathsf{Term} = \{t, s, \ldots\}$) y *valores* ($\mathsf{Val} = \{\mathtt{v}, \mathtt{w}, \ldots\}$) se definen inductivamente como sigue. Tipos:

$$A ::= \alpha \mid A \to A$$

donde α representa cualquier tipo base.

[1] Una excepción es [1], donde se estudia una semántica operacional *big-step* para un cálculo-λ extendido con características lógicas.

Términos:

$$\begin{array}{rcll} t & ::= & x^A & \text{variable} \\ & | & \lambda x^A.\, t & \text{abstracción} \\ & | & t\, s & \text{aplicación} \end{array}$$

Valores:

$$\mathtt{v} \quad ::= \quad \lambda x^A.\, t$$

Observemos que en esta presentación del cálculo-λ el tipado es *intrínseco* ("*à la* Church"), es decir, los términos están decorados explícitamente con su tipo. Generalmente omitimos las decoraciones de tipo sobre las variables, y escribimos por ejemplo $\lambda x.\, x$ en lugar de $\lambda x^A.\, x^A$ si el tipo A se puede deducir del contexto o no tiene especial relevancia.

Una ocurrencia de una variable x está *ligada* si se encuentra bajo el alcance de una abstracción de la forma $\lambda x^A.\, \ldots$, y de lo contrario está *libre*. Los términos se consideran módulo renombre de variables ligadas, por ejemplo $\lambda x^A.\, x^A = \lambda y^A.\, y^A$. Un término se dice *cerrado* si no tiene ocurrencias de variables libres.

Recordemos también que los juicios de tipado son de la forma $\Gamma \vdash t : A$, donde Γ es un *contexto de tipado*, es decir, un conjunto finito de asignaciones de tipos a variables, de la forma $x : A$. Las reglas de tipado son las siguientes:

$$\frac{x : A \in \Gamma}{\Gamma \vdash x^A : A}\ \texttt{T-Var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.\, t : A \to B}\ \texttt{T-Abs} \qquad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash s : A}{\Gamma \vdash t\, s : B}\ \texttt{T-App}$$

El significado de las expresiones e instrucciones de un lenguaje de programación se define a través de una o más *semánticas*, que describen distintas maneras de interpretar el *programa*, que en principio lo podemos definir como un término.

Dos tipos de semánticas que nos interesan en este trabajo son la *semántica operacional* y la *semántica denotacional*. Desde la perspectiva de la semántica operacional, el programa representa un estado en el cómputo y la semántica indica el mecanismo que convierte un estado en otro, hasta llegar a un estado final que representa el resultado del cómputo. En el caso de la semántica operacional *small step*, la transformación de estados es paso a paso y está reflejada en una relación de reducción o transición "$\to$" que dado un estado indica cuál es el siguiente. Por otro lado, desde la perspectiva de la semántica denotacional, el programa se interpreta como un valor matemático en algún dominio de interpretación. Así, la semántica establece una relación entre las expresiones del lenguaje y los elementos del dominio de interpretación (*i.e.* el significado de las expresiones).

Semántica operacional. A partir de las siguientes reglas, definimos una semántica operacional small step para el cálculo-λ simplemente tipado, basada en la estrategia de evaluación *call-by-value*.

$$\frac{t \to t'}{t\, s \to t'\, s}\ \mu \qquad \frac{s \to s'}{\mathtt{v}\, s \to \mathtt{v}\, s'}\ \nu$$

$$\frac{}{(\lambda x.\, t)\, \mathtt{v} \to t\{x := \mathtt{v}\}}\ \beta_v$$

donde $t\{x := \mathtt{v}\}$ denota la sustitución de las ocurrencias libres de x en t por $\mathtt{v}$, evitando la captura de variables.

Resumimos sin demostración algunos resultados de la semántica operacional. Más detalles pueden encontrarse en el libro de Pierce [13].

Proposición 1.1.1 (Progreso)**.** *Sea t un término cerrado y bien tipado,* i.e., $\vdash t : A$. *Entonces t es un valor o existe un término s tal que $t \to s$.*

Proposición 1.1.2 (Preservación)**.** *Sean t y s dos términos tales que $t \to s$ y $\Gamma \vdash t : A$. Entonces $\Gamma \vdash s : A$.*

Proposición 1.1.3 (Normalización)**.** *Si $\Gamma \vdash t : A$, entonces t es* normalizable*,* i.e. *el cómputo de t termina en una cantidad finita de pasos.*

Semántica denotacional. Antes de proponer una semántica denotacional para el cálculo, comenzamos definiendo conceptos a tener en cuenta. En primer lugar, a cada tipo A le asociamos un conjunto que notamos $[\![A]\!]$, también conocido como el *dominio de interpretación* asociado a A. Para ello, suponemos fijado un conjunto S_α para cada tipo base α, y definimos $[\![A]\!]$ recursivamente como sigue:

$$\begin{aligned} [\![\alpha]\!] &\stackrel{\text{def}}{=} \mathsf{S}_\alpha \\ [\![A \to B]\!] &\stackrel{\text{def}}{=} [\![B]\!]^{[\![A]\!]} \end{aligned}$$

donde Y^X denota el conjunto de funciones $f : X \to Y$.

Por ejemplo, una posible interpretación del tipo base Bool sería $\mathsf{S}_{\mathsf{Bool}} = \{0, 1\}$. Bajo esa interpretación, $[\![\mathsf{Bool} \to \mathsf{Bool}]\!]$ sería el conjunto de todas las funciones $f : \{0, 1\} \to \{0, 1\}$. En este trabajo, asumimos que la interpretación de los tipos base es siempre un conjunto **no vacío**.

Una *asignación de variables* es una función ρ que a cada variable le asocia un elemento de algún dominio de interpretación, es decir $\rho : \mathsf{Var} \to \bigcup_{A \in \mathsf{Type}} [\![A]\!]$. Además, si Γ *es un contexto de tipado*, decimos que ρ es *compatible con* Γ (notado $\Gamma \models \rho$) si y sólo si a cada variable de Γ se le asocia un elemento del dominio de interpretación de su correspondiente tipo, es decir, cada vez que $(x : A) \in \Gamma$ se tiene que $\rho(x) \in [\![A]\!]$. Si ρ es una asignación de variables y $a \in [\![A]\!]$, entonces $\rho[x \mapsto a]$ es la asignación de variables definida como sigue:

$$\rho[x \mapsto a](y) \stackrel{\text{def}}{=} \begin{cases} a & \text{si } x = y \\ \rho(y) & \text{en otro caso} \end{cases}$$

La *interpretación de términos* es una función $[\![-]\!]_{-}$ que dado un término de tipo A bajo el contexto Γ y una asignación de variables ρ compatible con Γ devuelve su significado o denotación, que es un elemento de $[\![A]\!]$. Esta función se define recursivamente como sigue:

$$\begin{aligned} [\![x^A]\!]_\rho &\stackrel{\text{def}}{=} \rho(x) & \\ [\![\lambda x^A.\, t]\!]_\rho &\stackrel{\text{def}}{=} f & \text{donde } f(a) = [\![t]\!]_{\rho[x \mapsto a]} \\ [\![t\, s]\!]_\rho &\stackrel{\text{def}}{=} [\![t]\!]_\rho([\![s]\!]_\rho) & \end{aligned}$$

Resumimos sin demostración algunas propiedades básicas de la semántica denotacional. Más detalles pueden encontrarse en el libro de Gunter [6].

Proposición 1.1.4 (Irrelevancia)**.** *Sea $\Gamma \vdash t : A$ y sean ρ y ρ' asignaciones de variables compatibles con Γ que coinciden en las variables de Γ, es decir, $\rho(x) = \rho'(x)$ para toda variable $(x : A) \in \Gamma$. Entonces $[\![t]\!]_\rho = [\![t]\!]_{\rho'}$.*

Proposición 1.1.5 (Composicionalidad)**.** *Sean $\Gamma, x : A \vdash t : B$ y $\Gamma \vdash s : A$, y sea ρ una asignación de variables compatible con Γ. Entonces $[\![t\{x := s\}]\!]_\rho = [\![t]\!]_{\rho[x \mapsto [\![s]\!]_\rho]}$.*

Teorema 1.1.6 (Correctitud). *Sea* $\Gamma \vdash t : A$*, sea* $t \to t'$ *un paso de reducción*[2] *y sea* ρ *una asignación de variables compatible con* Γ*. Entonces* $[\![t]\!]_\rho = [\![t']\!]_\rho$.

1.2. Trabajo relacionado

Como mencionamos al principio de este capítulo, ya se han desarrollado lenguajes de programación funcionales–lógicos. En esta sección describimos brevemente algunos de ellos, como así también algunos lenguajes del paradigma relacional (o puramente lógico) y del paradigma funcional que se relacionan en alguna medida con el cálculo-$\lambda^{\mathtt{U}}$.

1.2.1. Lenguajes relacionales

El punto de partida del cálculo-$\lambda^{\mathtt{U}}$ es extender el cálculo-λ (no tipado) con características de la familia de lenguajes miniKanren, perteneciente al paradigma relacional. Los lenguajes de la familia de miniKanren tienen en común con el cálculo-$\lambda^{\mathtt{U}}$ los operadores de unificación, introducción de variables frescas, conjunción (correspondiente a la secuencia del cálculo-$\lambda^{\mathtt{U}}$) y disyunción (correspondiente a la alternativa no determinística del cálculo-$\lambda^{\mathtt{U}}$). Se diferencian en que miniKanren está limitado a operar con términos de primer orden, es decir, no cuenta con abstracciones ni aplicaciones. En un trabajo reciente, Rozplokhas, Vyatkin y Boulytchev estudian la semántica operacional y la semántica denotacional de miniKanren [15].

1.2.2. Lenguajes lógico–funcionales

En la literatura se encuentran varias propuestas de lenguajes que combinan características de la programación funcional con la programación lógica. Mencionamos dos de los más destacados.

Miller y Nadathur [12] proponen una extensión del lenguaje relacional Prolog, que en lugar de trabajar con términos de primer orden, opera con λ-términos. El lenguaje se llama λProlog, y cuenta con implementaciones, algunas de ellas recientes, como Makam [17]. En cuanto a las diferencias con el cálculo-$\lambda^{\mathtt{U}}$, el modelo de ejecución de λProlog está basado en el método de resolución, mientras que en el cálculo-$\lambda^{\mathtt{U}}$ el modelo de ejecución se basa en reescritura de términos. En particular, el cálculo-$\lambda^{\mathtt{U}}$ es confluente, mientras que no estamos al tanto de trabajos que aborden este resultado en λProlog.

Por otra parte, Miller [11] identificó una restricción del problema de unificación de orden superior conocida como *unificación de patrones de orden superior* (*higher order pattern unification*). Este fragmento tiene como buena propiedad que es decidible y admite unificadores más generales. Para llevar a cabo estas pruebas utiliza un lenguaje llamado L_λ [10], el cual extiende un lenguaje lógico con λ-abstracciones. Las abstracciones tienen algunas restricciones en las ocurrencias de las variables ligadas por las lambdas, para evitar tener unificación de orden superior y así poder extender de forma decidible la unificación de primer orden.

Otro lenguaje de programación lógico–funcional es Curry [7], propuesto por Hanus y colaboradores. El modelo de cómputo de Curry se basa en combinar las técnicas de *narrowing* y *residuation*, y permite la elección del mecanismo de evaluación dependiendo del caso en que se encuentre la expresión a evaluar. La primera técnica es una generalización

[2] Observemos que vale también $\Gamma \vdash t' : A$ por el teorema de preservación de tipos.

de reescritura, y se utiliza cuando las expresiones a evaluar tienen variables que necesitan ser instanciadas para que la ejecución de la expresión pueda proceder. La segunda técnica permite una evaluación determinística y eficiente de llamados a funciones.

1.2.3. Cálculos con elección no determinística

Dentro de la amplia variedad de lenguajes del paradigma funcional, se han estudiado muchos cálculos que incorporan alguna noción de elección no determinística. Por ejemplo, el cálculo λ_{ndlr}, propuesto por Schmidt-Schauß y Huber [16], extiende el cálculo-λ con una noción de elección no determinística (errática) entre dos expresiones, correspondiente a la alternativa no determinística del cálculo-$\lambda^{\mathtt{U}}$. Los cálculos con elección no determinística se usan también para estudiar reescritura en un marco probabilístico [14, 4]. Sin embargo, estos cálculos no cuentan con unificación.

1.2.4. Cálculos con *pattern matching*

Se han propuesto muchos cálculos que extienden el cálculo-λ con *pattern matching*. Un ejemplo destacable es el *Pure Pattern Calculus* (PPC), de Jay y Kesner [9]. Este cálculo generaliza al cálculo-λ de tal modo que las abstracciones, en lugar de ser de la forma $\lambda x.\, t$, son de la forma $\lambda p.\, t$ donde p es un *patrón*. La regla β se modifica para que $(\lambda p.\, t)\, s$ haga coincidir el argumento s contra el patrón p, es decir, haga *pattern-matching*. El hilo conductor de PPC es que cualquier término puede hacer las veces de patrón, lo que permite programar con patrones dinámicos, es decir, patrones que se calculan en tiempo de ejecución. Dado que la unificación de términos es una generalización del *pattern-matching*, sería esperable que un cálculo con unificación como el cálculo-$\lambda^{\mathtt{U}}$ resulte ser una generalización de los cálculos con *pattern matching* como PPC.

1.3. Descripción informal del cálculo-$\lambda^{\mathtt{U}}$

En esta sección definimos de manera intuitiva el cálculo que estudiamos en este trabajo.

Como mencionamos en la sección previa, el cálculo-$\lambda^{\mathtt{U}}$ se basa en partir de los términos usuales del cálculo-λ no tipado: variable, abstracción y aplicación, y además incluimos algunos términos del lenguaje miniKanren: unificación entre dos términos, secuenciación, introducción de variables frescas y alternativa no determinística. Una sintaxis inicial es la siguiente, suponiendo dado un conjuntos infinitos numerables de variables $(x, y, z, \ldots)$ y de locaciones $(\ell_1, \ell_2, \ldots)$

Términos:

$$
\begin{array}{rclll}
t & ::= & x & \text{variable} \\
 & | & \mathbf{c} & \text{constructor} \\
 & | & \lambda x.\, P & \text{abstracción} \\
 & | & \lambda^{\ell} x.\, P & \text{abstracción alojada} \\
 & | & t\, s & \text{aplicación} \\
 & | & t \doteq s & \text{unificación} \\
 & | & t; s & \text{secuencia} \\
 & | & \nu x.\, t & \text{declaración de variable fresca}
\end{array}
$$

Programas:

$$
\begin{array}{rclll}
P & ::= & \mathtt{fail} & \text{programa vacío} \\
 & | & t \oplus P & \text{alternativa no determinística}
\end{array}
$$

En el capítulo siguiente detallamos la semántica operacional formal, pero introducimos a continuación la semántica informal de los términos y programas. Las variables tienen su significado usual, como parámetros formales, pero además también pueden ser instanciadas a través de la unificación. Los constructores son símbolos que no se pueden instanciar como por ejemplo $\mathbf{zero}, \mathbf{succ}$ y $\mathbf{nil}$. Las abstracciones tienen su significado usual, con la diferencia de que el cuerpo de la función es un *programa* (que representa la elección no determinística entre varios posibles términos). Además, hay abstracciones alojadas que se comportan igual que las abstracciones pero además están decoradas con una *locación.* La aplicación tiene su significado usual. El operador de unificación se comporta como el operador de unificación de lenguajes lógicos como Prolog, con la diferencia de que en lugar de unificar términos de primer orden se unifican términos del cálculo; la unificación puede fallar o tener éxito, lo que provoca como efecto secundario la instanciación de ciertas variables. La evaluación de la secuencia $t; s$ procede evaluando t y s de forma secuencial: si la ejecución de t falla, el término entero falla, mientras que si la ejecución de t tiene éxito, los resultados van a ser los que devuelva la evaluación de s. La introducción de variables frescas $\nu x.\, t$ evalúa t sustituyendo las ocurrencias de x en t por una variable fresca, es decir, una variable que no haya sido usada antes. El significado del programa vacío puede verse como el programa que no arroja ningún resultado. La alternativa no determinística $t \oplus P$ evalúa t y P en paralelo y produce posiblemente muchos resultados, juntando los resultados que produce t y los que produce P.

En el cálculo-$\lambda^{\mathtt{U}}$ hay términos que no necesariamente son de primer orden debido a la presencia de abstracciones. A priori, esto significaría que los problemas de unificación pasan a ser de orden superior (*higher order unification*). Por ejemplo, si tuviéramos el problema de unificación $f\,\mathbf{c} \overset{\bullet}{=} \mathbf{c}$ dos unificadores posibles podrían ser $\{f \mapsto \lambda x.\, x\}$ y $\{f \mapsto \lambda x.\, \mathbf{c}\}$. Desafortunadamente, el problema de unificación de orden superior es indecidible y más aún, no necesariamente admite unificador más general (ver [8] y [5]). Para zanjar este problema, el cálculo-$\lambda^{\mathtt{U}}$ impone la restricción de que en los problemas de unificación sólo se unifican valores. Si se tiene $t \overset{\bullet}{=} s$, se deben reducir t y s hasta que sean valores antes de poder proceder a resolver el problema de unificación. El conjunto de valores se define como el conjunto que incluye a las variables, las abstracciones alojadas y las *estructuras*, que son constructores aplicados (recursivamente) a valores. Por ejemplo, $\mathbf{cons}\,\mathbf{zero}\,(\mathbf{cons}\,(\mathbf{succ}\,\mathbf{zero})\,x)$ es un valor. Con esto en cuenta, dos abstracciones alojadas unifican cuando están decoradas con la misma locación.

Por ejemplo, el término

$$PRED = \lambda n.\, (((n \overset{\bullet}{=} \mathbf{zero}); \mathbf{zero}) \oplus (\nu x.\, ((n \overset{\bullet}{=} \mathbf{succ}\, x); x)) \oplus \mathtt{fail})$$

es la función que dado un natural n calcula el predecesor de n. El cuerpo de la función tiene dos alternativas. En la primera, se pide que n unifique con $\mathbf{zero}$ y en tal caso devuelve $\mathbf{zero}$. En la segunda, se pide que n unifique con $\mathbf{succ}\, x$, con x una variable fresca, y en este caso devuelve x.

1.4. Objetivo de la tesis

El objetivo que proponemos en esta tesis es el de formular una *semántica denotacional* para el fragmento tipado del cálculo-$\lambda^{\mathtt{U}}$ mencionado en la sección anterior. Pretendemos demostrar que la semántica cumple con propiedades esperables, en particular, la propiedad de *correctitud* de la semántica operacional con respecto a la denotacional debe asegurar que

si dos programas son *interconvertibles* (es decir, uno se puede convertir en el otro usando las reglas de reducción de la semántica operacional), entonces la semántica denotacional les atribuye la misma denotación. A la inversa, la propiedad de *completitud* asegura que dos programas con la misma denotación deben ser interconvertibles.

1.5. Contribuciones

En esta tesis:

- Definimos un sistema de tipos para el cálculo-$\lambda^{\mathtt{U}}$ (Definición 3.0.2).
- Mostramos que el sistema de tipos satisface varias propiedades básicas, entre las cuales la principal es la de preservación de tipos (Proposición 3.1.9).
- Definimos una noción de semántica denotacional para el cálculo-$\lambda^{\mathtt{U}}$ (Definición 4.1.4).
- Mostramos que la semántica denotacional propuesta cumple varias propiedades, incluyendo composicionalidad (Lema 4.2.2) y una noción débil de correctitud (4.2.8) de la semántica operacional con respecto a la semántica denotacional.

Parte del trabajo de la tesis fue presentado y publicado en el *17th International Colloquium on Theoretical Aspects of Computing* (ICTAC 2020).

1.6. Estructura de la tesis

En el capítulo 2 presentamos formalmente el cálculo-$\lambda^{\mathtt{U}}$, originalmente propuesto por Barenbaum y Lochbaum en 2020. Recordamos la sintaxis y la semántica operacional del cálculo. Además definimos conceptos necesarios para definir la semántica operacional, y damos ejemplos de reducciones de términos.

En el capítulo 3 definimos el sistema de tipos del cálculo-$\lambda^{\mathtt{U}}$, junto con la demostración de propiedades básicas sobre este. En particular, demostramos las propiedades de Debilitamiento, Fortalecimiento y Preservación de tipos.

En el capítulo 4 damos una semántica denotacional para el cálculo-$\lambda^{\mathtt{U}}$. Además, demostramos propiedades intermedias necesarias para enunciar y probar la propiedad de correctitud débil de la semántica operacional con respecto a la semántica denotacional.

El capítulo 5 está dedicado a discutir falencias de la semántica denotacional. Dos limitaciones son el hecho de que la correctitud sólo vale en un sentido débil y que la semántica operacional no es completa con respecto a la semántica denotacional.

En el capítulo 6 concluimos este trabajo, e incluimos trabajo futuro.

2. PRELIMINARES

En esta sección describimos con mayor profundidad el cálculo-$\lambda^{\mathtt{U}}$. Escribimos formalmente su sintaxis, definiendo sus términos, valores y programas. También extendemos el algoritmo de unificación de Martelli-Montanari y ejemplificamos su uso. Por último, damos las reglas de semántica operacional para el cálculo.

2.1. Sintaxis del cálculo-$\lambda^{\mathtt{U}}$

Definición 2.1.1 (Sintaxis)**.** Supongamos dados conjuntos infinitos de *variables* $\mathsf{Var} = \{x, y, z, \ldots\}$, *constructores* $\mathsf{Con} = \{\mathbf{c}, \mathbf{d}, \mathbf{e}, \ldots\}$, y *locaciones* $\mathsf{Loc} = \{\ell, \ell', \ell'', \ldots\}$. Asumimos que existe un constructor distinguido, **ok**. Los conjuntos de *términos* ($\{t, s, \ldots\}$), *valores* ($\mathsf{Val} = \{\mathtt{v}, \mathtt{w}, \ldots\}$), *programas* ($\{P, Q, \ldots\}$), y *contextos de evaluación débiles* ($\{\mathsf{W}, \mathsf{W}', \ldots\}$) se definen de manera mutuamente inductiva:

Términos:

$$\begin{array}{rcll}
t & ::= & x & \text{variable} \\
 & \mid & \mathbf{c} & \text{constructor} \\
 & \mid & \lambda x.\, P & \text{abstracción} \\
 & \mid & \lambda^{\ell} x.\, P & \text{abstracción alojada} \\
 & \mid & t\, s & \text{aplicación} \\
 & \mid & t \stackrel{\bullet}{=} s & \text{unificación} \\
 & \mid & t; s & \text{secuencia} \\
 & \mid & \nu x.\, t & \text{declaración de variable fresca}
\end{array}$$

Valores:

$$\begin{array}{rcl}
\mathtt{v} & ::= & x \\
 & \mid & \lambda^{\ell} x.\, P \\
 & \mid & \mathbf{c}\, \mathtt{v}_1 \ldots \mathtt{v}_n
\end{array}$$

Programas:

$$\begin{array}{rcll}
P & ::= & \mathtt{fail} & \text{programa vacío} \\
 & \mid & t \oplus P & \text{alternativa no determinística}
\end{array}$$

Un *contexto* es un término que tiene exactamente una ocurrencia libre de una variable distinguida que notamos "$\Box$" y llamamos *agujero*. Los *contextos de evaluación débiles* son un subconjunto de los contextos dados por la siguiente gramática:

$$\begin{array}{rcll}
\mathsf{W} & ::= & \Box & \text{contexto vacío} \\
 & \mid & \mathsf{W}\, t & \text{izquierda de una aplicación} \\
 & \mid & t\, \mathsf{W} & \text{derecha de una aplicación} \\
 & \mid & \mathsf{W} \stackrel{\bullet}{=} t & \text{izquierda de una unificación} \\
 & \mid & t \stackrel{\bullet}{=} \mathsf{W} & \text{derecha de una unificación} \\
 & \mid & \mathsf{W}; t & \text{izquierda de una secuencia} \\
 & \mid & t; \mathsf{W} & \text{derecha de una secuencia}
\end{array}$$

Notemos que los contextos débiles no entran bajo abstracciones o declaraciones de variables frescas; por ejemplo, $\nu x.\, \Box$ no es un contexto de evaluación débil, pero $\Box \stackrel{\bullet}{=} x$ sí.

Definición 2.1.2 (Variables libres y locaciones)**.** El conjunto de *variables libres* de un término ($\mathsf{fv}(t)$) y de un programa ($\mathsf{fv}(P)$) se definen recursivamente de la siguiente forma:

$$\begin{array}{rcl}
\mathsf{fv}(x) & \stackrel{\text{def}}{=} & \{x\} \\
\mathsf{fv}(\mathbf{c}) & \stackrel{\text{def}}{=} & \varnothing \\
\mathsf{fv}(\lambda x.\, P) & \stackrel{\text{def}}{=} & \mathsf{fv}(P) \setminus \{x\} \\
\mathsf{fv}(\lambda^{\ell} x.\, P) & \stackrel{\text{def}}{=} & \mathsf{fv}(P) \setminus \{x\} \qquad \mathsf{fv}(\mathtt{fail}) \stackrel{\text{def}}{=} \varnothing \\
\mathsf{fv}(t\, s) & \stackrel{\text{def}}{=} & \mathsf{fv}(t) \cup \mathsf{fv}(s) \quad \mathsf{fv}(t \oplus P) \stackrel{\text{def}}{=} \mathsf{fv}(t) \cup \mathsf{fv}(P) \\
\mathsf{fv}(t \stackrel{\bullet}{=} s) & \stackrel{\text{def}}{=} & \mathsf{fv}(t) \cup \mathsf{fv}(s) \\
\mathsf{fv}(t; s) & \stackrel{\text{def}}{=} & \mathsf{fv}(t) \cup \mathsf{fv}(s) \\
\mathsf{fv}(\nu x.\, t) & \stackrel{\text{def}}{=} & \mathsf{fv}(t) \setminus \{x\}
\end{array}$$

El conjunto de *locaciones* de un término ($\mathsf{locs}(t)$) y de un programa ($\mathsf{locs}(P)$) se definen recursivamente de la siguiente forma:

$$\begin{array}{rcl}
\mathsf{locs}(x) & \stackrel{\text{def}}{=} & \varnothing \\
\mathsf{locs}(\mathbf{c}) & \stackrel{\text{def}}{=} & \varnothing \\
\mathsf{locs}(\lambda x.\, P) & \stackrel{\text{def}}{=} & \mathsf{locs}(P) \\
\mathsf{locs}(\lambda^{\ell} x.\, P) & \stackrel{\text{def}}{=} & \{\ell\} \cup \mathsf{locs}(P) \qquad \mathsf{locs}(\mathtt{fail}) \stackrel{\text{def}}{=} \varnothing \\
\mathsf{locs}(t\, s) & \stackrel{\text{def}}{=} & \mathsf{locs}(t) \cup \mathsf{locs}(s) \quad \mathsf{locs}(t \oplus P) \stackrel{\text{def}}{=} \mathsf{locs}(t) \cup \mathsf{locs}(P) \\
\mathsf{locs}(t \stackrel{\bullet}{=} s) & \stackrel{\text{def}}{=} & \mathsf{locs}(t) \cup \mathsf{locs}(s) \\
\mathsf{locs}(t; s) & \stackrel{\text{def}}{=} & \mathsf{locs}(t) \cup \mathsf{locs}(s) \\
\mathsf{locs}(\nu x.\, t) & \stackrel{\text{def}}{=} & \mathsf{locs}(t)
\end{array}$$

Escribimos $t_1 \oplus t_2 \oplus \ldots \oplus t_n$ para denotar el programa $t_1 \oplus (t_2 \oplus \ldots \oplus (t_n \oplus \mathtt{fail}))$. En particular, si t es un término, puede representar el programa unitario $t \oplus \mathtt{fail}$.

Las reglas de reducción están dadas entre programas. El programa a ser evaluado se llama *programa principal*, y siempre es de la forma $t_1 \oplus t_2 \oplus \ldots \oplus t_n$ donde cada t_i se llama *proceso*.

Definición 2.1.3 (Operaciones auxiliares)**.**

1. Si W es un contexto de evaluación débil y t es un término, entonces $\mathsf{W}\langle t\rangle$ es un término definido como sigue, por inducción en W:

$$\begin{array}{rcl}
\Box\langle t\rangle & \stackrel{\text{def}}{=} & t \\
(\mathsf{W}\, s)\langle t\rangle & \stackrel{\text{def}}{=} & \mathsf{W}\langle t\rangle\, s \\
(s\, \mathsf{W})\langle t\rangle & \stackrel{\text{def}}{=} & s\, \mathsf{W}\langle t\rangle \\
(\mathsf{W} \stackrel{\bullet}{=} s)\langle t\rangle & \stackrel{\text{def}}{=} & \mathsf{W}\langle t\rangle \stackrel{\bullet}{=} s \\
(s \stackrel{\bullet}{=} \mathsf{W})\langle t\rangle & \stackrel{\text{def}}{=} & s \stackrel{\bullet}{=} \mathsf{W}\langle t\rangle \\
(\mathsf{W}; s)\langle t\rangle & \stackrel{\text{def}}{=} & \mathsf{W}\langle t\rangle; s \\
(s; \mathsf{W})\langle t\rangle & \stackrel{\text{def}}{=} & s; \mathsf{W}\langle t\rangle
\end{array}$$

2. Si W es un contexto de evaluación débil y P un programa, entonces $\mathsf{W}\langle P\rangle$ es un programa definido como sigue, por inducción en P:

$$\begin{array}{rcl} \mathsf{W}\langle \mathtt{fail}\rangle & \stackrel{\text{def}}{=} & \mathtt{fail} \\ \mathsf{W}\langle t \oplus P\rangle & \stackrel{\text{def}}{=} & \mathsf{W}\langle t\rangle \oplus \mathsf{W}\langle P\rangle \end{array}$$

3. Si P y Q son programas, su concatenación $P \oplus Q$ se define de la siguiente forma, por inducción en P:

$$\begin{array}{rcl} \mathtt{fail} \oplus Q & \stackrel{\text{def}}{=} & Q \\ (t \oplus P) \oplus Q & \stackrel{\text{def}}{=} & t \oplus (P \oplus Q) \end{array}$$

Definición 2.1.4 (Sustituciones)**.** Una *sustitución* es una función $\sigma : \mathsf{Var} \to \mathsf{Val}$ con *soporte finito*, es decir, tal que el soporte $\mathrm{supp}(\sigma) \stackrel{\text{def}}{=} \{x \mid \sigma(x) \neq x\}$ tiene una cantidad finita de elementos. Escribimos $\{x_1 \mapsto \mathtt{v}_1, \dots, x_n \mapsto \mathtt{v}_n\}$ para la sustitución σ tal que $\mathrm{supp}(\sigma) = \{x_1, \dots, x_n\}$ y $\sigma(x_i) = \mathtt{v}_i$ para todo $i \in 1..n$. Un *renombre* es una sustitución biyectiva que asigna cada variable a otra, es decir, una sustitución de la forma $\{x_1 \mapsto y_1, \dots, x_n \mapsto y_n\}$, donde $y_1, \dots, y_n$ es una permutación de $x_1, \dots, x_n$. Por otra parte, $t\{x := \mathtt{v}\}$ denota $t^{\{x \mapsto \mathtt{v}\}}$.

Definición 2.1.5 (Operaciones con sustituciones)**.** Sea $\sigma : \mathsf{Var} \to \mathsf{Val}$ una sustitución cualquiera. La operación para aplicarle la sustitución σ a un término (resp. programa) evitando captura de variables libres se nota t^σ (resp. P^σ) y se define de la siguiente forma:

$$\begin{array}{rcll} x^\sigma & \stackrel{\text{def}}{=} & \sigma(x) & \\ \mathbf{c}^\sigma & \stackrel{\text{def}}{=} & \mathbf{c} & \\ (\lambda x.\, P)^\sigma & \stackrel{\text{def}}{=} & \lambda x.\, P^\sigma & \text{si no hay captura, } \textit{i.e. } \forall y \in \mathrm{supp}(\sigma).\ x \notin \{y\} \cup \mathsf{fv}(\sigma(y)) \\ (\lambda^\ell x.\, P)^\sigma & \stackrel{\text{def}}{=} & \lambda^\ell x.\, P^\sigma & \text{si no hay captura, } \textit{i.e. } \forall y \in \mathrm{supp}(\sigma).\ x \notin \{y\} \cup \mathsf{fv}(\sigma(y)) \\ (\nu x.\, t)^\sigma & \stackrel{\text{def}}{=} & \nu x.\, t^\sigma & \text{si no hay captura, } \textit{i.e. } \forall y \in \mathrm{supp}(\sigma).\ x \notin \{y\} \cup \mathsf{fv}(\sigma(y)) \\ (t\, s)^\sigma & \stackrel{\text{def}}{=} & t^\sigma\, s^\sigma & \\ (t; s)^\sigma & \stackrel{\text{def}}{=} & t^\sigma; s^\sigma & \\ (t \stackrel{\bullet}{=} s)^\sigma & \stackrel{\text{def}}{=} & t^\sigma \stackrel{\bullet}{=} s^\sigma & \\ & & & \\ \mathtt{fail}^\sigma & \stackrel{\text{def}}{=} & \mathtt{fail} & \\ (t \oplus P)^\sigma & \stackrel{\text{def}}{=} & t^\sigma \oplus P^\sigma & \end{array}$$

Las sustituciones también pueden aplicarse a contextos débiles, tomando $\Box^\sigma \stackrel{\text{def}}{=} \Box$.
La *composición* de σ con otra sustitución ρ se nota $\rho \cdot \sigma$, y se define como:

$$(\rho \cdot \sigma)(x) \stackrel{\text{def}}{=} \rho(x)^\sigma$$

Importante: la composición de sustituciones se escribe al revés de la notación usual. Por este motivo, utilizamos $\cdot$ para notar esta operación en lugar del símbolo de composición de funciones, $\circ$.

La sustitución σ es *idempotente* si $\sigma \cdot \sigma = \sigma$. La sustitución σ es *más general* que una sustitución ρ, escrita como $\sigma \lesssim \rho$ si hay una sustitución τ tal que $\rho = \sigma \cdot \tau$.

2.2. Algoritmo de unificación

Definición 2.2.1 (Ecuaciones y unificadores)**.** Una *ecuación* es un término de la forma $\mathtt{v} \overset{\bullet}{=} \mathtt{w}$. Un *problema de unificación* está dado por un conjunto finito de ecuaciones $\mathcal{E} = \{\mathtt{v}_1 \overset{\bullet}{=} \mathtt{w}_1, \dots, \mathtt{v}_n \overset{\bullet}{=} \mathtt{w}_n\}$. Si σ es una sustitución, escribimos $\mathcal{E}^\sigma$ para $\{\mathtt{v}_1{}^\sigma \overset{\bullet}{=} \mathtt{w}_1{}^\sigma, \dots, \mathtt{v}_n{}^\sigma \overset{\bullet}{=} \mathtt{w}_n{}^\sigma\}$. Un *unificador* para $\mathcal{E}$ es una sustitución σ tal que $\mathtt{v}_i{}^\sigma = \mathtt{w}_i{}^\sigma$ para todo $1 \leq i \leq n$. Un unificador σ para $\mathcal{E}$ es el *más general* si para cualquier otro unificador ρ se tiene $\sigma \lesssim \rho$.

Extendemos las nociones de variables libres ($\mathsf{fv}(\mathcal{E})$), locaciones ($\mathsf{locs}(\mathcal{E})$), y sustitución sin captura de variables ($\mathcal{E}^\sigma$) para ecuaciones como sigue:

$$\begin{array}{rcl} \mathsf{fv}(\{\mathtt{v}_1 \overset{\bullet}{=} \mathtt{w}_1, \dots, \mathtt{v}_n \overset{\bullet}{=} \mathtt{w}_n\}) & \overset{\text{def}}{=} & \mathsf{fv}(\mathtt{v}_1 \overset{\bullet}{=} \mathtt{w}_1) \cup \ldots \cup \mathsf{fv}(\mathtt{v}_n \overset{\bullet}{=} \mathtt{w}_n) \\ \mathsf{locs}(\{\mathtt{v}_1 \overset{\bullet}{=} \mathtt{w}_1, \dots, \mathtt{v}_n \overset{\bullet}{=} \mathtt{w}_n\}) & \overset{\text{def}}{=} & \mathsf{locs}(\mathtt{v}_1 \overset{\bullet}{=} \mathtt{w}_1) \cup \ldots \cup \mathsf{locs}(\mathtt{v}_n \overset{\bullet}{=} \mathtt{w}_n) \end{array}$$

Definición 2.2.2. Dado un conjunto X, una *relación de reducción* sobre X es una relación binaria $R \subseteq X \times X$. Notamos $x \to_R y$ si $(x, y) \in R$ es un par en la relación.

Una relación de reducción R es *fuertemente normalizante* si no existe una sucesión infinita $x_1, x_2, x_3, \dots$ de elementos de X tal que $x_1 \to_R x_2 \to_R x_3 \to_R \dots$.

Definición 2.2.3 (Algoritmo de unificación)**.** El siguiente algoritmo es una variante del algoritmo de unificación de Martelli–Montanari. Decimos que dos valores $\mathtt{v}, \mathtt{w}$ *colisionan* si vale cualquiera de las siguientes condiciones:

1. Colisión de constructores: $\mathtt{v} = \mathbf{c}\,\mathtt{v}_1 \dots \mathtt{v}_n$ y $\mathtt{w} = \mathbf{d}\,\mathtt{w}_1 \dots \mathtt{w}_m$ con $\mathbf{c} \neq \mathbf{d}$.
2. Colisión de aridad: $\mathtt{v} = \mathbf{c}\,\mathtt{v}_1 \dots \mathtt{v}_n$ y $\mathtt{w} = \mathbf{c}\,\mathtt{w}_1 \dots \mathtt{w}_m$ con $n \neq m$.
3. Colisión de locaciones: $\mathtt{v} = \lambda^\ell x.\, P$ y $\mathtt{w} = \lambda^{\ell'} y.\, Q$ con $\ell \neq \ell'$.
4. Colisión de tipos: $\mathtt{v} = \mathbf{c}\,\mathtt{v}_1 \dots \mathtt{v}_n$ y $\mathtt{w} = \lambda^\ell x.\, P$, o viceversa.

Definimos un sistema de reescritura cuyos objetos son problemas de unificación $\mathcal{E}$ y el símbolo FAIL. La relación binaria de reescritura $\rightsquigarrow$ está dada por la unión de las siguientes reglas. Notar que "$\uplus$" representa la unión disjunta de conjuntos:

$$\begin{array}{rlll} \{x \overset{\bullet}{=} x\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-delete}} & \mathcal{E} & \\ \{\mathtt{v} \overset{\bullet}{=} x\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-orient}} & \{x \overset{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E} & \text{si } \mathtt{v} \notin \mathsf{Var} \\ \{\lambda^\ell x.\, P \overset{\bullet}{=} \lambda^\ell x.\, P\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-match-lam}} & \mathcal{E} & \\ \{\mathbf{c}\,\mathtt{v}_1 \dots \mathtt{v}_n \overset{\bullet}{=} \mathbf{c}\,\mathtt{w}_1 \dots \mathtt{w}_n\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-match-cons}} & \{\mathtt{v}_1 \overset{\bullet}{=} \mathtt{w}_1, \dots, \mathtt{v}_n \overset{\bullet}{=} \mathtt{w}_n\} \uplus \mathcal{E} & \\ \{\mathtt{v} \overset{\bullet}{=} \mathtt{w}\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-clash}} & \text{FAIL} & \text{si } \mathtt{v} \text{ y } \mathtt{w} \text{ colisionan} \\ \{x \overset{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-eliminate}} & \{x \overset{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E}^{\{x:=\mathtt{v}\}} & \text{si } x \in \mathsf{fv}(\mathcal{E}) \setminus \mathsf{fv}(\mathtt{v}) \\ \{x \overset{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-occurs-check}} & \text{FAIL} & \text{si } x \neq \mathtt{v} \text{ y } x \in \mathsf{fv}(\mathtt{v}) \end{array}$$

Definición 2.2.4 (Coherencia de locación)**.** Sea C un contexto de evaluación. Un conjunto de términos X es *coherente con respecto a locaciones* si valen las siguientes dos condiciones:

1. Si $t \in X$ con $t = \mathsf{C}\langle \lambda^\ell x.\, P\rangle$, *i.e.* considerar cualquier abstracción alojada bajo cualquier contexto C. Entonces C no liga a ninguna variable libre de $\lambda^\ell x.\, P$.
2. Si $t, s \in X$ con $t = \mathsf{C}\langle \lambda^\ell x.\, P\rangle$ y $s = \mathsf{C}'\langle \lambda^\ell y.\, Q\rangle$, *i.e.* considerar dos abstracciones alojadas cualesquiera en t y en s que tengan la misma locación. Entonces vale $P\{x := y\} = Q$.

Además:

1. Un problema de unificación $\mathcal{E}$ es *coherente* si lo es visto como un conjunto.
2. Un término t es *coherente* si $\{t\}$ lo es.
3. Un programa $P = t_1 \oplus \ldots \oplus t_n$ es *coherente* si cualquier proceso t_i lo es.

Teorema 2.2.5 (Cómputo de unificadores más generales)**.** *Considerar la relación restringida a problemas de unificación que sean* coherentes con la locación*. Entonces:*

1. La relación $\rightsquigarrow$ es fuertemente normalizante.

2. Las formas normales de $\rightsquigarrow$ son FAIL *y conjuntos de ecuaciones de la forma $\{x_1 \stackrel{\bullet}{=} \mathtt{v}_1, \ldots, x_n \stackrel{\bullet}{=} \mathtt{v}_n\}$ donde $x_i \neq x_j$ y $x_i \notin \mathsf{fv}(\mathtt{v}_j)$ para cada $i, j \in 1..n$.*

Si la forma normal de $\mathcal{E}$ es $\{x_1 \stackrel{\bullet}{=} \mathtt{v}_1, \ldots, x_n \stackrel{\bullet}{=} \mathtt{v}_n\}$, decimos que el $\mathsf{mgu}(\mathcal{E})$ existe, y $\mathsf{mgu}(\mathcal{E}) = \{x_1 \mapsto \mathtt{v}_1, \ldots, x_n \mapsto \mathtt{v}_n\}$. Si la forma normal es FAIL*, entonces decimos que $\mathsf{mgu}(\mathcal{E})$ falla.*

3. La sustitución $\sigma = \mathsf{mgu}(\mathcal{E})$ existe si y sólo si existe un unificador para $\mathcal{E}$. Cuando existe, $\mathsf{mgu}(\mathcal{E})$ es un unificador más general idempotente*. Además:*

3.1 El conjunto $\mathcal{E}^\sigma \cup \{\sigma(x) \mid x \in \mathsf{Var}\}$ es coherente.

3.2 Para cualquier $x \in \mathsf{Var}$ y cualquier abstracción alojada $\lambda^\ell y. P$ en $\sigma(x)$, la locación ℓ decora una abstracción alojada en $\mathcal{E}$.

Demostración. La demostración de este teorema no forma parte del presente trabajo, se encuentra en [3]. □

2.2.1. Ejemplos

Ejemplo 2.2.6. *Sea el conjunto de ecuaciones $\{\lambda^\ell x. \mathbf{c} \stackrel{\bullet}{=} x, \mathbf{d}\, x\, (\lambda^{\ell'} z. z) \stackrel{\bullet}{=} \mathbf{d}\, y\, (\lambda^{\ell'} z. z)\}$, aplicando el algoritmo de unificación obtenemos:*

$$
\begin{array}{ll}
 & \{\lambda^\ell x. \mathbf{c} \stackrel{\bullet}{=} x, \mathbf{d}\, x\, \lambda^{\ell'} z. z \stackrel{\bullet}{=} \mathbf{d}\, y\, \lambda^{\ell'} z. z\} \\
\rightsquigarrow_{\texttt{u-orient}} & \{x \stackrel{\bullet}{=} \lambda^\ell x. \mathbf{c}, \mathbf{d}\, x\, \lambda^{\ell'} z. z \stackrel{\bullet}{=} \mathbf{d}\, y\, \lambda^{\ell'} z. z\} \\
\rightsquigarrow_{\texttt{u-match-cons}} & \{x \stackrel{\bullet}{=} \lambda^\ell x. \mathbf{c}, x \stackrel{\bullet}{=} y, \lambda^{\ell'} z. z \stackrel{\bullet}{=} \lambda^{\ell'} z. z\} \\
\rightsquigarrow_{\texttt{u-eliminate}} & \{x \stackrel{\bullet}{=} \lambda^\ell x. \mathbf{c}, \lambda^\ell x. \mathbf{c} \stackrel{\bullet}{=} y, \lambda^{\ell'} z. z \stackrel{\bullet}{=} \lambda^{\ell'} z. z\} \\
\rightsquigarrow_{\texttt{u-orient}} & \{x \stackrel{\bullet}{=} \lambda^\ell x. \mathbf{c}, y \stackrel{\bullet}{=} \lambda^\ell x. \mathbf{c}, \lambda^{\ell'} z. z \stackrel{\bullet}{=} \lambda^{\ell'} z. z\} \\
\rightsquigarrow_{\texttt{u-match-lam}} & \{x \stackrel{\bullet}{=} \lambda^\ell x. \mathbf{c}, y \stackrel{\bullet}{=} \lambda^\ell x. \mathbf{c}\}
\end{array}
$$

El algoritmo de unificación termina, y retorna $\mathsf{mgu}(\{\lambda^\ell x. \mathbf{c} \stackrel{\bullet}{=} x, \mathbf{d}\, x\, \lambda^{\ell'} z. z \stackrel{\bullet}{=} \mathbf{d}\, y\, \lambda^{\ell'} z. z\}) = \{x \mapsto \lambda^\ell x. \mathbf{c}, y \mapsto \lambda^\ell x. \mathbf{c}\}$

Ejemplo 2.2.7. *Sea el conjunto de ecuaciones $\{\mathbf{c}\, \lambda^\ell x. x\, y \stackrel{\bullet}{=} \mathbf{c}\, y\}$, aplicando el algoritmo de unificación obtenemos:*

$$
\begin{array}{ll}
 & \{\mathbf{c}\, \lambda^\ell x. x\, y \stackrel{\bullet}{=} \mathbf{c}\, y\} \\
\rightsquigarrow_{\texttt{u-match-cons}} & \{\lambda^\ell x. x\, y \stackrel{\bullet}{=} y\} \\
\rightsquigarrow_{\texttt{u-orient}} & \{y \stackrel{\bullet}{=} \lambda^\ell x. x\, y\} \\
\rightsquigarrow_{\texttt{u-occurs-check}} & \text{FAIL}
\end{array}
$$

2.3. Semántica operacional del cálculo-$\lambda^{\mathtt{U}}$

Definición 2.3.1 (Reglas de reducción)**.** El cálculo-$\lambda^{\mathtt{U}}$ es el sistema de reescritura cuyos objetos son programas que cumplen el invariante de coherencia, y cuyas reglas de reescritura están dadas por:

1. `alloc`:
$$P_1 \oplus \mathsf{W}\langle \lambda x.\, Q\rangle \oplus P_2 \to P_1 \oplus \mathsf{W}\langle \lambda^{\ell} x.\, Q\rangle \oplus P_2$$
donde ℓ es una locación fresca.

2. `beta`:
$$P_1 \oplus \mathsf{W}\langle (\lambda^{\ell} x.\, Q)\, \mathtt{v}\rangle \oplus P_2 \to P_1 \oplus \mathsf{W}\langle Q\{x := \mathtt{v}\}\rangle \oplus P_2$$

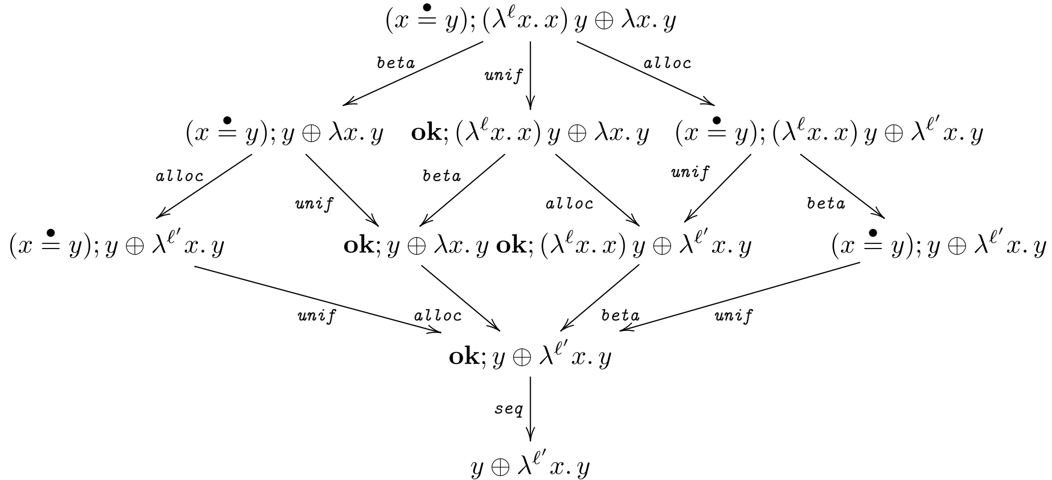
3. `seq`:
$$P_1 \oplus \mathsf{W}\langle \mathtt{v}; t\rangle \oplus P_2 \to P_1 \oplus \mathsf{W}\langle t\rangle \oplus P_2$$

4. `fresh`:
$$P_1 \oplus \mathsf{W}\langle \nu x.\, t\rangle \oplus P_2 \to P_1 \oplus \mathsf{W}\langle t\{x := y\}\rangle \oplus P_2$$
donde y es una variable fresca.

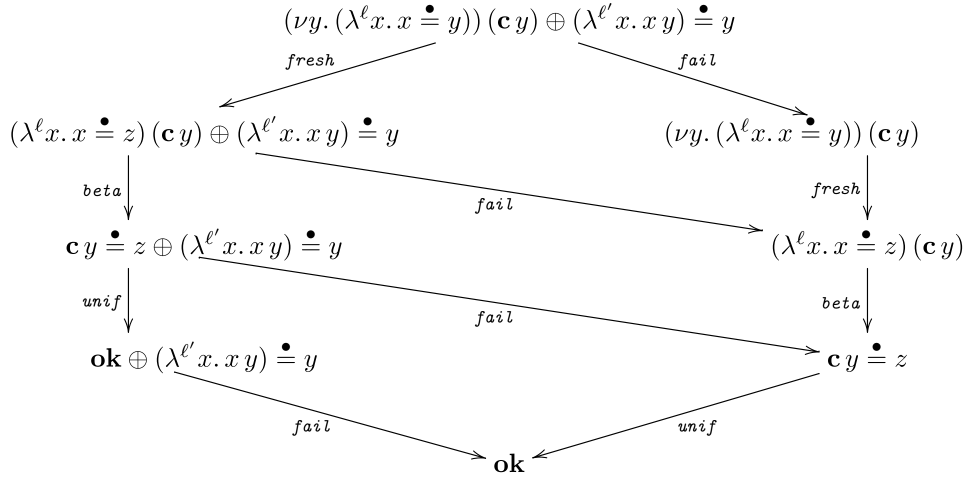
5. `unif`:
$$P_1 \oplus \mathsf{W}\langle \mathtt{v} \overset{\bullet}{=} \mathtt{w}\rangle \oplus P_2 \to P_1 \oplus \mathsf{W}\langle \mathbf{ok}\rangle^{\sigma} \oplus P_2$$
donde $\sigma = \mathsf{mgu}(\{\mathtt{v} \overset{\bullet}{=} \mathtt{w}\})$.

6. `fail`:
$$P_1 \oplus \mathsf{W}\langle \mathtt{v} \overset{\bullet}{=} \mathtt{w}\rangle \oplus P_2 \to P_1 \oplus P_2$$
si $\mathsf{mgu}(\{\mathtt{v} \overset{\bullet}{=} \mathtt{w}\})$ falla.

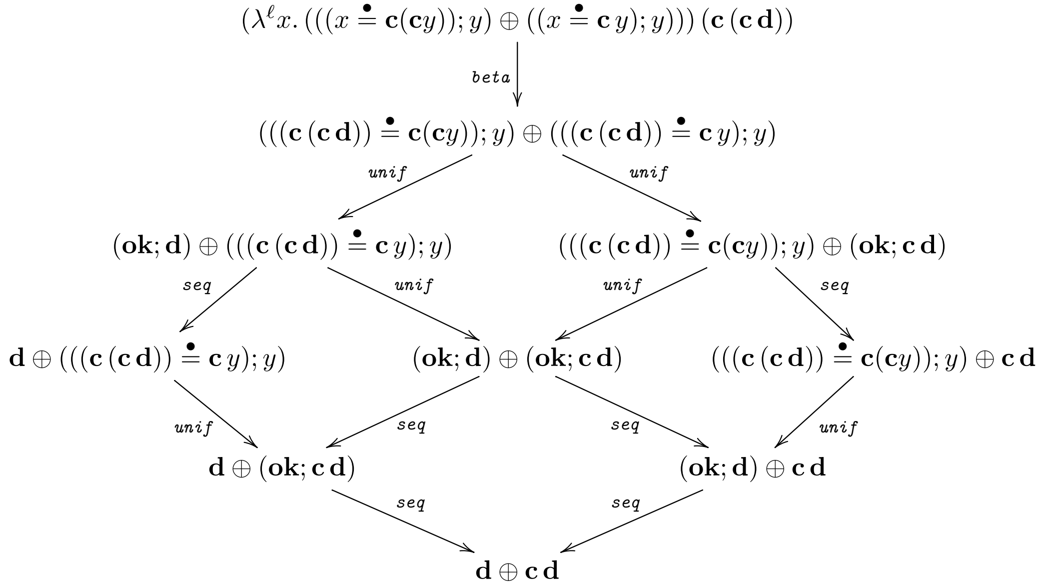
Ejemplo 2.3.2. *Tomemos el término* $(x \overset{\bullet}{=} y); (\lambda^{\ell} x.\, x)\, y \oplus \lambda x.\, y$. *Las formas en que se puede reducir son:*



Ejemplo 2.3.3. *Tomemos el término* $(\nu y.\,(\lambda^{\ell} x.\, x \overset{\bullet}{=} y))\,(\mathbf{c}\, y) \oplus (\lambda^{\ell'} x.\, x\, y) \overset{\bullet}{=} y$*, que se puede reducir de las siguientes maneras:*

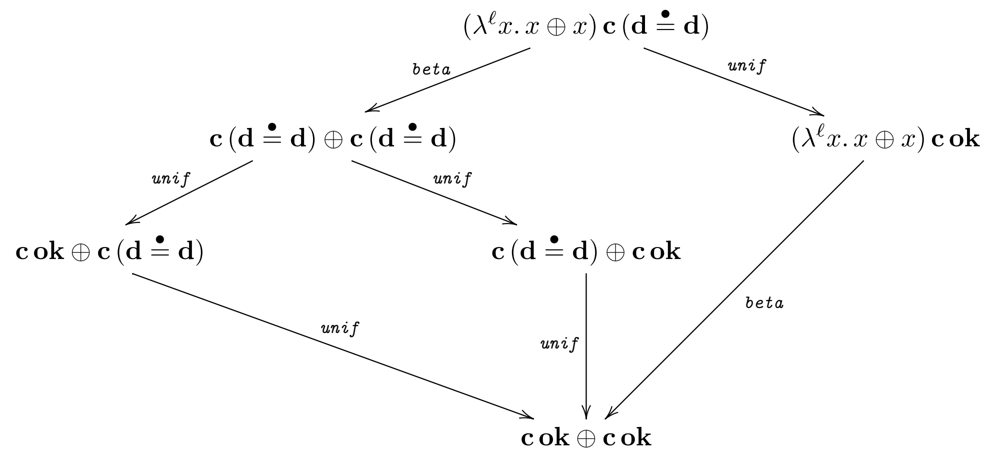


Ejemplo 2.3.4. *Tomemos el término* $(\lambda^{\ell} x.\,(((x \overset{\bullet}{=} \mathbf{c}(\mathbf{c} y)); y) \oplus ((x \overset{\bullet}{=} \mathbf{c}\, y); y)))\,(\mathbf{c}\,(\mathbf{c}\,\mathbf{d}))$*, que se puede reducir de las siguientes maneras:*



Ejemplo 2.3.5. *Tomemos el término* $(\lambda^{\ell} x.\, x \oplus x)\,\mathbf{c}\,(\mathbf{d} \overset{\bullet}{=} \mathbf{d})$*, que se puede reducir de las*

siguientes maneras:



3. TIPADO DEL CÁLCULO-$\lambda^{\mathtt{U}}$

En este capítulo damos un sistema de tipos para el cálculo-$\lambda^{\mathtt{U}}$, y demostramos las propiedades de Debilitamiento, Fortalecimiento y Preservación de tipos.

Definición 3.0.1 (Tipos y contextos de tipado)**.** Supongamos dado un conjunto infinito numerable de *tipos base* $\alpha, \beta, \gamma, \ldots$. El conjunto de *tipos* (Type) está dado por la siguiente gramática:

$$A ::= \alpha \mid A \to A$$

Un *contexto de tipado* $(\Gamma, \Delta, \ldots)$ es un conjunto finito de asignaciones de variables a tipos, cada asignación es de la forma $x : A$. Asumimos que cada constructor $\mathbf{c}$ tiene un tipo asociado $\mathcal{T}_{\mathbf{c}}$.

En este trabajo formulamos un sistema con tipado intrínseco (*à la* Church). En particular, suponemos que las ocurrencias de variables, el término `fail`, y las variables ligadas por abstracciones e introducciones de variables simbólicas se encuentran decoradas con su tipo. Es decir, los términos pasan a ser:

$$\begin{array}{rcll}
t & ::= & x^A & \text{variable} \\
& \mid & \mathbf{c} & \text{constructor} \\
& \mid & \lambda x^A.\, P & \text{abstracción} \\
& \mid & \lambda^{\ell} x^A.\, P & \text{abstracción alojada} \\
& \mid & t\, s & \text{aplicación} \\
& \mid & t \overset{\bullet}{=} s & \text{unificación} \\
& \mid & t; s & \text{secuencia} \\
& \mid & \nu x^A.\, t & \text{declaración de variable fresca} \\
P & ::= & \mathtt{fail}^A & \text{programa vacío} \\
& \mid & t \oplus P & \text{alternativa no determinística}
\end{array}$$

Generalmente omitimos la decoración de tipos si puede deducirse del contexto o no es especialmente relevante.

Definición 3.0.2 (Sistema de tipos)**.** Hay tres formas de juicios:

1. $\Gamma \vdash t : A$ que significa que el término t tiene tipo A bajo el contexto de tipado Γ;

2. $\Gamma \vdash P : A$ que significa que el programa P tiene tipo A bajo el contexto de tipado Γ;

3. $\rhd P : A$ que significa que P es un programa principal de tipo A.

Las derivaciones de juicios están dadas por las siguientes reglas de tipado:

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x^A : A}\,\texttt{t-var} \qquad \frac{}{\Gamma \vdash \mathbf{c} : \mathcal{T}_{\mathbf{c}}}\,\texttt{t-cons}$$

$$\frac{\Gamma, x : A \vdash P : B}{\Gamma \vdash \lambda x^A.\, P : A \to B}\,\texttt{t-lam} \qquad \frac{\Gamma, x : A \vdash P : B}{\Gamma \vdash \lambda^{\ell} x^A.\, P : A \to B}\,\texttt{t-laml}$$

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash s : A}{\Gamma \vdash t\, s : B}\,\texttt{t-app} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash s : A}{\Gamma \vdash t \overset{\bullet}{=} s : \mathcal{T}_{\mathbf{ok}}}\,\texttt{t-unif}$$

$$\frac{\Gamma \vdash t : \mathcal{T}_{\mathbf{ok}} \quad \Gamma \vdash s : A}{\Gamma \vdash t ; s : A}\,\texttt{t-seq} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \nu x^{A}. t : B}\,\texttt{t-fresh}$$

$$\frac{}{\Gamma \vdash \mathtt{fail}^{A} : A}\,\texttt{t-fail} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash P : A}{\Gamma \vdash t \oplus P : A}\,\texttt{t-alt}$$

$$\frac{\Gamma \vdash P : A}{\rhd P : A}\,\texttt{t-prog}$$

3.1. Propiedades básicas del sistema de tipos

Lema 3.1.1 (Debilitamiento).

1. *Si* $\Gamma \vdash t : A$ *entonces* $\Gamma, z : C \vdash t : A$.
2. *Si* $\Gamma \vdash P : A$ *entonces* $\Gamma, z : C \vdash P : A$.

Demostración. Por inducción mutua en la derivación del juicio $\Gamma \vdash t : A$ (resp. $\Gamma \vdash P : A$).

1. `t-var`: Supongamos que $\Gamma \vdash x : A$ se deriva usando la regla `t-var`, *i.e.* $(x : A) \in \Gamma$. Como $\Gamma \subseteq \Gamma, z : C$ entonces también $(x : A) \in \Gamma, z : C$. Aplicando la regla `t-var` tenemos que $\Gamma, z : C \vdash x : A$, como se pide.

2. `t-cons`: Inmediato, pues $\Gamma, z : C \vdash \mathbf{c} : \mathcal{T}_{\mathbf{c}}$ puede derivarse usando la regla `t-cons`.

3. `t-lam`: Supongamos que $\Gamma \vdash \lambda x. P : A \to B$ se deriva de $\Gamma, x : A \vdash P : B$. Entonces, por *h.i.* tenemos que $\Gamma, x : A, z : C \vdash P : B$. Aplicando la regla `t-lam` sobre este último juicio llegamos a que $\Gamma, z : C \vdash \lambda x. P : A \to B$, como requiere el enunciado.

4. `t-laml`: Supongamos que $\Gamma \vdash \lambda^{\ell} x. P : A \to B$ se deriva de $\Gamma, x : A \vdash P : B$. Entonces, por *h.i.* tenemos que $\Gamma, x : A, z : C \vdash P : B$. Aplicando la regla `t-laml` sobre este juicio llegamos a que $\Gamma, z : C \vdash \lambda^{\ell} x. P : A \to B$, como se pide.

5. `t-app`: Supongamos que $\Gamma \vdash t\, s : B$ se deriva de $\Gamma \vdash t : A \to B$ y de $\Gamma \vdash s : A$. Entonces, por *h.i.* tenemos que $\Gamma, z : C \vdash t : A \to B$ y $\Gamma, z : C \vdash s : A$. Aplicando la regla `t-app` llegamos a que $\Gamma, z : C \vdash t\, s : B$, como requiere el enunciado.

6. `t-unif`: Supongamos que $\Gamma \vdash t \doteq s : \mathcal{T}_{\mathbf{ok}}$ se deriva de $\Gamma \vdash t : A$ y de $\Gamma \vdash s : A$. Entonces, por *h.i.* tenemos que $\Gamma, z : C \vdash t : A$ y $\Gamma, z : C \vdash s : A$. Aplicando la regla `t-unif` llegamos a que $\Gamma, z : C \vdash t \doteq s : \mathcal{T}_{\mathbf{ok}}$, como se pide.

7. `t-seq`: Supongamos que $\Gamma \vdash t; s : A$ se deriva de $\Gamma \vdash t : \mathcal{T}_{\mathbf{c}}$ y de $\Gamma \vdash s : A$. Entonces, por *h.i.* tenemos que $\Gamma, z : C \vdash t : \mathcal{T}_{\mathbf{c}}$ y $\Gamma, z : C \vdash s : A$. Aplicando la regla `t-seq` llegamos a que $\Gamma, z : C \vdash t; s : A$, como requiere el enunciado.

8. `t-fresh`: Supongamos que $\Gamma \vdash \nu x. t : B$ se deriva de $\Gamma, x : A \vdash t : B$. Entonces, por *h.i.* tenemos que $\Gamma, x : A, z : C \vdash t : B$. Aplicando la regla `t-fresh` obtenemos $\Gamma, z : C \vdash \nu x. t : B$, como se pide.

9. `t-fail`: Inmediato, pudiendo derivarse $\Gamma, z : C \vdash \mathtt{fail} : A$ aplicando la regla `fail`.

10. `t-alt`: Supongamos que $\Gamma \vdash t \oplus P : A$ se deriva de $\Gamma \vdash t : A$ y de $\Gamma \vdash P : A$. Entonces, por *h.i.* tenemos que $\Gamma, z : C \vdash t : A$ y $\Gamma, z : C \vdash P : A$. Aplicando la regla `t-alt`, obtenemos que $\Gamma, z : C \vdash t \oplus P : A$, como se pide.

□

Lema 3.1.2 (Fortalecimiento)**.**

1. *Sea* $\Gamma, x : A \vdash t : B$ *y supongamos que* $x \notin \mathsf{fv}(t)$*. Entonces,* $\Gamma \vdash t : B$*.*
2. *Sea* $\Gamma, x : A \vdash P : B$ *y supongamos que* $x \notin \mathsf{fv}(P)$*. Entonces,* $\Gamma \vdash P : B$*.*

Demostración. Por inducción mutua en la derivación del juicio $\Gamma, x : A \vdash t : B$ (resp. $\Gamma, x : A \vdash P : B$).

1. `t-var`: Tenemos que $\Gamma, x : A \vdash y : B$ y como $x \notin \mathsf{fv}(y)$, entonces $x \neq y$. Además, como hipótesis tenemos que $(y : B) \in \Gamma, x : A$, por lo que podemos decir que $(y : B) \in \Gamma$. Aplicando la regla `t-var`, obtenemos que $\Gamma \vdash y : B$, como se pide.
2. `t-cons`: Tenemos que $\Gamma, x : A \vdash \mathbf{c} : \mathcal{T}_{\mathbf{c}}$. Aplicando la regla `t-cons`, obtenemos $\Gamma \vdash \mathbf{c} : \mathcal{T}_{\mathbf{c}}$.
3. `t-lam`: Supongamos que $\Gamma, x : A \vdash \lambda y. P : B \to C$ se deriva de $\Gamma, x : A, y : B \vdash P : C$. Entonces, por *h.i.* tenemos que $\Gamma, y : B \vdash P : C$. Aplicando la regla `t-lam` podemos concluir que $\Gamma \vdash \lambda y. P : B \to C$, como se pide.
4. `t-laml`: Supongamos que $\Gamma, x : A \vdash \lambda^{\ell} y. P : B \to C$ se deriva de $\Gamma, x : A, y : B \vdash P : C$. Entonces, por *h.i.* tenemos que $\Gamma, y : B \vdash P : C$. Aplicando la regla `t-laml` podemos concluir que $\Gamma \vdash \lambda^{\ell} y. P : B \to C$, como se pide.
5. `t-app`: Supongamos que $\Gamma, x : A \vdash t_1\, t_2 : C$ se deriva de $\Gamma, x : A \vdash t_1 : B \to C$ y de $\Gamma, x : A \vdash t_2 : B$. Entonces, por *h.i.* tenemos que $\Gamma \vdash t_1 : B \to C$ y que $\Gamma \vdash t_2 : B$. Aplicando la regla `t-app` podemos concluir que $\Gamma \vdash t_1\, t_2 : C$.
6. `t-unif`: Supongamos que $\Gamma, x : A \vdash t_1 \doteq t_2 : \mathcal{T}_{\mathbf{ok}}$ se deriva de $\Gamma, x : A \vdash t_1 : B$ y de $\Gamma, x : A \vdash t_2 : B$. Entonces, por *h.i.* tenemos que $\Gamma \vdash t_1 : B$ y $\Gamma \vdash t_2 : B$. Aplicando la regla `t-unif` podemos concluir que $\Gamma \vdash t_1 \doteq t_2 : \mathcal{T}_{\mathbf{ok}}$.
7. `t-seq`: Supongamos que $\Gamma, x : A \vdash t_1; t_2 : B$ se deriva de $\Gamma, x : A \vdash t_1 : \mathcal{T}_{\mathbf{c}}$ y de $\Gamma, x : A \vdash t_2 : B$. Entonces, por *h.i.* tenemos que $\Gamma \vdash t_1 : \mathcal{T}_{\mathbf{c}}$ y que $\Gamma \vdash t_2 : B$. Aplicando la regla `t-seq` podemos concluir que $\Gamma \vdash t_1; t_2 : B$.
8. `t-fresh`: Supongamos que $\Gamma, x : A \vdash \nu y. t : C$ se deriva de $\Gamma, x : A, y : B \vdash t : C$. Entonces, por *h.i.* tenemos que $\Gamma, y : B \vdash t : C$. Aplicando la regla `t-fresh` podemos concluir que $\Gamma \vdash \nu y. t : C$.
9. `t-fail`: Tenemos que $\Gamma, x : A \vdash \texttt{fail} : B$. Aplicando la regla `t-fail`, llegamos a que $\Gamma \vdash \texttt{fail} : B$, como se pide.
10. `t-alt`: Supongamos que $\Gamma, x : A \vdash t \oplus P : B$ se deriva de $\Gamma, x : A \vdash t : B$ y de $\Gamma, x : A \vdash P : B$. Entonces, por *h.i.* tenemos que $\Gamma \vdash t : B$ y que $\Gamma \vdash P : B$. Aplicando la regla `t-alt` podemos concluir que $\Gamma \vdash t \oplus P : B$.

□

Lema 3.1.3 (Sustitución)**.**

1. *Si* $\Gamma, x : A \vdash t : B$ *y* $\Gamma \vdash s : A$ *entonces* $\Gamma \vdash t\{x := s\} : B$*.*

2. *Si* $\Gamma, x : A \vdash P : B$ *y* $\Gamma \vdash s : A$ *entonces* $\Gamma \vdash P\{x := s\} : B$.

Demostración. Por inducción mutua en la derivación del juicio $\Gamma, x : A \vdash t : A$ (resp. $\Gamma, x : A \vdash P : A$).

1. `t-var`: Hay dos subcasos, dependiendo de si la variable es x o no:

   1.1 Si la variable es x, tenemos que $\Gamma, x : A \vdash x : A$. Como $x\{x := s\} = s$ y $\Gamma \vdash s : A$, entonces $\Gamma \vdash x\{x := s\} : A$, como se pide.

   1.2 En el otro subcaso, tenemos que $\Gamma, x : A \vdash y : B$ con $x \neq y$ y $(y : B) \in \Gamma$. Como $y\{x := s\} = y$, y $(y : B) \in \Gamma$, aplicando la regla `t-var`, tenemos que $\Gamma \vdash y\{x := s\} : B$ como se pide.

2. `t-cons`: Tenemos que $\Gamma, x : A \vdash \mathbf{c} : \mathcal{T}_{\mathbf{c}}$. Como $\mathbf{c}\{x := s\} \stackrel{\text{def}}{=} \mathbf{c}$, entonces aplicando la regla `t-cons`, podemos concluir que $\Gamma \vdash \mathbf{c}\{x := s\} : \mathcal{T}_{\mathbf{c}}$.

3. `t-lam`: Supongamos que $\Gamma, x : A \vdash \lambda y. P : B \to C$ se deriva de $\Gamma, x : A, y : B \vdash P : C$. Entonces, por *h.i.* tenemos que $\Gamma, y : B \vdash P\{x := s\} : C$. Aplicando la regla `t-lam` podemos concluir que $\Gamma \vdash \lambda y. P\{x := s\} : B \to C$, o lo que es equivalente, $\Gamma \vdash (\lambda y. P)\{x := s\} : B \to C$.

4. `t-laml`: Supongamos que $\Gamma, x : A \vdash \lambda^{\ell} y. P : B \to C$ se deriva de $\Gamma, x : A, y : B \vdash P : C$. Entonces, por *h.i.* tenemos que $\Gamma, y : B \vdash P\{x := s\} : C$. Aplicando la regla `t-laml` podemos concluir que $\Gamma \vdash \lambda^{\ell} y. P\{x := s\} : B \to C$, o lo que es equivalente, $\Gamma \vdash (\lambda^{\ell} y. P)\{x := s\} : B \to C$.

5. `t-app`: Supongamos que $\Gamma, x : A \vdash t_1\, t_2 : C$ se deriva de $\Gamma, x : A \vdash t_1 : B \to C$ y de $\Gamma, x : A \vdash t_2 : B$. Entonces, por *h.i.* tenemos que $\Gamma \vdash t_1\{x := s\} : B \to C$ y que $\Gamma \vdash t_2\{x := s\} : B$. Aplicando la regla `t-app` podemos concluir que $\Gamma \vdash t_1\{x := s\}\, t_2\{x := s\} : C$, o lo que es equivalente, $\Gamma \vdash (t_1\, t_2)\{x := s\} : C$.

6. `t-unif`: Supongamos que $\Gamma, x : A \vdash t_1 \doteq t_2 : \mathcal{T}_{\mathbf{ok}}$ se deriva de $\Gamma, x : A \vdash t_1 : B$ y de $\Gamma, x : A \vdash t_2 : B$. Entonces, por *h.i.* tenemos que $\Gamma \vdash t_1\{x := s\} : B$ y que $\Gamma \vdash t_2\{x := s\} : B$. Aplicando la regla `t-unif` podemos concluir que $\Gamma \vdash t_1\{x := s\} \doteq t_2\{x := s\} : \mathcal{T}_{\mathbf{ok}}$, o lo que es equivalente, $\Gamma \vdash (t_1 \doteq t_2)\{x := s\} : \mathcal{T}_{\mathbf{ok}}$.

7. `t-seq`: Supongamos que $\Gamma, x : A \vdash t_1; t_2 : B$ se deriva de $\Gamma, x : A \vdash t_1 : \mathcal{T}_{\mathbf{c}}$ y de $\Gamma, x : A \vdash t_2 : B$. Entonces, por *h.i.* tenemos que $\Gamma \vdash t_1\{x := s\} : \mathcal{T}_{\mathbf{c}}$ y que $\Gamma \vdash t_2\{x := s\} : B$. Aplicando la regla `t-seq` podemos concluir que $\Gamma \vdash t_1\{x := s\}; t_2\{x := s\} : B$, o lo que es equivalente, $\Gamma \vdash (t_1; t_2)\{x := s\} : B$.

8. `t-fresh`: Supongamos que $\Gamma, x : A \vdash \nu y. t : C$ se deriva de $\Gamma, x : A, y : B \vdash t : C$. Entonces, por *h.i.* tenemos que $\Gamma, y : B \vdash t\{x := s\} : C$. Aplicando la regla `t-fresh` podemos concluir que $\Gamma \vdash \nu y. t\{x := s\} : C$, o lo que es equivalente, $\Gamma \vdash (\nu y. t)\{x := s\} : C$.

9. `t-fail`: Tenemos que $\Gamma, x : A \vdash \mathtt{fail} : B$. Como $\mathtt{fail}\{x := s\} \stackrel{\text{def}}{=} \mathtt{fail}$, entonces aplicando la regla `t-fail`, podemos concluir que $\Gamma \vdash \mathtt{fail}\{x := s\} : B$.

10. `t-alt`: Supongamos que $\Gamma, x : A \vdash t \oplus P : B$ se deriva de $\Gamma, x : A \vdash t : B$ y de $\Gamma, x : A \vdash P : B$. Entonces, por *h.i.* tenemos que $\Gamma \vdash t\{x := s\} : B$ y $\Gamma \vdash P\{x := s\} : B$. Aplicando la regla `t-alt` podemos concluir que $\Gamma \vdash t\{x := s\} \oplus P\{x := s\} : B$, o lo que es equivalente, $\Gamma \vdash (t \oplus P)\{x := s\} : B$.

□

Lema 3.1.4 (Sustitución contextual)**.** *Son equivalentes:*

1. *El juicio* $\Gamma \vdash \mathsf{W}\langle t\rangle : A$ *vale.*
2. *Existe un tipo* B *tal que* $\Gamma, \Box : B \vdash \mathsf{W} : A$ *y* $\Gamma \vdash t : B$*.*

Demostración.

- $(1 \implies 2)$ Supongamos que $\Gamma \vdash \mathsf{W}\langle t\rangle : A$. Procedemos por inducción en la estructura del contexto débil W:

  1. *Contexto vacío,* $\mathsf{W} = \Box$. Por hipótesis $\Gamma \vdash t : A$. Además, por la regla `t-var` tenemos que $\Gamma, \Box : A \vdash \Box : A$, con lo que concluimos este ítem.
  2. *Izquierda de una aplicación,* $\mathsf{W} = \mathsf{W}'\, s$. Por hipótesis tenemos que $\Gamma \vdash \mathsf{W}'\langle t\rangle\, s : A$. Este juicio sólo puede derivarse usando la regla `t-app`, por lo que existe un tipo B tal que $\Gamma \vdash \mathsf{W}'\langle t\rangle : B \to A$ y $\Gamma \vdash s : B$. Entonces, por *h.i.* en el contexto más chico W' existe un tipo C tal que $\Gamma, \Box : C \vdash \mathsf{W}' : B \to A$ y $\Gamma \vdash t : C$. Para concluir, notemos que $\Gamma, \Box : C \vdash \mathsf{W}'\, s : A$ aplicando debilitamiento (Lema 3.1.1) y la regla `t-app`.
  3. *Derecha de una aplicación,* $\mathsf{W} = s\, \mathsf{W}'$. Por hipótesis tenemos que $\Gamma \vdash s\, \mathsf{W}'\langle t\rangle : A$. Este juicio sólo puede derivarse usando la regla `t-app`, por lo que existe un tipo B tal que $\Gamma \vdash s : B \to A$ y $\Gamma \vdash \mathsf{W}'\langle t\rangle : B$. Entonces, por *h.i.* en el contexto más chico W' existe un tipo C tal que $\Gamma, \Box : C \vdash \mathsf{W}' : B \to A$ y $\Gamma \vdash t : C$. Aplicando debilitamiento (Lema 3.1.1) y la regla `t-app`, podemos concluir que $\Gamma, \Box : C \vdash \mathsf{W}'\, s : A$.
  4. *Izquierda de una unificación,* $\mathsf{W} = (\mathsf{W}' \stackrel{\bullet}{=} s)$. Por hipótesis tenemos que $\Gamma \vdash \mathsf{W}'\langle t\rangle \stackrel{\bullet}{=} s : \mathcal{T}_{\mathbf{ok}}$. Este juicio sólo puede derivarse usando la regla `t-unif`, por lo que existe un tipo A tal que $\Gamma \vdash \mathsf{W}'\langle t\rangle : A$ y $\Gamma \vdash s : A$. Entonces, por *h.i.* en el contexto más chico W' existe un tipo B tal que $\Gamma, \Box : B \vdash \mathsf{W}' : A$ y $\Gamma \vdash t : B$. Aplicando debilitamiento (Lema 3.1.1) y la regla `t-unif`, podemos concluir que $\Gamma, \Box : B \vdash \mathsf{W}' \stackrel{\bullet}{=} s : \mathcal{T}_{\mathbf{ok}}$.
  5. *Derecha de una unificación,* $\mathsf{W} = (s \stackrel{\bullet}{=} \mathsf{W}')$. Por hipótesis tenemos que $\Gamma \vdash s \stackrel{\bullet}{=} \mathsf{W}'\langle t\rangle : \mathcal{T}_{\mathbf{ok}}$. Este juicio sólo puede derivarse usando la regla `t-unif`, por lo que existe un tipo A tal que $\Gamma \vdash s : A$ y $\Gamma \vdash \mathsf{W}'\langle t\rangle : A$. Entonces, por *h.i.* en el contexto más chico W' existe un tipo B tal que $\Gamma, \Box : B \vdash \mathsf{W}' : A$ y $\Gamma \vdash t : B$. Aplicando debilitamiento (Lema 3.1.1) y la regla `t-unif`, podemos concluir que $\Gamma, \Box : B \vdash s \stackrel{\bullet}{=} \mathsf{W}' : \mathcal{T}_{\mathbf{ok}}$.
  6. *Izquierda de una secuencia,* $\mathsf{W} = \mathsf{W}'; s$. Por hipótesis tenemos que $\Gamma \vdash \mathsf{W}'\langle t\rangle; s : A$, que sólo puede derivarse usando la regla `t-seq`, por lo que tenemos que $\Gamma \vdash \mathsf{W}'\langle t\rangle : \mathcal{T}_{\mathbf{ok}}$, y $\Gamma \vdash s : A$. Entonces, por *h.i.* en el contexto más chico W' existe un tipo B tal que $\Gamma, \Box : B \vdash \mathsf{W}' : \mathcal{T}_{\mathbf{ok}}$ y $\Gamma \vdash t : B$. Aplicando debilitamiento (Lema 3.1.1) y la regla `t-seq`, podemos concluir que $\Gamma, \Box : B \vdash \mathsf{W}'; s : A$.

7. *Derecha de una secuencia,* $\mathsf{W} = s; \mathsf{W}'$. Por hipótesis tenemos que $\Gamma \vdash s; \mathsf{W}'\langle t \rangle : A$, que sólo puede derivarse usando la regla `t-seq`, por lo que tenemos que $\Gamma \vdash s : \mathcal{T}_{\mathbf{ok}}$, y $\Gamma \vdash \mathsf{W}'\langle t \rangle : A$. Entonces, por *h.i.* en el contexto más chico W' existe un tipo B tal que $\Gamma, \Box : B \vdash \mathsf{W}' : A$ y $\Gamma \vdash t : B$. Aplicando debilitamiento (Lema 3.1.1) y la regla `t-seq`, podemos concluir que $\Gamma, \Box : B \vdash s; \mathsf{W}' : A$.

- ($2 \implies 1$) Supongamos que $\Gamma, \Box : B \vdash \mathsf{W} : A$, y $\Gamma \vdash t : B$. Procedemos por inducción en la estructura del contexto débil W.

1. *Vacío,* $\mathsf{W} = \Box$. Por hipótesis, $\Gamma, \Box : B \vdash \Box : A$. Notemos que este juicio sólo puede derivarse usando la regla `t-var`, por lo que necesariamente $A = B$. Además, por hipótesis tenemos que $\Gamma \vdash t : B = A$, como se pide.
2. *Izquierda de una aplicación,* $\mathsf{W} = \mathsf{W}'\, s$. Por hipótesis, $\Gamma, \Box : B \vdash \mathsf{W}'\, s : A$. Notemos que este juicio sólo puede derivarse usando la regla `t-app`, por lo que tenemos que hay un tipo C tal que $\Gamma, \Box : B \vdash \mathsf{W}' : C \to A$ y $\Gamma, \Box : B \vdash s : C$. Por *h.i.* en el contexto más chico W' tenemos que $\Gamma \vdash \mathsf{W}'\langle t \rangle : C \to A$. Además, por fortalecimiento (Lema 3.1.2), tenemos que $\Gamma \vdash s : C$. Aplicando la regla `t-app` concluimos que $\Gamma \vdash \mathsf{W}'\langle t \rangle\, s : A$, como se pide.
3. *Derecha de una aplicación,* $\mathsf{W} = s\, \mathsf{W}'$. Por hipótesis, $\Gamma, \Box : B \vdash s\, \mathsf{W}' : A$. Notemos que este juicio sólo puede derivarse usando la regla `t-app`, por lo que tenemos que hay un tipo C tal que $\Gamma, \Box : B \vdash s : C \to A$ y $\Gamma, \Box : B \vdash \mathsf{W}' : C$. Por *h.i.* en el contexto más chico W' tenemos que $\Gamma \vdash \mathsf{W}'\langle t \rangle : C$. Además, por fortalecimiento (Lema 3.1.2)m tenemos que $\Gamma \vdash s : C \to A$. Aplicando la regla `t-app` concluimos que $\Gamma \vdash s\, \mathsf{W}'\langle t \rangle : A$, como se pide.
4. *Izquierda de una unificación,* $\mathsf{W} = (\mathsf{W}' \overset{\bullet}{=} s)$. Por hipótesis, $\Gamma, \Box : B \vdash \mathsf{W}' \overset{\bullet}{=} s : \mathcal{T}_{\mathbf{ok}}$. Notemos que este juicio sólo puede derivarse usando la regla `t-unif`, por lo que tenemos que hay un tipo A tal que $\Gamma, \Box : B \vdash \mathsf{W}' : A$ y $\Gamma, \Box : B \vdash s : A$. Por *h.i.* en el contexto más chico W' tenemos que $\Gamma \vdash \mathsf{W}'\langle t \rangle : A$. Además, por fortalecimiento (Lema 3.1.2) tenemos que $\Gamma \vdash s : A$. Aplicando la regla `t-unif` concluimos que $\Gamma \vdash \mathsf{W}'\langle t \rangle \overset{\bullet}{=} s : \mathcal{T}_{\mathbf{ok}}$ como se pide.
5. *Derecha de una unificación,* $\mathsf{W} = (s \overset{\bullet}{=} \mathsf{W}')$. Por hipótesis, $\Gamma, \Box : B \vdash s \overset{\bullet}{=} \mathsf{W}' : \mathcal{T}_{\mathbf{ok}}$. Notemos que este juicio sólo puede derivarse usando la regla `t-unif`, por lo que tenemos que hay un tipo A tal que $\Gamma, \Box : B \vdash s : A$ y $\Gamma, \Box : B \vdash \mathsf{W}' : A$. Por *h.i.* en el contexto más chico W' tenemos que $\Gamma \vdash \mathsf{W}'\langle t \rangle : A$. Además, por fortalecimiento (Lema 3.1.2) tenemos que $\Gamma \vdash s : A$. Aplicando la regla `t-unif` concluimos que $\Gamma \vdash s \overset{\bullet}{=} \mathsf{W}'\langle t \rangle : \mathcal{T}_{\mathbf{ok}}$ como se pide.
6. *Izquierda de una secuencia,* $\mathsf{W} = \mathsf{W}'; s$. Por hipótesis, $\Gamma, \Box : B \vdash \mathsf{W}'; s : A$. Notemos que este juicio sólo puede derivarse usando la regla `t-seq`, por lo que tenemos que $\Gamma, \Box : B \vdash \mathsf{W}' : \mathcal{T}_{\mathbf{ok}}$ y $\Gamma, \Box : B \vdash s : A$. Por *h.i.* en el contexto más chico W' tenemos que $\Gamma \vdash \mathsf{W}'\langle t \rangle : \mathcal{T}_{\mathbf{ok}}$. Además, por fortalecimiento (Lema 3.1.2) tenemos que $\Gamma \vdash s : A$. Aplicando la regla `t-seq` concluimos que $\Gamma \vdash \mathsf{W}'\langle t \rangle; s : A$ como se pide.
7. *Derecha de una secuencia,* $\mathsf{W} = s; \mathsf{W}'$. Por hipótesis, $\Gamma, \Box : B \vdash s; \mathsf{W}' : A$. Notemos que este juicio sólo puede derivarse usando la regla `t-seq`, por lo que tenemos que $\Gamma, \Box : B \vdash s : \mathcal{T}_{\mathbf{ok}}$ y $\Gamma, \Box : B \vdash \mathsf{W}' : A$. Por *h.i.* en el contexto más chico W' tenemos que $\Gamma \vdash \mathsf{W}'\langle t \rangle : A$. Además, por fortalecimiento

(Lema 3.1.2) tenemos que $\Gamma \vdash s : \mathcal{T}_{\mathbf{ok}}$. Aplicando la regla `t-seq` concluimos que $\Gamma \vdash s; \mathsf{W}'\langle t \rangle : A$ como se pide.

□

Lema 3.1.5 (Composición/descomposición de programas)**.** *Son equivalentes:*

1. $\Gamma \vdash P \oplus Q : A$

2. $\Gamma \vdash P : A$ *y* $\Gamma \vdash Q : A$

Demostración.

- ($1 \implies 2$) Por inducción en P.

  1. *Programa vacío,* i.e. $P = \mathtt{fail}$: Por hipótesis, $\Gamma \vdash Q : A$, y aplicando la regla `t-fail`, obtenemos $\Gamma \vdash \mathtt{fail} : A$, como se pide.
  2. *Alternativa,* i.e. $P = t \oplus P'$: Por hipótesis, $\Gamma \vdash t \oplus (P' \oplus Q) : A$. Notemos que este juicio sólo puede derivarse usando la regla `t-alt`, así que tenemos que $\Gamma \vdash t : A$ y $\Gamma \vdash P' \oplus Q : A$, entonces por *h.i.* tenemos que $\Gamma \vdash P' : A$ y que $\Gamma \vdash Q : A$. Podemos concluir, aplicando la regla `t-alt`, que $\Gamma \vdash t \oplus P : A$.

- ($2 \implies 1$) Por inducción en P.

  1. *Programa vacío,* i.e. $P = \mathtt{fail}$: Tenemos que demostrar que $\Gamma \vdash \mathtt{fail} \oplus Q : A$, y por hipótesis tenemos que $\Gamma \vdash Q : A$, con lo cual es directo.
  2. *Alternativa,* i.e. $P = t \oplus P'$: Por hipótesis, $\Gamma \vdash t \oplus P' : A$ y $\Gamma \vdash Q : A$. Notemos que el primer juicio sólo puede derivarse usando la regla `t-alt`, por lo que tenemos que $\Gamma \vdash t : A$ and $\Gamma \vdash P' : A$, entonces por *h.i.* tenemos que $\Gamma \vdash P' \oplus Q : A$. Aplicando la regla `t-alt`, obtenemos $\Gamma \vdash t \oplus (P' \oplus Q) : A$.

□

3.1.1. Tipado de problemas de unificación

Definición 3.1.6 (Tipado de problemas de unificación)**.** Definimos el juicio $\Gamma \vdash \mathcal{E}$ para cada problema de unificación $\mathcal{E}$ como sigue:

$$\frac{\Gamma \vdash \mathtt{v}_i \doteq \mathtt{w}_i : \mathcal{T}_{\mathbf{ok}} \quad \text{para todo } i = 1..n}{\Gamma \vdash \{\mathtt{v}_1 \doteq \mathtt{w}_1, \dots, \mathtt{v}_n \doteq \mathtt{w}_n\}}$$

Observación 3.1.7. El juicio $\Gamma \vdash \mathcal{E} \cup \mathcal{H}$ vale si y sólo si valen los juicios $\Gamma \vdash \mathcal{E}$ y $\Gamma \vdash \mathcal{H}$.

Lema 3.1.8 (Preservación para el algoritmo de unificación)**.** *Sea* $\Gamma \vdash \mathcal{E}$ *y supongamos que* $\mathcal{E} \rightsquigarrow \mathcal{H}$ *es un paso que no falla. Entonces* $\Gamma \vdash \mathcal{H}$*.*

Demostración. Sea $\mathcal{E} \rightsquigarrow \mathcal{H}$. Notemos que el paso no falla por lo que este juicio no puede ser resultado de aplicar las reglas `u-clash` o `u-occurs-check`. Consideramos los cinco casos restantes:

1. `u-delete`: En este caso tenemos:

$$\{x \stackrel{\bullet}{=} x\} \uplus \mathcal{E}' \rightsquigarrow \mathcal{E}'$$

Supongamos que vale $\Gamma \vdash \{x \stackrel{\bullet}{=} x\} \uplus \mathcal{E}'$. Entonces es inmediato concluir $\Gamma \vdash \mathcal{E}'$.

2. `u-orient`: En este caso tenemos:

$$\{\mathtt{v} \stackrel{\bullet}{=} x\} \uplus \mathcal{E}' \rightsquigarrow \{x \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E}'$$

donde $\mathtt{v} \notin \mathsf{Var}$. Supongamos que vale $\Gamma \vdash \{\mathtt{v} \stackrel{\bullet}{=} x\} \uplus \mathcal{E}'$. Notemos que entonces $\Gamma \vdash \mathtt{v} \stackrel{\bullet}{=} x : \mathcal{T}_{\mathbf{ok}}$ y $\Gamma \vdash \mathcal{E}'$. El primer juicio sólo puede derivarse usando la regla `t-unif`, así que valen $\Gamma \vdash \mathtt{v} : A$ y $\Gamma \vdash x : A$ para algún tipo A. Aplicando la regla `t-unif`, obtenemos $\Gamma \vdash x \stackrel{\bullet}{=} \mathtt{v} : \mathcal{T}_{\mathbf{ok}}$, y podemos concluir que $\Gamma \vdash \{x \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E}'$.

3. `u-match-lam`: En este caso tenemos que:

$$\{\lambda^{\ell} x.\, P \stackrel{\bullet}{=} \lambda^{\ell} x.\, P\} \uplus \mathcal{E}' \rightsquigarrow \mathcal{E}'$$

Supongamos que vale $\Gamma \vdash \{\lambda^{\ell} x.\, P \stackrel{\bullet}{=} \lambda^{\ell} x.\, P\} \uplus \mathcal{E}'$. Entonces, es inmediato concluir que $\Gamma \vdash \mathcal{E}'$.

4. `u-match-cons`: En este caso tenemos que:

$$\{\mathbf{c}\, \mathtt{v}_1 \ldots \mathtt{v}_n \stackrel{\bullet}{=} \mathbf{c}\, \mathtt{w}_1 \ldots \mathtt{w}_n\} \uplus \mathcal{E}' \rightsquigarrow \{\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{w}_1, \ldots, \mathtt{v}_n \stackrel{\bullet}{=} \mathtt{w}_n\} \uplus \mathcal{E}'$$

Supongamos que vale $\Gamma \vdash \{\mathbf{c}\, \mathtt{v}_1 \ldots \mathtt{v}_n \stackrel{\bullet}{=} \mathbf{c}\, \mathtt{w}_1 \ldots \mathtt{w}_n\} \uplus \mathcal{E}'$, entonces valen

$$\Gamma \vdash \mathbf{c}\, \mathtt{v}_1 \ldots \mathtt{v}_n \stackrel{\bullet}{=} \mathbf{c}\, \mathtt{w}_1 \ldots \mathtt{w}_n : \mathcal{T}_{\mathbf{ok}}$$

y $\Gamma \vdash \mathcal{E}'$. El primer juicio sólo puede derivarse usando la regla `t-unif`, así que los juicios $\Gamma \vdash \mathbf{c}\, \mathtt{v}_1 \ldots \mathtt{v}_n : A$ y $\Gamma \vdash \mathbf{c}\, \mathtt{w}_1 \ldots \mathtt{w}_n : A$ valen para algún tipo A. Sucesivamente, estos juicios sólo pueden derivarse usando la regla `t-cons` una vez y la regla `t-app` n veces, por lo que deben existir tipos $B_1, B_2, \ldots, B_n$ tales que el constructor $\mathbf{c}$ tenga como tipo $\mathcal{T}_{\mathbf{c}} = B_1 \to B_2 \to \ldots \to B_n \to A$ y, para todo $i = 1..n$, tenemos que $\Gamma \vdash \mathtt{v}_i : B_i$ y $\Gamma \vdash \mathtt{w}_i : B_i$. Aplicando la regla `t-unif`, obtenemos que $\Gamma \vdash \mathtt{v}_i \stackrel{\bullet}{=} \mathtt{w}_i : \mathcal{T}_{\mathbf{ok}}$ para todo $i = 1..n$, así que concluimos que $\Gamma \vdash \{\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{w}_1 \ldots \mathtt{v}_n \stackrel{\bullet}{=} \mathtt{w}_n\} \uplus \mathcal{E}'$, como se pide.

5. `u-eliminate`: En este caso tenemos que:

$$\{x \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E}' \rightsquigarrow \{x \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E}'\{x := \mathtt{v}\}$$

donde $x \in \mathsf{fv}(\mathcal{E}) \setminus \mathsf{fv}(\mathtt{v})$. Sea $\mathcal{E} = \{t_1, \ldots, t_n\}$, donde cada t_i es una ecuación. Supongamos que vale $\Gamma \vdash \{x \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E}'$. Entonces tenemos que valen $\Gamma \vdash x \stackrel{\bullet}{=} \mathtt{v} : \mathcal{T}_{\mathbf{ok}}$ y $\Gamma \vdash t_i : \mathcal{T}_{\mathbf{ok}}$ para todo $i = 1..n$. El primer juicio sólo puede derivarse usando la regla `t-unif`, así que $\Gamma \vdash x : A$ and $\Gamma \vdash \mathtt{v} : A$ vale para algún tipo A. El primer juicio necesariamente se deriva usando la regla `t-var`, entonces Γ es de la forma $\Delta, x : A$. Notemos que vale $\Delta, x : A \vdash t_i : \mathcal{T}_{\mathbf{ok}}$ para todo $i = 1..n$, así que usando el lema de sustitución, (Lema 3.1.3) vale $\Delta \vdash t_i\{x := \mathtt{v}\} : \mathcal{T}_{\mathbf{ok}}$ para todo $i = 1..n$. Además, cada uno de estos juicios pueden debilitarse (Lema 3.1.1) agregando $x : A$ como una hipótesis (que no se usa), para obtener que $\Gamma \vdash t_i\{x := \mathtt{v}\} : \mathcal{T}_{\mathbf{ok}}$. De esto concluimos que $\Gamma \vdash \{x \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E}'\{x := \mathtt{v}\}$, como se pide.

$\square$

3.1.2. Preservación de tipos

Proposición 3.1.9 (Preservación). *Sea* $\triangleright P : A$ *y supongamos que* $P \to Q$*. Entonces* $\triangleright Q : A$*.*

Demostración. Sea $P \to Q$. Consideramos seis casos, dependiendo de qué regla se aplica:

1. `alloc`: En este caso tenemos que:

$$P_1 \oplus \mathsf{W}\langle \lambda x.\, R\rangle \oplus P_2 \xrightarrow{\texttt{alloc}} P_1 \oplus \mathsf{W}\langle \lambda^\ell x.\, R\rangle \oplus P_2$$

donde ℓ es una locación fresca. Supongamos que $\triangleright P : A$, *i.e.* para algún contexto Γ tenemos:

$$\Gamma \vdash P_1 \oplus \mathsf{W}\langle \lambda x.\, R\rangle \oplus P_2 : A$$

Entonces, usando los lemas de composición (Lema 3.1.5) y de sustitución contextual (Lema 3.1.4) tenemos que vale lo siguiente, para algún tipo B:

$$\Gamma \vdash P_1 : A \quad \Gamma, \Box : B \vdash \mathsf{W} : A \quad \Gamma \vdash \lambda x.\, R : B \quad \Gamma \vdash P_2 : A$$

Además, el tercer juicio sólo puede derivarse usando la regla `t-lam`, así que B debe ser de la forma $B_1 \to B_2$, y vale $\Gamma, x : B_1 \vdash R : B_2$. Por lo tanto, aplicando la regla `t-laml` tenemos que $\Gamma \vdash \lambda^\ell x.\, R : B_1 \to B_2$. Aplicando nuevamente los lemas de sustitución contextual (Lema 3.1.4) y de composición (Lema 3.1.5), en sentido opuesto, tenemos que:

$$\Gamma \vdash P_1 \oplus \mathsf{W}\langle \lambda^\ell x.\, R\rangle \oplus P_2 : A$$

lo que significa que $\triangleright Q : A$, como se pide.

2. `beta`: Tenemos que:

$$P_1 \oplus \mathsf{W}\langle (\lambda^\ell x.\, R)\, \mathtt{v}\rangle \oplus P_2 \xrightarrow{\texttt{beta}} P_1 \oplus \mathsf{W}\langle R\{x := \mathtt{v}\}\rangle \oplus P_2$$

Supongamos que $\triangleright P : A$, *i.e.* que para algún contexto Γ tenemos:

$$\Gamma \vdash P_1 \oplus \mathsf{W}\langle (\lambda^\ell x.\, R)\, \mathtt{v}\rangle \oplus P_2 : A$$

Entonces por los lemas de composición (Lema 3.1.5) y de sustituticón contextual (Lema 3.1.4) tenemos que vale lo siguiente para algún tipo B:

$$\Gamma \vdash P_1 : A \quad \Gamma, \Box : B \vdash \mathsf{W} : A \quad \Gamma \vdash (\lambda^\ell x.\, R)\, \mathtt{v} : B \quad \Gamma \vdash P_2 : A$$

El tercer juicio sólo puede derivarse usando las reglas `t-app` y `t-laml`, así que hay un tipo C tal que valen $\Gamma, x : C \vdash R : B$ y $\Gamma \vdash \mathtt{v} : C$. Por el lema de sustitución (Lema 3.1.3), tenemos que $\Gamma \vdash R\{x := \mathtt{v}\} : B$. Usando nuevamente los lemas de sustitución contextual (Lema 3.1.4) y de composición (Lema 3.1.5), en sentido opuesto, tenemos que:

$$\Gamma \vdash P_1 \oplus \mathsf{W}\langle R\{x := \mathtt{v}\}\rangle \oplus P_2 : A$$

lo que significa que $\triangleright Q : A$, como se pide.

3. `seq`: Tenemos que:

$$P_1 \oplus \mathsf{W}\langle \mathtt{v}; t\rangle \oplus P_2 \xrightarrow{\mathtt{seq}} P_1 \oplus \mathsf{W}\langle t\rangle \oplus P_2$$

Supongamos que $\triangleright P : A$, *i.e.* que para algún contexto Γ tenemos:

$$\Gamma \vdash P_1 \oplus \mathsf{W}\langle \mathtt{v}; t\rangle \oplus P_2 : A$$

Por los lemas de composición (Lema 3.1.5) y de sustitutićon contextual (Lema 3.1.4) tenemos que lo siguiente vale, para algún tipo B:

$$\Gamma \vdash P_1 : A \quad \Gamma, \Box : B \vdash \mathsf{W} : A \quad \Gamma \vdash \mathtt{v}; t : B \quad \Gamma \vdash P_2 : A$$

El tercer juicio sólo puede derivarse usando la regla `t-seq`, por lo que vale $\Gamma \vdash \mathtt{v} : \mathcal{T}_{\mathbf{ok}}$ y $\Gamma \vdash t : B$. Por lo tanto aplicando nuevamente los lemas de sustitución contextual (Lema 3.1.4) y de composición (Lema 3.1.5), en sentido opuesto, tenemos que:

$$\Gamma \vdash P_1 \oplus \mathsf{W}\langle t\rangle \oplus P_2 : A$$

lo que significa que $\triangleright Q : A$, como se pide.

4. `fresh`: Tenemos que:

$$P_1 \oplus \mathsf{W}\langle \nu x. t\rangle \oplus P_2 \xrightarrow{\mathtt{fresh}} \mathsf{W}\langle t\{x := y\}\rangle \oplus P_2$$

donde y es una variable fresca. Supongamos que $\triangleright P : A$, *i.e.* para algún contexto Γ tenemos:

$$\Gamma \vdash P_1 \oplus \mathsf{W}\langle \nu x. t\rangle \oplus P_2 : A$$

Por los lemas de composición (Lema 3.1.5) y de sustitutićon contextual (Lema 3.1.4) tenemos que lo siguiente vale, para algún tipo B:

$$\Gamma \vdash P_1 : A \quad \Gamma, \Box : B \vdash \mathsf{W} : A \quad \Gamma \vdash \nu x. t : B \quad \Gamma \vdash P_2 : A$$

El tercer juicio sólo puede derivarse usando la regla `t-fresh`, por lo que existe un tipo C tal que vale $\Gamma, x : C \vdash t : B$. Por debilitamiento (Lema 3.1.1) tenemos que $\Gamma, y : C, x : C \vdash t : B$, y aplicando `t-var`, es inmediato ver que $\Gamma, y : C \vdash y : C$. Entonces, por el lema de sustitución (Lema 3.1.3) tenemos que $\Gamma, y : C \vdash t\{x := y\} : B$. Por lo tanto aplicando nuevamente los lemas de sustitución contextual (Lema 3.1.4) y de composición (Lema 3.1.5), en sentido opuesto, tenemos que:

$$\Gamma, y : C \vdash P_1 \oplus \mathsf{W}\langle t\{x := y\}\rangle \oplus P_2 : A$$

lo que significa que $\triangleright Q : A$, como se pide.

5. `unif`: Tenemos que:

$$P_1 \oplus \mathsf{W}\langle \mathtt{v} \stackrel{\bullet}{=} \mathtt{w}\rangle \oplus P_2 \xrightarrow{\mathtt{unif}} P_1 \oplus \mathsf{W}\langle \mathbf{ok}\rangle^{\sigma} \oplus P_2$$

donde $\sigma = \mathsf{mgu}(\{\mathtt{v} \stackrel{\bullet}{=} \mathtt{w}\})$. Supongamos que $\triangleright P : A$, *i.e.* que para algún contexto Γ tenemos:

$$\Gamma \vdash P_1 \oplus \mathsf{W}\langle \mathtt{v} \stackrel{\bullet}{=} \mathtt{w}\rangle \oplus P_2 : A$$

Por los lemas de composición (Lema 3.1.5) y de sustitutición contextual (Lema 3.1.4) tenemos que lo siguiente vale, para algún tipo B:

$$\Gamma \vdash P_1 : A \quad \Gamma, \Box : B \vdash \mathsf{W} : A \quad \Gamma \vdash \mathtt{v} \stackrel{\bullet}{=} \mathtt{w} : B \quad \Gamma \vdash P_2 : A$$

El tercer juicio sólo puede derivarse usando la regla `t-unif`, por lo que necesariamente $B = \mathcal{T}_{\mathbf{ok}}$, y en particular $\Gamma \vdash \mathsf{W}\langle \mathbf{ok} \rangle : A$ por el lema de sustitución contextual (Lema 3.1.4). Notemos que $\Gamma \vdash \{\mathtt{v} \stackrel{\bullet}{=} \mathtt{w}\}$. Además el unificador más general existe por hipótesis, por lo que por Teo. 2.2.5 hay una secuencia finita de $n \geq 0$ pasos:

$$\{\mathtt{v} \stackrel{\bullet}{=} \mathtt{w}\} = \mathcal{E}_0 \rightsquigarrow \mathcal{E}_1 \rightsquigarrow \ldots \rightsquigarrow \mathcal{E}_n = \{x_1 \stackrel{\bullet}{=} \mathtt{v}'_1, \ldots, x_n \stackrel{\bullet}{=} \mathtt{v}'_n\}$$

tal que para todo i, j tenemos que $x_i \neq x_j$ y $x_i \notin \mathsf{fv}(\mathtt{v}'_j)$. Además $\sigma = \mathsf{mgu}(\{\mathtt{v} \stackrel{\bullet}{=} \mathtt{w}\}) = \{x_1 \mapsto \mathtt{v}'_1, \ldots, x_n \mapsto \mathtt{v}'_n\}$. Recordemos que el algoritmo de unificación preserva el tipo (Lema 3.1.8) así que para cada $i = 1..n$ hay un tipo C_i tal que $\Gamma \vdash x_i : C_i$ y $\Gamma \vdash \mathtt{v}'_i : C_i$ vale. Esto significa que Γ tiene la forma $\Delta, x_1 : C_1, \ldots, x_n : C_n$. Aplicando repetidas veces el lema de sustitución (Lema 3.1.3), concluimos que $\Delta \vdash \mathsf{W}\langle \mathbf{ok} \rangle \{x_1 := \mathtt{v}'_1\} \ldots \{x_n := \mathtt{v}'_n\} : A$, es decir, $\Delta \vdash \mathsf{W}\langle \mathbf{ok} \rangle^{\sigma} : A$, con $\sigma = \{x_1 \mapsto \mathtt{v}'_1, \ldots, x_n \mapsto \mathtt{v}'_n\}$. Finalmente, aplicando los lemas de debilitamiento (Lema 3.1.1) y de composición (Lema 3.1.5) en el sentido opuesto, obtenemos que vale el juicio:

$$\Gamma \vdash P_1 \oplus \mathsf{W}\langle \mathbf{ok} \rangle^{\sigma} \oplus P_2 : A$$

Esto a su vez significa que $\rhd P_1 \oplus \mathsf{W}\langle \mathbf{ok} \rangle^{\sigma} \oplus P_2 : A$, como se pide.

6. `fail`: Tenemos que:

$$P_1 \oplus \mathsf{W}\langle \mathtt{v} \stackrel{\bullet}{=} \mathtt{w} \rangle \oplus P_2 \xrightarrow{\mathtt{fail}} P_1 \oplus P_2$$

donde $\mathsf{mgu}(\{\mathtt{v} \stackrel{\bullet}{=} \mathtt{w}\})$ falla. Supongamos que $\rhd P : A$, *i.e.* que para algún contexto Γ tenemos:

$$\Gamma \vdash P_1 \oplus \mathsf{W}\langle \mathtt{v} \stackrel{\bullet}{=} \mathtt{w} \rangle \oplus P_2 : A$$

Por el lema de composición (Lema 3.1.5) vale lo siguiente:

$$\Gamma \vdash P_1 : A \quad \Gamma \vdash P_2 : A$$

Aplicando nuevamente el lema de composición (Lema 3.1.5), en el sentido contrario, tenemos que $\Gamma \vdash P_1 \oplus P_2 : A$, lo que significa que $\rhd Q : A$, como se pide.

□

4. SEMÁNTICA DENOTACIONAL PARA EL CÁLCULO-$\lambda^{\mathtt{U}}$

En esta sección introducimos la semántica denotacional propuesta para el cálculo $\lambda^{\mathtt{U}}$. El resultado principal es un resultado de correctitud (débil) de la semántica operacional con respecto a la semántica denotacional. Para obtener este resultado, demostramos lemas intermedios, en particular, Irrelevencia y Composicionalidad.

4.1. Interpretaciones de tipos y términos

Lema 4.1.1 (Tipado único)**.** *Sea* $\Gamma \vdash X : A$ *un juicio derivable, donde* X *es un término o un programa. Entonces cada derivación para* X *tiene la misma forma. En particular, si* $\Gamma' \vdash X : A'$ *es derivable, entonces* $A = A'$ *y los contextos* Γ *y* Γ' *coinciden en los tipos de todas las variables en* $\mathsf{fv}(X)$*.*

Este lema justifica que vayamos a escribir $\vdash X : A$ si X es un término/programa tipable de tipo A (omitiendo el contexto Γ).

A cada tipo A le asociamos un dominio de interpretación que se nota $[\![A]\!]$. A cada programa y a cada término de tipo A le asignamos no un elemento del dominio de interpretación $[\![A]\!]$, sino un *conjunto* de elementos de $[\![A]\!]$. Esto es debido a que, por la presencia del no determinismo, hay programas o términos que pueden no tener ninguna interpretación o tener más de una; utilizar conjuntos permite modelar esto. Por ejemplo, el programa `fail` informalmente se interpreta como la ausencia de resultados, y formalmente esto se refleja en el hecho de que la denotación de `fail` es el conjunto vacío. Un ejemplo de programas con más de una interpretación posible es $y \oplus \lambda^{\ell} x.\, y$, cuya interpretación nos gustaría que se componga por las interpretaciones de y y por las de $\lambda^{\ell} x.\, y$.

En pos de definir las interpretaciones de tipos y términos, debemos hacer algunas consideraciones. Un objetivo al que aspiramos es que la semántica operacional sea correcta con respecto a la denotacional. Esto trae como consecuencia que la denotación de un valor debería ser un conjunto *unitario*, es decir, un conjunto de exactamente un elemento. Esto se puede ilustrar con un ejemplo: como $(\lambda^{\ell} x.\, \mathbf{c}\, x\, x)\, \mathtt{v} \to \mathbf{c}\, \mathtt{v}\, \mathtt{v}$, la interpretación de $(\lambda^{\ell} x.\, \mathbf{c}\, x\, x)\, \mathtt{v}$ debería coincidir con la de $\mathbf{c}\, \mathtt{v}\, \mathtt{v}$. Informalmente, si la semántica de $\mathtt{v}$ no fuera un conjunto unitario, por ejemplo si se tuviera que $[\![\mathtt{v}]\!] = \{a, b\}$, la denotación de $\mathbf{c}\, \mathtt{v}\, \mathtt{v}$ sería $\{caa, cab, cba, cbb\}$, en tanto que la denotación de $(\lambda^{\ell} x.\, \mathbf{c}\, x\, x)\, \mathtt{v}$ sería $\{caa, cbb\}$, al menos si se interpretaran la aplicación y la abstracción de la manera "usual". Otro ejemplo que ilustra esta situación es $(\lambda^{\ell} x.\, \mathbf{c})\, \mathtt{v} \to \mathbf{c}$. Si la semántica de $\mathtt{v}$ fuera el conjunto vacío, es decir, si $[\![\mathtt{v}]\!] = \varnothing$, la denotación de $(\lambda^{\ell} x.\, \mathbf{c})\, \mathtt{v}$ sería siempre vacía, en tanto que la denotación de $\mathbf{c}$ sería posiblemente un conjunto no vacío. Estos ejemplos sugieren que la interpretación de un valor debería en efecto ser un conjunto unitario.

Según explicamos en el párrafo anterior, la interpretación de un valor de tipo A debería ser un conjunto unitario incluido en el dominio de interpretación asociado al tipo A. Esto impone la restricción de que los dominios de interpretación asociados a cada tipo deben ser conjuntos no vacíos. Como consecuencia, exigimos que el dominio de interpretación asociado a cada *tipo base* sea un conjunto no vacío. Además, es posible demostrar que, si se cumple este requisito, la interpretación de un tipo arbitrario resulta ser no vacía.

Definición 4.1.2 (Interpretación de tipos)**.** Supongamos dado un conjunto **no vacío**,

S_α, para cada tipo base α. La *interpretación* de un tipo A se escribe $[\![A]\!]$ y se define recursivamente como sigue:

$$[\![\alpha]\!] \stackrel{\text{def}}{=} \mathsf{S}_\alpha$$
$$[\![A \to B]\!] \stackrel{\text{def}}{=} [\![A]\!] \to \mathcal{P}([\![B]\!])$$

donde $X \to Y$ denota el conjunto de funciones $f : X \to Y$.

Si A es un tipo y $a \in [\![A]\!]$, entonces decimos que a es A-*unitario* de acuerdo con la siguiente definición inductiva:

- Cualquier elemento $a \in [\![\alpha]\!]$ es α-unitario.
- Un elemento $f \in [\![A \to B]\!]$ es $(A \to B)$-unitario si para cada $a \in [\![A]\!]$ el conjunto $f(a) = \{b\} \subseteq [\![B]\!]$ es un conjunto de un solo elemento y b es B-unitario.

A veces decimos que un objeto a es *unitario* si el tipo se deduce del contexto. Si f es $(A \to B)$-unitario, y $a \in [\![A]\!]$ entonces, por abuso de notación, podemos escribir $f(a)$ para el único elemento $b \in f(a)$.

Observación 4.1.3 (Las interpretaciones son no vacías)*.* Por cada tipo A, el conjunto $[\![A]\!]$ es no vacío, dado que requerimos que S_α sea no vacío.

Definición 4.1.4 (Interpretación de los términos)**.** Supongamos dado un elemento $\mathcal{T}_{\mathbf{c}}$-unitario $\mathsf{R}_{\mathbf{c}} \in [\![\mathcal{T}_{\mathbf{c}}]\!]$ para cada constructor $\mathbf{c}$.

Para asegurar que el algoritmo de unificación es correcto con respecto a la semántica denotacional, asumimos que vale el principio de **no confusión** sobre las interpretaciones de constructores, es decir que sus interpretaciones son inyectivas y disjuntas. Más precisamente, supongamos que:

$$\mathsf{R}_{\mathbf{c}}(a_1)\dots(a_n) = \mathsf{R}_{\mathbf{d}}(b_1)\dots(b_m)$$

Entonces $\mathbf{c} = \mathbf{d}$, $n = m$ y $a_i = b_i$ para todo $i = 1..n$.

Una *asignación de variables* es una función $\rho : \mathsf{Var} \to \bigcup_{A \in \mathsf{Type}} [\![A]\!]$ tal que $\rho(x^A) \in [\![A]\!]$ para cada variable x^A de cada tipo A. Si ρ es una asignación de variables y $a \in [\![A]\!]$, entonces $\rho[x^A \mapsto a]$ es la asignación de variables definida como:

$$\rho[x^A \mapsto a](y) \stackrel{\text{def}}{=} \begin{cases} a & \text{si } x = y \\ \rho(y) & \text{en otro caso} \end{cases}$$

Sea $\vdash t : A$ (resp. $\vdash P : A$) un término tipable (resp. programa) y sea ρ una asignación de variables. Escribimos Φ, Φ', etc. para secuencias de variables ($\Phi = x_1^{A_1}, \dots, x_n^{A_n}$) sin repeticiones. Si $\vec{A} = (A_1, \dots, A_n)$ es una secuencia de tipos, escribimos $[\![\vec{A}]\!]$ para $[\![A_1]\!] \times \dots \times [\![A_n]\!]$. Además, si $\vec{x} = (x_1, \dots, x_n)$ es una secuencia de nombres de variables, escribimos $\vec{x}^{\vec{A}}$ para la secuencia $(x_1^{A_1}, \dots, x_n^{A_n})$. Además, si $\vec{a} = (a_1, \dots, a_n) \in [\![\vec{A}]\!]$ entonces escribimos $\rho[\vec{x} \mapsto \vec{a}]$ para $\rho[x_1 \mapsto a_1]\dots[x_n \mapsto a_n]$. A veces tratamos a las secuencias de variables como conjuntos, cuando el orden no es relevante. Definimos la interpretación $[\![t]\!]_\rho^\Phi \subseteq [\![A]\!]$ (resp. $[\![P]\!]_\rho^\Phi \subseteq [\![A]\!]$) como sigue. Cuando Φ es vacío, escribimos $[\![t]\!]_\rho$ (resp. $[\![P]\!]_\rho$). Intuitivamente las variables listadas en Φ se supone que son vabiales *frescas*. Las siguientes ecuaciones definen la semántica de un término (resp. un programa) cuando Φ es no vacío, o sea, $\Phi = x^A, \Phi'$.

$$\llbracket t \rrbracket_\rho^{x^A,\Phi'} \stackrel{\text{def}}{=} \{b \mid a \in \llbracket A \rrbracket, b \in \llbracket t \rrbracket^{\Phi'}_{\rho[x \mapsto a]}\}$$
$$\llbracket P \rrbracket_\rho^{x^A,\Phi'} \stackrel{\text{def}}{=} \{b \mid a \in \llbracket A \rrbracket, b \in \llbracket P \rrbracket^{\Phi'}_{\rho[x \mapsto a]}\}$$

La interpretación de los términos está definida recursivamente como:

$$\begin{aligned}
\llbracket x^A \rrbracket_\rho &\stackrel{\text{def}}{=} \{\rho(x^A)\}\\
\llbracket \mathbf{c} \rrbracket_\rho &\stackrel{\text{def}}{=} \{\mathsf{R}_\mathbf{c}\}\\
\llbracket \lambda x^A.\, P \rrbracket_\rho &\stackrel{\text{def}}{=} \{f\} \quad \text{donde } f : \llbracket A \rrbracket \to \mathcal{P}(\llbracket B \rrbracket) \text{ está dado por } f(a) = \llbracket P \rrbracket_{\rho[x^A \mapsto a]}\\
\llbracket \lambda^\ell x^A.\, P \rrbracket_\rho &\stackrel{\text{def}}{=} \{f\} \quad \text{donde } f : \llbracket A \rrbracket \to \mathcal{P}(\llbracket B \rrbracket) \text{ está dado por } f(a) = \llbracket P \rrbracket_{\rho[x^A \mapsto a]}\\
\llbracket t\, s \rrbracket_\rho &\stackrel{\text{def}}{=} \{b \mid f \in \llbracket t \rrbracket_\rho, a \in \llbracket s \rrbracket_\rho, b \in f(a)\}\\
\llbracket t \stackrel{\bullet}{=} s \rrbracket_\rho &\stackrel{\text{def}}{=} \{\mathsf{R}_\mathbf{ok} \mid a \in \llbracket t \rrbracket_\rho, b \in \llbracket s \rrbracket_\rho, a = b\}\\
\llbracket t; s \rrbracket_\rho &\stackrel{\text{def}}{=} \{a \mid b \in \llbracket t \rrbracket_\rho, a \in \llbracket s \rrbracket_\rho\}\\
\llbracket \nu x^A.\, t \rrbracket_\rho &\stackrel{\text{def}}{=} \{b \mid a \in \llbracket A \rrbracket, b \in \llbracket t \rrbracket_{\rho[x^A \mapsto a]}\}\\
\llbracket \mathtt{fail}^A \rrbracket_\rho &\stackrel{\text{def}}{=} \varnothing\\
\llbracket t \oplus P \rrbracket_\rho &\stackrel{\text{def}}{=} \llbracket t \rrbracket_\rho \cup \llbracket P \rrbracket_\rho
\end{aligned}$$

Ejemplo 4.1.5. $\llbracket \lambda^\ell x^A.\, x^A \rrbracket = \{f\}$, *donde* $f : \llbracket A \rrbracket \to \mathcal{P}(\llbracket A \rrbracket)$ *está dado porestá dado* $f(a) = \llbracket x^A \rrbracket_{[x^A \mapsto a]} = \{a\}$

Ejemplo 4.1.6. $\llbracket (\lambda^\ell x^{\mathcal{T}_\mathbf{ok}}.\, x^{\mathcal{T}_\mathbf{ok}})\, (\mathbf{c} \stackrel{\bullet}{=} y^{\mathcal{T}_\mathbf{c}}) \rrbracket_{[y^{\mathcal{T}_\mathbf{c}} \mapsto \mathsf{R}_\mathbf{c}]} = \{a \mid f \in \llbracket \lambda^\ell x^{\mathcal{T}_\mathbf{ok}}.\, x^{\mathcal{T}_\mathbf{ok}} \rrbracket_{[y^{\mathcal{T}_\mathbf{c}} \mapsto \mathsf{R}_\mathbf{c}]}, b \in \llbracket \mathbf{c} \stackrel{\bullet}{=} y \rrbracket_{[y^{\mathcal{T}_\mathbf{c}} \mapsto \mathsf{R}_\mathbf{c}]}, a \in f(b)\}$ *donde*

- $f : \llbracket \mathcal{T}_\mathbf{ok} \rrbracket \to \mathcal{P}(\llbracket \mathcal{T}_\mathbf{ok} \rrbracket)$ *está dado por* $f(\mathsf{R}_\mathbf{ok}) = \llbracket x^{\mathcal{T}_\mathbf{ok}} \rrbracket_{[x^{\mathcal{T}_\mathbf{ok}} \mapsto \mathsf{R}_\mathbf{ok}][y^{\mathcal{T}_\mathbf{c}} \mapsto \mathsf{R}_\mathbf{c}]} = \{\mathsf{R}_\mathbf{ok}\}$
- $b \in \{\mathsf{R}_\mathbf{ok} \mid \mathsf{R}_\mathbf{c} \in \llbracket \mathbf{c} \rrbracket_{[y^{\mathcal{T}_\mathbf{c}} \mapsto \mathsf{R}_\mathbf{c}]}, \mathsf{R}_\mathbf{c} \in \llbracket y^{\mathcal{T}_\mathbf{c}} \rrbracket_{[y^{\mathcal{T}_\mathbf{c}} \mapsto \mathsf{R}_\mathbf{c}]}, \mathsf{R}_\mathbf{c} = \mathsf{R}_\mathbf{c}\}$.

Ejemplo 4.1.7.

$$\begin{aligned}
&\llbracket y^{B \to A} \oplus \lambda^\ell x^B.\, z^A \rrbracket_{[y^{B\to A} \mapsto f][z^A \mapsto c]}\\
= \;&\llbracket y^{B \to A} \rrbracket_{[y^{B\to A} \mapsto f][z^A \mapsto c]} \cup \llbracket \lambda^\ell x^B.\, z^A \rrbracket_{[y^{B\to A} \mapsto f][z^A \mapsto c]}\\
= \;&\{f\} \cup \llbracket \lambda^\ell x^B.\, z^A \rrbracket_{[y^{B\to A} \mapsto f][z^A \mapsto c]}\\
= \;&\{f\} \cup \{g\}\\
= \;&\{f, g\}
\end{aligned}$$

donde $g : \llbracket B \rrbracket \to \llbracket \mathcal{P}(A) \rrbracket$ *está dado por* $g(b) = \llbracket z^A \rrbracket_{[y^{B\to A} \mapsto f][z^A \mapsto c][x^B \mapsto b]} = \{c\}$

Ejemplo 4.1.8.

$$\begin{aligned}
&\llbracket ((\lambda x^B.\, \mathbf{c}\, y^{\mathcal{T}_\mathbf{d}} \stackrel{\bullet}{=} x^B)\, (\mathbf{c}\, \mathbf{d})); y^{\mathcal{T}_\mathbf{d}} \rrbracket_{[y^{\mathcal{T}_\mathbf{d}} \mapsto \mathsf{R}_\mathbf{d}]}\\
= \;&\{\mathsf{R}_\mathbf{d} \mid \mathsf{R}_\mathbf{ok} \in \llbracket (\lambda x^B.\, \mathbf{c}\, y^{\mathcal{T}_\mathbf{d}} \stackrel{\bullet}{=} x^B)\, (\mathbf{c}\, \mathbf{d}) \rrbracket_{[y^{\mathcal{T}_\mathbf{d}} \mapsto \mathsf{R}_\mathbf{d}]}, \mathsf{R}_\mathbf{d} \in \llbracket y^{\mathcal{T}_\mathbf{d}} \rrbracket_{[y^{\mathcal{T}_\mathbf{d}} \mapsto \mathsf{R}_\mathbf{d}]}\}\\
= \;&\{\mathsf{R}_\mathbf{d} \mid \mathsf{R}_\mathbf{ok} \in \llbracket (\lambda x^B.\, \mathbf{c}\, y^{\mathcal{T}_\mathbf{d}} \stackrel{\bullet}{=} x^B)\, (\mathbf{c}\, \mathbf{d}) \rrbracket_{[y^{\mathcal{T}_\mathbf{d}} \mapsto \mathsf{R}_\mathbf{d}]}\}\\
= \;&\{\mathsf{R}_\mathbf{d} \mid f \in \llbracket \lambda x^B.\, \mathbf{c}\, y^{\mathcal{T}_\mathbf{d}} \stackrel{\bullet}{=} x^B \rrbracket_{[y^{\mathcal{T}_\mathbf{d}} \mapsto \mathsf{R}_\mathbf{d}]}, b \in \llbracket \mathbf{c}\, \mathbf{d} \rrbracket_{[y^{\mathcal{T}_\mathbf{d}} \mapsto \mathsf{R}_\mathbf{d}]}, \mathsf{R}_\mathbf{ok} \in f(b)\}\\
= \;&\{\mathsf{R}_\mathbf{d} \mid b \in \llbracket \mathbf{c}\, \mathbf{d} \rrbracket_{[y^{\mathcal{T}_\mathbf{d}} \mapsto \mathsf{R}_\mathbf{d}]}, \mathsf{R}_\mathbf{ok} \in f(b)\}\\
= \;&\{\mathsf{R}_\mathbf{d} \mid b \in \mathsf{R}_\mathbf{c}(\mathsf{R}_\mathbf{d}), \mathsf{R}_\mathbf{ok} \in f(b)\}\\
= \;&\{\mathsf{R}_\mathbf{d}\}
\end{aligned}$$

donde $f : [\![B]\!] \to \mathcal{P}([\![\mathcal{T}_{\mathbf{ok}}]\!])$ *está dado por:*

$$\begin{aligned} f(b) &= [\![\mathbf{c}\, y^{\mathcal{T}_{\mathbf{d}}} \overset{\bullet}{=} x^B]\!]_{[y^{\mathcal{T}_{\mathbf{d}}} \mapsto \mathsf{R}_{\mathbf{d}}][x^B \mapsto b]} \\ &= \{\mathsf{R}_{\mathbf{ok}} \mid c \in [\![\mathbf{c}\, y^{\mathcal{T}_{\mathbf{d}}}]\!]_{[y^{\mathcal{T}_{\mathbf{d}}} \mapsto \mathsf{R}_{\mathbf{d}}][x^B \mapsto b]}, b \in [\![x^B]\!]_{[y^{\mathcal{T}_{\mathbf{d}}} \mapsto \mathsf{R}_{\mathbf{d}}][x^B \mapsto b]}, c = b\} \\ &= \{\mathsf{R}_{\mathbf{ok}} \mid c \in [\![\mathbf{c}\, y^{\mathcal{T}_{\mathbf{d}}}]\!]_{[y^{\mathcal{T}_{\mathbf{d}}} \mapsto \mathsf{R}_{\mathbf{d}}][x^B \mapsto b]}, c = b\} \\ &= \{\mathsf{R}_{\mathbf{ok}} \mid c \in \mathsf{R}_{\mathbf{c}}(\mathsf{R}_{\mathbf{d}}), c = b\} \\ &= \{\mathsf{R}_{\mathbf{ok}}\} \end{aligned}$$

4.2. Propiedades básicas de la semántica denotacional

Lema 4.2.1 (Irrelevancia)**.** *Sea* $\vdash X : A$ *un término o programa tipable.*

1. *Si* ρ, ρ' *son asignaciones de variables que coinciden en* $\mathsf{fv}(X) \setminus \Phi$*,* i.e. *para cualquier variable* $x^B \in \mathsf{fv}(X) \setminus \Phi$ *se tiene que* $\rho(x^B) = \rho'(x^B)$*, entonces* $[\![X]\!]^{\Phi}_{\rho} = [\![X]\!]^{\Phi}_{\rho'}$*.*

2. *Sean* Φ, Φ' *secuencias de variables tales que* $\mathsf{fv}(X) \setminus \Phi = \mathsf{fv}(X) \setminus \Phi'$*. Entonces* $[\![X]\!]^{\Phi}_{\rho} = [\![X]\!]^{\Phi'}_{\rho}$*.*

Demostración.

1. Por inducción en Φ.

   1.1 **Vacío, *i.e.* $\Phi = \varnothing$.** Por inducción en X, *i.e.* el término o programa:

   1.1.1 **Variable, $X = x^A$.** Inmediato, ya que $[\![x^A]\!]_{\rho} = \rho(x^A) = \rho'(x^A) = [\![x^A]\!]_{\rho'}$.

   1.1.2 **Constructor, $X = \mathbf{c}$.** Inmediato, ya que $[\![\mathbf{c}]\!]_{\rho} = \{\mathsf{R}_{\mathbf{c}}\} = [\![\mathbf{c}]\!]_{\rho'}$.

   1.1.3 **Abstracción, $X = \lambda x^A. P$.** Notemos que $[\![\lambda x^A. P]\!]_{\rho} = \{f\}$ donde $f(a) = [\![P]\!]_{\rho[x^A \mapsto a]}$. Simétricamente, $[\![\lambda x^A. P]\!]_{\rho'} = \{g\}$ donde $g(a) = [\![P]\!]_{\rho'[x^A \mapsto a]}$. Notemos que, para cualquier $a \in [\![A]\!]$ fijo, tenemos que $\rho[x^A \mapsto a]$ y $\rho'[x^A \mapsto a]$ coinciden en $\mathsf{fv}(\lambda x. P)$ y también en x así que además coinciden en $\mathsf{fv}(P)$. Esto nos permite aplicar la *h.i.* para concluir que $[\![P]\!]_{\rho[x^A \mapsto a]} = [\![P]\!]_{\rho'[x^A \mapsto a]}$, por lo que $f = g$, como se pide.

   1.1.4 **Abstracción alojada, $X = \lambda^{\ell} x^A. P$.** Análogo al caso anterior.

   1.1.5 **Aplicación, $X = t\, s$.** Directo por *h.i.*, ya que $[\![t\, s]\!]_{\rho} = \{b \mid f \in [\![t]\!]_{\rho}, a \in [\![s]\!]_{\rho}, b \in f(a)\} = \{b \mid f \in [\![t]\!]_{\rho'}, a \in [\![s]\!]_{\rho'}, b \in f(a)\} = [\![t\, s]\!]_{\rho'}$.

   1.1.6 **Unificación, $X = (t \overset{\bullet}{=} s)$.** Directo por *h.i.* ya que $[\![t \overset{\bullet}{=} s]\!]_{\rho} = \{\mathsf{R}_{\mathbf{ok}} \mid a \in [\![t]\!]_{\rho}, b \in [\![s]\!]_{\rho}, a = b\} = \{\mathsf{R}_{\mathbf{ok}} \mid a \in [\![t]\!]_{\rho'}, b \in [\![s]\!]_{\rho'}, a = b\} = [\![t \overset{\bullet}{=} s]\!]_{\rho'}$.

   1.1.7 **Secuencia, $X = t; s$.** Directo por *h.i.* ya que $[\![t; s]\!]_{\rho} = \{b \mid a \in [\![t]\!]_{\rho}, b \in [\![s]\!]_{\rho}\} = \{b \mid a \in [\![t]\!]_{\rho'}, b \in [\![s]\!]_{\rho'}\} = [\![t; s]\!]_{\rho'}$.

   1.1.8 **Variable fresca, $X = \nu x^A. t$.** Notemos que $[\![\nu x^A. t]\!]_{\rho} = \{b \mid a \in [\![A]\!], b \in [\![t]\!]_{\rho[x^A \mapsto a]}\}$. Simétricamente, $[\![\nu x^A. t]\!]_{\rho'} = \{b \mid a \in [\![A]\!], b \in [\![t]\!]_{\rho'[x^A \mapsto a]}\}$. Notemos que, para cualquier $a \in [\![A]\!]$ fijo, tenemos que $\rho[x^A \mapsto a]$ y que $\rho'[x^A \mapsto a]$ coinciden en $\mathsf{fv}(\nu x^A. t)$ y también en x, por lo que coinciden en $\mathsf{fv}(t)$. Esto nos permite aplicar la *h.i.* para conclui que $[\![t]\!]_{\rho[x^A \mapsto a]} = [\![t]\!]_{\rho'[x^A \mapsto a]}$, entonces $[\![\nu x^A. t]\!]_{\rho} = [\![\nu x^A. t]\!]_{\rho'}$, como se pide.

   1.1.9 **Fail, $X = \mathtt{fail}^A$.** Inmediato, ya que $[\![\mathtt{fail}^A]\!]_{\rho} = \varnothing = [\![\mathtt{fail}^A]\!]_{\rho'}$.

   1.1.10 **Alternativa, $X = t \oplus P$.** Directo por *h.i.* ya que $[\![t \oplus P]\!]_{\rho} = [\![t]\!]_{\rho} \cup [\![P]\!]_{\rho} = [\![t]\!]_{\rho'} \cup [\![P]\!]_{\rho'} = [\![t \oplus P]\!]_{\rho}$.

1.2 **No vacío, *i.e.* $\Phi = x^A, \Psi$.** Entonces notemos que $\rho[x \mapsto a]$ y $\rho[x \mapsto a]$ coinciden en $\mathsf{fv}(X) \setminus \Psi$ para cualquier $a \in [\![A]\!]$. Entonces:

$$\begin{aligned} [\![X]\!]_\rho^{x^A,\Psi} &= \{b \mid a \in [\![A]\!], [\![X]\!]_{\rho[x\mapsto a]}^{\Psi}\} \\ &= \{b \mid a \in [\![A]\!], [\![X]\!]_{\rho'[x\mapsto a]}^{\Psi}\} \quad \text{(Por } h.i.\text{)} \\ &= [\![X]\!]_{\rho'}^{x^A,\Psi} \end{aligned}$$

2. Notemos que, vistos como conjuntos, $\mathsf{fv}(X) \cap \Phi = \mathsf{fv}(X) \cap \Phi'$, entonces la secuencia Φ puede convertirse en la secuencia Φ' a través de las siguientes tres operaciones:

- eliminar una variable *espúrea* (que no está en $\mathsf{fv}(X)$),
- agregar una variable espúrea,
- intercambiar variables.

Efectivamente, primero notamos que las dos propiedades que siguen valen:

- **Agregar/remover variables espúreas.** $[\![X]\!]_\rho^{\Phi} = [\![X]\!]_\rho^{x^A,\Phi}$ si $x^A \notin \mathsf{fv}(X)$. Basta mostrar que $[\![X]\!]_\rho^{\Phi} = \{b \mid a \in [\![A]\!], b \in [\![X]\!]_{\rho[x\mapsto a]}^{\Phi}\}$, lo que es inmediato por el ítem 1. de este lema: $[\![X]\!]_\rho^{\Phi} = [\![X]\!]_{\rho[x\mapsto a]}^{\Phi}$ para todo $a \in [\![A]\!]$. Notemos que usamos el hecho de que $[\![A]\!]$ es un conjunto no vacío.
- **Intercambiar.** $[\![X]\!]_\rho^{\Phi_1,x^A,\Phi_2} = [\![X]\!]_\rho^{x^A,\Phi_1,\Phi_2}$.
  Procedemos por inducción en Φ_1. Si Φ_1 es vacío, es inmediato. En otro caso, sea $\Phi_1 = y^B, \Phi_1'$. Entonces:

$$\begin{aligned} [\![X]\!]_\rho^{y^B,\Phi_1',x^A,\Phi_2} &= \{c \mid b \in [\![B]\!], c \in [\![X]\!]_{\rho[y\mapsto b]}^{\Phi_1',x^A,\Phi_2}\} \\ &= \{c \mid b \in [\![B]\!], c \in [\![X]\!]_{\rho[y\mapsto b]}^{x^A,\Phi_1',\Phi_2}\} \quad \text{(Por } h.i.\text{)} \\ &= \{c \mid b \in [\![B]\!], a \in [\![A]\!], c \in [\![X]\!]_{\rho[y\mapsto b][x\mapsto a]}^{\Phi_1',\Phi_2}\} \\ &= \{c \mid a \in [\![A]\!], b \in [\![B]\!], c \in [\![X]\!]_{\rho[x\mapsto a][y\mapsto b]}^{\Phi_1',\Phi_2}\} \quad (\star) \\ &= [\![X]\!]_\rho^{x^A,y^B,\Phi_1',\Phi_2} \end{aligned}$$

  Para justificar el paso $(\star)$, notemos que $\rho[y \mapsto b][x \mapsto a] = \rho[x \mapsto a][y \mapsto b]$ vale por definición.

Ahora procedemos por inducción en Φ:

2.1 **Vacío, *i.e.* $\Phi = \varnothing$.** Entonces $\mathsf{fv}(X) = \mathsf{fv}(X) \setminus \Phi'$ por lo que $\Phi' \cap \mathsf{fv}(X) = \varnothing$. Agregando iterativamente variables espúreas tenemos que $[\![X]\!]_\rho^{\Phi} = [\![X]\!]_\rho = [\![X]\!]_\rho^{\Phi'}$, como se pide.

2.2 **No vacío, *i.e.* $\Phi = x^A, \Psi$.** Consideramos dos casos, dependiendo de si la variable x^A es espúrea (*i.e.* $x^A \notin \mathsf{fv}(X)$) o no:

2.2.1 Si $x^A \notin \mathsf{fv}(X)$, notemos que $\mathsf{fv}(X) \setminus \Psi = \mathsf{fv}(X) \setminus \Phi = \mathsf{fv}(X) \setminus \Phi'$, entonces quitando la variable espúrea y aplicando la *h.i.* tenemos que $[\![X]\!]_\rho^{x^A,\Psi} = [\![X]\!]_\rho^{\Psi} = [\![X]\!]_\rho^{\Phi'}$.

2.2.2 Si $x^A \in \mathsf{fv}(X)$, como $\mathsf{fv}(t) \setminus \Phi = \mathsf{fv}(t) \setminus \Phi'$ tenemos que $x^A \in \Phi'$. Por lo tanto Φ' debe ser de la forma $\Phi' = \Phi'_1, x^A, \Phi'_2$. Entonces aplicando la *h.i.* e intercambiando tenemos que $[\![X]\!]_\rho^{x^A,\Psi} = [\![X]\!]_\rho^{x^A,\Phi'_1,\Phi'_2} = [\![X]\!]_\rho^{\Phi'_1,x^A,\Phi'_2}$ como se pide.

$\square$

Lema 4.2.2 (Composicionalidad)**.**

1. $[\![P \oplus Q]\!]_\rho^\Phi = [\![P]\!]_\rho^\Phi \cup [\![Q]\!]_\rho^\Phi$.

2. *Si* W *es un contexto cuyo agujero tiene tipo* A*, entonces* $[\![\mathsf{W}\langle t\rangle]\!]_\rho = \{b \mid a \in [\![t]\!]_\rho, b \in [\![\mathsf{W}]\!]_{\rho[\square^A \mapsto a]}\}$.

Demostración.

1. Por inducción en la longitud de Φ.

1.1 **Vacío, $\Phi = \varnothing$.** Entonces procedemos por inducción en P:

1.1.1 Si $P = \mathtt{fail}$, entonces $[\![\mathtt{fail} \oplus Q]\!]_\rho = [\![Q]\!]_\rho = [\![\mathtt{fail}]\!]_\rho \cup [\![Q]\!]_\rho$.

1.1.2 Si $P = t \oplus P'$, entonces

$$\begin{array}{rcll} [\![(t \oplus P') \oplus Q]\!]_\rho & = & [\![t \oplus (P' \oplus Q)]\!]_\rho & \\ & = & [\![t]\!]_\rho \cup [\![P' \oplus Q]\!]_\rho & \\ & = & [\![t]\!]_\rho \cup [\![P']\!]_\rho \cup [\![Q]\!]_\rho & \text{(Por } h.i.\text{)} \\ & = & [\![t \oplus P']\!]_\rho \cup [\![Q]\!]_\rho & \end{array}$$

1.2 **No vacío, $\Phi = x^A, \Phi'$.** Entonces:

$$\begin{array}{cll} & [\![P \oplus Q]\!]_\rho^{x^A,\Phi'} & \\ = & \{b \mid a \in [\![A]\!], b \in [\![P \oplus Q]\!]_{\rho[x\mapsto a]}^{\Phi'}\} & \\ = & \{b \mid a \in [\![A]\!], b \in ([\![P]\!]_{\rho[x\mapsto a]}^{\Phi'} \cup [\![Q]\!]_{\rho[x\mapsto a]}^{\Phi'})\} & \text{(Por } h.i.\text{)} \\ = & \{b \mid a \in [\![A]\!], b \in [\![P]\!]_{\rho[x\mapsto a]}^{\Phi'}\} \cup \{b \mid a \in [\![A]\!], b \in [\![Q]\!]_{\rho[x\mapsto a]}^{\Phi'}\} & \\ = & [\![P]\!]_\rho^{x^A,\Phi'} \cup [\![Q]\!]_\rho^{x^A,\Phi'} & \end{array}$$

2. Por inducción en la estructura del contexto débil W.

- **Vacío, $\mathsf{W} = \square$.**

$$\begin{array}{rcl} [\![t]\!]_\rho & = & \{b \mid a \in [\![t]\!]_\rho, b \in \{a\}\} \\ & = & \{b \mid a \in [\![t]\!]_\rho, b \in [\![\square]\!]_{\rho[\square^A \mapsto a]}\} \end{array}$$

- **Izquierda de una aplicación, $\mathsf{W} = \mathsf{W}'\, s$.**

$$\begin{array}{cll} & [\![\mathsf{W}'\langle t\rangle\, s]\!]_\rho & \\ = & \{b \mid f \in [\![\mathsf{W}'\langle t\rangle]\!]_\rho, c \in [\![s]\!]_\rho, b \in f(c)\} & \\ = & \{b \mid a \in [\![t]\!]_\rho, f \in [\![\mathsf{W}']\!]_{\rho[\square^A\mapsto a]}, c \in [\![s]\!]_\rho, b \in f(c)\} & \text{(Por } h.i.\text{)} \\ = & \{b \mid a \in [\![t]\!]_\rho, f \in [\![\mathsf{W}']\!]_{\rho[\square^A\mapsto a]}, c \in [\![s]\!]_{\rho[\square^A\mapsto a]}, b \in f(c)\} & \text{(Por Lema 4.2.1)} \\ = & \{b \mid a \in [\![t]\!]_\rho, b \in [\![\mathsf{W}'\, s]\!]_{\rho[\square^A\mapsto a]}\} & \end{array}$$

- **Derecha de una aplicación,** $\mathsf{W} = s\,\mathsf{W}'$**.**

$$
\begin{array}{ll}
& \llbracket s\,\mathsf{W}'\langle t\rangle\rrbracket_\rho \\
= & \{b \mid f \in \llbracket s\rrbracket_\rho, c \in \llbracket \mathsf{W}'\langle t\rangle\rrbracket_\rho, b \in f(c)\} \\
= & \{b \mid f \in \llbracket s\rrbracket_\rho, a \in \llbracket t\rrbracket_\rho, c \in \llbracket \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}, b \in f(c)\} \quad \text{(Por } h.i.\text{)} \\
= & \{b \mid a \in \llbracket t\rrbracket_\rho, f \in \llbracket s\rrbracket_{\rho[\square^A \mapsto a]}, c \in \llbracket \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}, b \in f(c)\} \quad \text{(Por Lema 4.2.1)} \\
= & \{b \mid a \in \llbracket t\rrbracket_\rho, b \in \llbracket s\,\mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}\}
\end{array}
$$

- **Izquierda de una unificación,** $\mathsf{W} = \mathsf{W}' \stackrel{\bullet}{=} s$**.**

$$
\begin{array}{ll}
& \llbracket \mathsf{W}'\langle t\rangle \stackrel{\bullet}{=} s\rrbracket_\rho \\
= & \{\mathsf{R}_{\mathbf{ok}} \mid c \in \llbracket \mathsf{W}'\langle t\rangle\rrbracket_\rho, d \in \llbracket s\rrbracket_\rho, c = d\} \\
= & \{\mathsf{R}_{\mathbf{ok}} \mid a \in \llbracket t\rrbracket_\rho, c \in \llbracket \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}, d \in \llbracket s\rrbracket_\rho, c = d\} \quad \text{(Por } h.i.\text{)} \\
= & \{\mathsf{R}_{\mathbf{ok}} \mid a \in \llbracket t\rrbracket_\rho, c \in \llbracket \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}, d \in \llbracket s\rrbracket_{\rho[\square^A \mapsto a]}, c = d\} \quad \text{(Por Lema 4.2.1)} \\
= & \{b \mid a \in \llbracket t\rrbracket_\rho, b \in \llbracket \mathsf{W}' \stackrel{\bullet}{=} s\rrbracket_{\rho[\square^A \mapsto a]}\}
\end{array}
$$

- **Derecha de una unificación,** $\mathsf{W} = s \stackrel{\bullet}{=} \mathsf{W}'$**.**

$$
\begin{array}{ll}
& \llbracket s \stackrel{\bullet}{=} \mathsf{W}'\langle t\rangle\rrbracket_\rho \\
= & \{\mathsf{R}_{\mathbf{ok}} \mid c \in \llbracket s\rrbracket_\rho, d \in \llbracket \mathsf{W}'\langle t\rangle\rrbracket_\rho, c = d\} \\
= & \{\mathsf{R}_{\mathbf{ok}} \mid a \in \llbracket t\rrbracket_\rho, c \in \llbracket s\rrbracket_\rho, d \in \llbracket \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}, c = d\} \quad \text{(Por } h.i.\text{)} \\
= & \{\mathsf{R}_{\mathbf{ok}} \mid a \in \llbracket t\rrbracket_\rho, c \in \llbracket s\rrbracket_{\rho[\square^A \mapsto a]}, d \in \llbracket \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}, c = d\} \quad \text{(Por Lema 4.2.1)} \\
= & \{b \mid a \in \llbracket t\rrbracket_\rho, b \in \llbracket s \stackrel{\bullet}{=} \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}\}
\end{array}
$$

- **Izquierda de una secuencia,** $\mathsf{W} = \mathsf{W}'; s$**.**

$$
\begin{array}{ll}
& \llbracket \mathsf{W}'\langle t\rangle; s\rrbracket_\rho \\
= & \{b \mid c \in \llbracket \mathsf{W}'\langle t\rangle\rrbracket_\rho, b \in \llbracket s\rrbracket_\rho\} \\
= & \{b \mid a \in \llbracket t\rrbracket_\rho, c \in \llbracket \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}, b \in \llbracket s\rrbracket_\rho\} \quad \text{(Por } h.i.\text{)} \\
= & \{b \mid a \in \llbracket t\rrbracket_\rho, c \in \llbracket \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}, b \in \llbracket s\rrbracket_{\rho[\square^A \mapsto a]}\} \quad \text{(Por Lema 4.2.1)} \\
= & \{b \mid a \in \llbracket t\rrbracket_\rho, b \in \llbracket \mathsf{W}'; s\rrbracket_{\rho[\square^A \mapsto a]}\}
\end{array}
$$

- **Derecha de una secuencia,** $\mathsf{W} = s; \mathsf{W}'$**.**

$$
\begin{array}{ll}
& \llbracket s; \mathsf{W}'\langle t\rangle\rrbracket_\rho \\
= & \{b \mid c \in \llbracket s\rrbracket_\rho, b \in \llbracket \mathsf{W}'\langle t\rangle\rrbracket_\rho\} \\
= & \{b \mid a \in \llbracket t\rrbracket_\rho, c \in \llbracket s\rrbracket_\rho, b \in \llbracket \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}\} \quad \text{(Por } h.i.\text{)} \\
= & \{b \mid a \in \llbracket t\rrbracket_\rho, c \in \llbracket s\rrbracket_{\rho[\square^A \mapsto a]}, b \in \llbracket \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}\} \quad \text{(Por Lema 4.2.1)} \\
= & \{b \mid a \in \llbracket t\rrbracket_\rho, b \in \llbracket s; \mathsf{W}'\rrbracket_{\rho[\square^A \mapsto a]}\}
\end{array}
$$

□

Lema 4.2.3 (Variables libres)**.** *Valen las siguientes:*

1. $\mathsf{fv}(P \oplus Q) = \mathsf{fv}(P) \cup \mathsf{fv}(Q)$
2. $\mathsf{fv}(\mathsf{W}\langle t\rangle) = \mathsf{fv}(\mathsf{W}) \cup \mathsf{fv}(t)$
3. $\mathsf{fv}(\mathsf{W}\langle P\rangle) = \mathsf{fv}(\mathsf{W}) \cup \mathsf{fv}(P)$

4. $\mathsf{fv}(t^\sigma) \subseteq (\mathsf{fv}(t) \setminus \operatorname{supp}\sigma) \cup \bigcup_{x\in\operatorname{supp}\sigma} \mathsf{fv}(\sigma(x))$

5. $\mathsf{fv}(P^\sigma) \subseteq (\mathsf{fv}(P) \setminus \operatorname{supp}\sigma) \cup \bigcup_{x\in\operatorname{supp}\sigma} \mathsf{fv}(\sigma(x))$

Demostración. Por inducción en P, W, o t, según corresponda. □

Lema 4.2.4 (Interpretación de valores)**.** *Si* $\mathtt{v}$ *es un valor entonces* $[\![\mathtt{v}]\!]_\rho$ *es un conjunto de un solo elemento.*

Demostración. Por inducción en $\mathtt{v}$. Si $\mathtt{v}$ es una variable o una abstracción alojada, es inmediato, así que sea $\mathtt{v} = \mathbf{c}\,\mathtt{v}_1 \ldots \mathtt{v}_n$. En este caso, por inducción en n, afirmamos que $[\![\mathbf{c}\,\mathtt{v}_1 \ldots \mathtt{v}_n]\!]_\rho$ es un conjunto de un solo elemento de la forma $\{a\}$, donde además a es unitario:

1. **Si** $n = 0$**.** Entonces $[\![\mathbf{c}]\!]_\rho = \{\mathsf{R}_\mathbf{c}\}$, que es un conjunto unitario. Además, recordemos que siempre se pide que $\mathsf{R}_\mathbf{c}$ sea unitario.

2. **Si** $n > 0$**.** Entonces por *h.i.* en la inducción más interna $[\![\mathbf{c}\,\mathtt{v}_1 \ldots \mathtt{v}_{n-1}]\!]_\rho$ es un conjunto unitario de la forma $\{f_0\}$, donde f_0 es unitario, y por *h.i.* en la inducción más externa $[\![\mathtt{v}_n]\!]_\rho$ es un conjunto unitario de la forma $\{a_0\}$, entonces tenemos que:

$$\begin{aligned} [\![\mathbf{c}\,\mathtt{v}_1 \ldots \mathtt{v}_{n-1}\,\mathtt{v}_n]\!]_\rho &= \{b \mid f \in [\![\mathbf{c}\,\mathtt{v}_1 \ldots \mathtt{v}_{n-1}]\!]_\rho, a \in [\![\mathtt{v}_n]\!]_\rho, b \in f(a)\} \\ &= f_0(a_0) \end{aligned}$$

   Como f_0 es unitario, $f_0(a_0)$ es un conjunto unitario de la forma $\{b\}$, donde b es unitario, como se pide.

□

Lema 4.2.5 (Interpretación de las sustituciones)**.** *Sea* $\sigma = \{x_1^{A_1} \mapsto \mathtt{v}_1, \ldots, x_n^{A_n} \mapsto \mathtt{v}_n\}$ *una sustitución con soporte* $\{x_1^{A_1}, \ldots, x_n^{A}\}$ *y tal que* $x_i \notin \mathsf{fv}(\mathtt{v}_j)$ *para* $1 \le i, j \le n$ *cualesquiera. Recordemos que la interpretación de un valor es siempre un conjunto unitario (Lema 4.2.4), así que sea* $[\![\mathtt{v}_i]\!]_\rho = \{a_i\}$ *para cada* $i = 1..n$*. Entonces:*

1. $[\![t^\sigma]\!]_\rho = [\![t]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}$

2. $[\![P^\sigma]\!]_\rho = [\![P]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}$

Demostración. Por inducción mutua en el término t (resp. programa P).

1. **Variable,** $t = x^A$**.** Hay dos subcasos, dependiendo si $x \in \{x_1, \ldots, x_n\}$ o no.

   1.1 Si $x = x_i$ para algún $1 \le i \le n$, entonces:

$$[\![(x_i^A)^\sigma]\!]_\rho = [\![\mathtt{v}_i]\!]_\rho = \{a_i\} = [\![x_i^A]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}$$

   1.2 Si $x \notin \{x_1, \ldots, x_n\}$, entonces:

$$[\![(x^A)^\sigma]\!]_\rho = \rho(x^A) = [\![x^A]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}$$

2. **Constructor,** $t = \mathbf{c}$**.** Inmediato, ya que:

$$[\![\mathbf{c}^\sigma]\!]_\rho = [\![\mathbf{c}]\!]_\rho = \{\mathsf{R}_\mathbf{c}\} = [\![\mathbf{c}]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}$$

3. **Abstracción, $t = \lambda x^A. P$.** Entonces:

$$\begin{array}{rcll} [\![(\lambda x^A. P)^\sigma]\!]_\rho & = & [\![\lambda x^A. P^\sigma]\!]_\rho & \\ & = & \{f\} \quad \text{donde } f(a) = [\![P^\sigma]\!]_{\rho[x\mapsto a]} & \\ & = & \{g\} \quad \text{donde } g(a) = [\![P]\!]_{\rho[x\mapsto a][x_1\mapsto a_1]\ldots[x_n\mapsto a_n]} & \text{(Por } h.i.) \\ & = & [\![\lambda x^A. P]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]} & \end{array}$$

4. **Abstracción alojada, $t = \lambda^\ell x. P$.** Análogo al caso anterior.

5. **Aplicación, $t = s\,u$.** Entonces:

$$\begin{array}{rcll} [\![(s\,u)^\sigma]\!]_\rho & = & [\![s^\sigma\,u^\sigma]\!]_\rho & \\ & = & \{b \mid f \in [\![s^\sigma]\!]_\rho, a \in [\![u^\sigma]\!]_\rho, b \in f(a)\} & \\ & = & \{b \mid f \in [\![s]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}, a \in [\![u]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}, b \in f(a)\} & \text{(Por } h.i.) \\ & = & [\![s\,u]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]} & \end{array}$$

6. **Unificación, $t = (s \doteq u)$.** Entonces:

$$\begin{array}{rcll} [\![(s \doteq u)^\sigma]\!]_\rho & = & [\![s^\sigma \doteq u^\sigma]\!]_\rho & \\ & = & \{\mathsf{R}_{\mathbf{ok}} \mid a \in [\![s^\sigma]\!]_\rho, b \in [\![u^\sigma]\!]_\rho, a = b\} & \\ & = & \{\mathsf{R}_{\mathbf{ok}} \mid a \in [\![s]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}, b \in [\![u]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}, a = b\} & \text{(Por } h.i.) \\ & = & [\![s \doteq u]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]} & \end{array}$$

7. **Secuencia, $t = s; u$.** Entonces:

$$\begin{array}{rcll} [\![(s;u)^\sigma]\!]_\rho & = & [\![s^\sigma;u^\sigma]\!]_\rho & \\ & = & \{a \mid b \in [\![s^\sigma]\!]_\rho, a \in [\![u^\sigma]\!]_\rho\} & \\ & = & \{a \mid b \in [\![s]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}, a \in [\![u]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}\} & \text{(Por } h.i.) \\ & = & [\![s;u]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]} & \end{array}$$

8. **Variable fresca, $t = \nu x^A. s$.** Entonces:

$$\begin{array}{rcll} [\![(\nu x^A. s)^\sigma]\!]_\rho & = & [\![\nu x^A. s^\sigma]\!]_\rho & \\ & = & \{b \mid a \in [\![A]\!], b \in [\![s^\sigma]\!]_{\rho[x\mapsto a]}\} & \\ & = & \{b \mid a \in [\![A]\!], b \in [\![s]\!]_{\rho[x\mapsto a][x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}\} & \text{(Por } h.i.) \\ & = & \{b \mid a \in [\![A]\!], b \in [\![s]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n][x\mapsto a]}\} & \text{(Ya que } x \notin \{x_1, \ldots, x_n\}) \\ & = & [\![\nu x^A. s]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]} & \end{array}$$

9. **Fail, $P = \texttt{fail}$.** Inmediato, ya que:

$$[\![\texttt{fail}^\sigma]\!]_\rho = [\![\texttt{fail}]\!]_\rho = \varnothing = [\![\texttt{fail}]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]}$$

10. **Alternativa, $P = t \oplus P$.** Entonces:

$$\begin{array}{rcll} [\![(t \oplus P)^\sigma]\!]_\rho & = & [\![t^\sigma \oplus P^\sigma]\!]_\rho & \\ & = & [\![t^\sigma]\!]_\rho \cup [\![P^\sigma]\!]_\rho & \\ & = & [\![t]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]} \cup [\![P]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]} & \text{(Por } h.i.) \\ & = & [\![t \oplus P]\!]_{\rho[x_1\mapsto a_1]\ldots[x_n\mapsto a_n]} & \end{array}$$

□

Definición 4.2.6 (Satisfacibilidad de ecuaciones)**.** Sea ρ una asignación fija de variables, y sea $\vec{x}^{\vec{A}}$ una secuencia fija de variables. Además, sea $\mathcal{E} = \{(\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{w}_1), \dots, (\mathtt{v}_n \stackrel{\bullet}{=} \mathtt{w}_n)\}$ un problema de unificación. Dada una secuencia de objetos $\vec{a} \in [\![\vec{A}]\!]$ decimos que $\vec{a}$ *satisface* $\mathcal{E}$ (con respecto a $\rho, \vec{x}$), notado como $\vec{a} \vDash_{\rho,\vec{x}} \mathcal{E}$, si y sólo si $[\![\mathtt{v}_i]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = [\![\mathtt{w}_i]\!]_{\rho[\vec{x}\mapsto\vec{a}]}$ para todo $i = 1..n$. Escribimos $\vec{a} \vDash \mathcal{E}$ si ρ y $\vec{x}$ se deducen del contexto.

Lema 4.2.7 (Unificación preserva satisfacibilidad)**.** *Sea $\mathcal{E} \rightsquigarrow \mathcal{H}$ un paso del algoritmo de unificación que no falla. Entonces, para cada $\rho, \vec{x}^{\vec{A}}$ tenemos que:*

$$\{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \mathcal{E}\} = \{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \mathcal{H}\}$$

Demostración. Notemos que el paso no falla, por lo que no puede ser el resultado de aplicar la regla `u-clash` o la regla `u-occurs-check`. Consideramos los cinco casos restantes:

1. `u-delete`: Nuestro objetivo es probar que:

$$\{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \{y \stackrel{\bullet}{=} y\} \uplus \mathcal{E}'\} = \{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \mathcal{E}'\}$$

   Esto es inmediato pues $[\![y]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = [\![y]\!]_{\rho[\vec{x}\mapsto\vec{a}]}$ siempre vale.

2. `u-orient`: Nuestro objetivo es probar que:

$$\{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \{\mathtt{v} \stackrel{\bullet}{=} y\} \uplus \mathcal{E}'\} = \{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \{y \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E}'\}$$

   Esto es inmediato por definición.

3. `u-match-lam`: Nuestro objetivo es probar que:

$$\{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \{\lambda^\ell y.\, P \stackrel{\bullet}{=} \lambda^\ell y.\, P\} \uplus \mathcal{E}'\} = \{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \mathcal{E}'\}$$

   Esto es inmediato pues $[\![\lambda^\ell y.\, P]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = [\![\lambda^\ell y.\, P]\!]_{\rho[\vec{x}\mapsto\vec{a}]}$ siempre vale.

4. `u-match-cons`: Nuestro objetivo es probar que:

$$\{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \{\mathbf{c}\,\mathtt{v}_1 \dots \mathtt{v}_n \stackrel{\bullet}{=} \mathbf{c}\,\mathtt{w}_1 \dots \mathtt{w}_n\} \uplus \mathcal{E}'\} = \{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \{\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{w}_1, \dots, \mathtt{v}_n \stackrel{\bullet}{=} \mathtt{w}_n\} \uplus \mathcal{E}'\}$$

   Recordemos que $[\![\mathbf{c}]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = \{\mathsf{R}_\mathbf{c}\}$ es $\mathcal{T}_\mathbf{c}$-unitario, y que la interpretación de un valor siempre es un conjunto unitario (Lema 4.2.4), entonces sea $[\![\mathtt{v}_i]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = \{b_i\}$ y $[\![\mathtt{w}_i]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = \{b'_i\}$. Basta notar que:

$$\begin{array}{ll} & \vec{a} \vDash_{\rho,\vec{x}} \{\mathbf{c}\,\mathtt{v}_1 \dots \mathtt{v}_n \stackrel{\bullet}{=} \mathbf{c}\,\mathtt{w}_1 \dots \mathtt{w}_n\} \\ \iff & [\![\mathbf{c}\,\mathtt{v}_1 \dots \mathtt{v}_n]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = [\![\mathbf{c}\,\mathtt{w}_1 \dots \mathtt{w}_n]\!]_{\rho[\vec{x}\mapsto\vec{a}]} \\ \iff & \mathsf{R}_\mathbf{c}(b_1)\dots(b_n) = \mathsf{R}_\mathbf{c}(b'_1)\dots(b'_n) \\ \iff & b_i = b'_i \text{ para todo } i = 1..n \qquad (\star) \\ \iff & [\![\mathtt{v}_i]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = [\![\mathtt{w}_i]\!]_{\rho[\vec{x}\mapsto\vec{a}]}, \text{ para todo } i = 1..n \\ \iff & \vec{a} \vDash_{\rho,\vec{x}} \{\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{w}_1, \dots, \mathtt{v}_n \stackrel{\bullet}{=} \mathtt{w}_n\} \end{array}$$

   El paso $(\star)$ se justifica por el hecho de que asumimos el principio de **no confusión** en las interpretaciones de los constructores.

5. `u-eliminate`: Nuestro objetivo es probar que:

$$\{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \{y \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E}'\} = \{\vec{a} \mid \vec{a} \vDash_{\rho,\vec{x}} \{y \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E}'\{y := \mathtt{v}\}\}$$

si $y \in \mathsf{fv}(\mathcal{E}') \setminus \mathsf{fv}(\mathtt{v})$. Además, sea $\mathcal{E}' = \{(\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{w}_1), \dots, (\mathtt{v}_n \stackrel{\bullet}{=} \mathtt{w}_n)\}$. Recordemos que la interpretación de un valor es siempre un conjunto unitario (Lema 4.2.4), entonces sea $[\![\mathtt{v}]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = \{b\}$. Sea $\vec{a} \in [\![\vec{A}]\!]$. Basta con demostrar que siempre que $\rho[\vec{x} \mapsto \vec{a}](y) = b$ entonces vale la siguiente equivalencia:

$$\vec{a} \vDash_{\rho,\vec{x}} \mathcal{E}' \iff \vec{a} \vDash_{\rho,\vec{x}} \mathcal{E}'\{y := \mathtt{v}\}$$

Notemos que, para cada $i = 1..n$ fijo:

$$\begin{array}{rcll} [\![\mathtt{v}_i]\!]_{\rho[\vec{x}\mapsto\vec{a}]} & = & [\![\mathtt{v}_i]\!]_{\rho[\vec{x}\mapsto\vec{a}][y\mapsto b]} & (\star) \\ & = & [\![\mathtt{v}_i\{y := \mathtt{v}\}]\!]_{\rho[\vec{x}\mapsto\vec{a}]} & \text{(Por Lema 4.2.5)} \end{array}$$

El paso $(\star)$ es trivial porque, como ya notamos, $\rho[\vec{x} \mapsto \vec{a}](y) = b$ por lo que $\rho[\vec{x} \mapsto \vec{a}]$ y $\rho[\vec{x} \mapsto \vec{a}][y \mapsto b]$ son la misma asignación de variables. Y similarmente, $[\![\mathtt{w}_i]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = [\![\mathtt{w}_i\{y := \mathtt{v}\}]\!]_{\rho[\vec{x}\mapsto\vec{a}]}$. Entonces:

$$\begin{array}{rll} & \vec{a} \vDash_{\rho,\vec{x}} \mathcal{E}' & \\ \iff & [\![\mathtt{v}_i]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = [\![\mathtt{w}_i]\!]_{\rho[\vec{x}\mapsto\vec{a}]} \text{ para todo } i = 1..n & \\ \iff & [\![\mathtt{v}_i\{y := \mathtt{v}\}]\!]_{\rho[\vec{x}\mapsto\vec{a}]} = [\![\mathtt{w}_i\{y := \mathtt{v}\}]\!]_{\rho[\vec{x}\mapsto\vec{a}]} \text{ para todo } i = 1..n & \text{(Por Lema 4.2.5.)} \\ \iff & \vec{a} \vDash_{\rho,\vec{x}} \mathcal{E}'\{y := \mathtt{v}\} & \end{array}$$

□

Teorema 4.2.8 (Correctitud débil)**.** *Sea $\Gamma \vdash P : A$ y $P \to Q$. Sea $\Phi = \mathsf{fv}(P)$ y $\Phi' = \mathsf{fv}(Q)$. Entonces, para cada asignación de variables ρ:*

$$[\![P]\!]^{\Phi}_{\rho} \supseteq [\![Q]\!]^{\Phi'}_{\rho}$$

Además, la inclusión es una igualdad para cualquier regla de reducción a excepción de la regla `fail`.

Demostración. Sea $P \to Q$. Consideramos seis casos, dependiendo de la regla que se aplica para concluir que $P \to Q$:

1. `alloc`: Notemos que $\Phi = \Phi'$, y supongamos que $\Phi = \vec{y}^{\vec{B}}$. Entonces:

$$\begin{array}{rcll} [\![P_1 \oplus \mathsf{W}\langle \lambda x^A.\, Q\rangle \oplus P_2]\!]^{\Phi}_{\rho} & = & \{a \mid \vec{b} \in [\![\vec{B}]\!], a \in [\![P_1 \oplus \mathsf{W}\langle \lambda x^A.\, Q\rangle \oplus P_2]\!]_{\rho[\vec{y}\mapsto\vec{b}]}\} & \\ & = & \{a \mid \vec{b} \in [\![\vec{B}]\!], a \in [\![P_1 \oplus \mathsf{W}\langle \lambda^{\ell} x^A.\, Q\rangle \oplus P_2]\!]_{\rho[\vec{y}\mapsto\vec{b}]}\} & (\star) \\ & = & [\![P_1 \oplus \mathsf{W}\langle \lambda^{\ell} x^A.\, Q\rangle \oplus P_2]\!]^{\Phi}_{\rho} & \end{array}$$

Para justificar el paso $(\star)$, notemos que, por Composicionalidad (Lema 4.2.2), basta probar que $[\![\lambda x^A.\, Q]\!]_{\rho[\vec{y}\mapsto\vec{b}]} = [\![\lambda^{\ell} x^A.\, Q]\!]_{\rho[\vec{y}\mapsto\vec{b}]}$ para todo $\vec{b} \in [\![B]\!]$. Esto vale por definición, por lo que terminamos de probar este caso.

2. `beta`: Notemos que $\Phi = \Phi', \vec{z}^{\vec{C}}$ donde

$$\vec{z} = \begin{cases} \varnothing & \text{si } x \notin \mathsf{fv}(Q) \\ \mathsf{fv}(\mathtt{v}) \setminus \mathsf{fv}(P_1 \oplus \mathsf{W}\langle(\lambda x.\, Q)\Box\rangle \oplus P_2) & \text{si } x \in \mathsf{fv}(Q) \end{cases}$$

Además, supongamos que $\Phi' = \vec{y}^{\vec{B}}$. Entonces:

$$\begin{array}{cl} & [\![P_1 \oplus \mathsf{W}\langle(\lambda^\ell x^A.\, Q)\, \mathtt{v}\rangle \oplus P_2]\!]_\rho^{\vec{y}^{\vec{B}}, \vec{z}^{\vec{C}}} \\ = & \{a \mid \vec{b} \in [\![\vec{B}]\!], \vec{c} \in [\![\vec{C}]\!], a \in [\![P_1 \oplus \mathsf{W}\langle(\lambda^\ell x^A.\, Q)\, \mathtt{v}\rangle \oplus P_2]\!]_{\rho[\vec{y} \mapsto \vec{b}][\vec{z} \mapsto \vec{c}]}\} \\ = & \{a \mid \vec{b} \in [\![\vec{B}]\!], a \in [\![P_1 \oplus \mathsf{W}\langle Q\{x^A := \mathtt{v}\}\rangle \oplus P_2]\!]_{\rho[\vec{y} \mapsto \vec{b}]}\} \qquad (\star) \\ = & [\![P_1 \oplus \mathsf{W}\langle Q\{x^A := \mathtt{v}\}\rangle \oplus P_2]\!]_\rho^{\vec{y}^{\vec{B}}} \end{array}$$

Para justificar el paso $(\star)$, procedemos como sigue. Escribamos $\rho[\vec{y} \mapsto \vec{b}]$ como ρ'. Recordemos que la interpretación de un valor es siempre un conjunto unitario (Lema 4.2.4), así que sea $[\![\mathtt{v}]\!]_{\rho'[\vec{z} \mapsto \vec{c}]} = \{a_0\}$. Por Composicionalidad (Lema 4.2.2) basta notar que:

$$\begin{array}{cll} & [\![(\lambda^\ell x^A.\, Q)\, \mathtt{v}]\!]_{\rho'[\vec{z} \mapsto \vec{c}]} & \\ = & \{b \mid a \in [\![\mathtt{v}]\!]_{\rho'[\vec{z} \mapsto \vec{c}]}, b \in [\![Q]\!]_{\rho'[\vec{z} \mapsto \vec{c}][x^A \mapsto a]}\} & \\ = & [\![Q]\!]_{\rho'[\vec{z} \mapsto \vec{c}][x^A \mapsto a_0]} & \\ = & [\![Q]\!]_{\rho'[x^A \mapsto a_0]} & \text{(Por Irrelevancia (Lema 4.2.1))} \\ = & [\![Q\{x^A := \mathtt{v}\}]\!]_{\rho'} & \text{(Por Lema 4.2.5)} \end{array}$$

3. `seq`: Notemos que $\Phi = \Phi', \vec{z}^{\vec{C}}$, donde $\vec{z}^{\vec{C}} = \mathsf{fv}(\mathtt{v}) \setminus \mathsf{fv}(P_1 \oplus \mathsf{W}\langle\Box; t\rangle \oplus P_2)$. Supongamos que $\Phi' = \vec{y}^{\vec{B}}$. Entonces:

$$\begin{array}{cl} & [\![P_1 \oplus \mathsf{W}\langle\mathtt{v}; t\rangle \oplus P_2]\!]_\rho^{\vec{y}^{\vec{B}}, \vec{z}^{\vec{C}}} \\ = & \{a \mid \vec{b} \in [\![\vec{B}]\!], \vec{c} \in [\![\vec{C}]\!], a \in [\![P_1 \oplus \mathsf{W}\langle\mathtt{v}; t\rangle \oplus P_2]\!]_{\rho[\vec{y} \mapsto \vec{b}][\vec{z} \mapsto \vec{c}]}\} \\ = & \{a \mid \vec{b} \in [\![\vec{B}]\!], a \in [\![P_1 \oplus \mathsf{W}\langle t\rangle \oplus P_2]\!]_{\rho[\vec{y} \mapsto \vec{b}]}\} \qquad (\star) \\ = & [\![P_1 \oplus \mathsf{W}\langle t\rangle \oplus P_2]\!]_\rho^{\vec{y}^{\vec{B}}} \end{array}$$

Para justificar el paso $(\star)$ procedemos como sigue. Escribamos $\rho[\vec{y} \mapsto \vec{b}]$ como ρ'. Recordemos que la interpretación de un valor es siempre un conjunto unitario (Lema 4.2.4), así que sea $[\![\mathtt{v}]\!]_{\rho'[\vec{z} \mapsto \vec{c}]} = \{b_0\}$. Por Composicionalidad (Lema 4.2.2) basta notar que:

$$\begin{array}{rcll} [\![\mathtt{v}; t]\!]_{\rho'[\vec{z} \mapsto c]} & = & \{a \mid b \in [\![\mathtt{v}]\!]_{\rho'[\vec{z} \mapsto \vec{c}]}, a \in [\![t]\!]_{\rho'[\vec{z} \mapsto \vec{c}]}\} & \text{(Por Irrelevancia Lema 4.2.1)} \\ & = & [\![t]\!]_{\rho'} & \end{array}$$

4. `fresh`: Notemos que $\Phi' = \Phi, y^A$ donde y es una variable fresca. Supongamos que $\Phi = \vec{z}^{\vec{B}}$. Entonces:

$$\begin{array}{cl} & [\![P_1 \oplus \mathsf{W}\langle\nu x^A.\, t\rangle \oplus P_2]\!]_\rho^\Phi \\ = & \{a \mid \vec{b} \in [\![\vec{B}]\!], a \in [\![P_1 \oplus \mathsf{W}\langle\nu x^A.\, t\rangle \oplus P_2]\!]_{\rho[\vec{z} \mapsto \vec{b}]}\} \\ = & \{a \mid \vec{b} \in [\![\vec{B}]\!], a \in [\![P_1 \oplus \mathsf{W}\langle t\{x^A := y^A\}\rangle \oplus P_2]\!]_{\rho[\vec{z} \mapsto \vec{b}]}^{y^A}\} \quad (\star) \\ = & [\![P_1 \oplus \mathsf{W}\langle t\{x^A := y^A\}\rangle \oplus P_2]\!]_\rho^{\Phi, y^A} \end{array}$$

Para justificar el paso ($\star$) procedemos como sigue. Sea ρ' una abreviatura para $\rho[\vec{z} \mapsto \vec{b}]$. Por Irrelevancia (Lema 4.2.1), $[\![P_1]\!]_{\rho'} = [\![P_1]\!]^{y^A}_{\rho'}$. Análogamente, $[\![P_2]\!]_{\rho'} = [\![P_2]\!]^{y^A}_{\rho'}$. Por Composicionalidad (Lema 4.2.2), basta mostrar que $[\![\mathsf{W}\langle \nu x^A.\, t\rangle]\!]_{\rho'} = [\![\mathsf{W}\langle t\{x^A := y^A\}\rangle]\!]^{y^A}_{\rho'}$. En efecto:

$$\begin{array}{lll}
& [\![\mathsf{W}\langle \nu x^A.\, t\rangle]\!]_{\rho'} & \\
= & \{c \mid b \in [\![\nu x^A.\, t]\!]_{\rho'}, c \in [\![\mathsf{W}]\!]_{\rho'[\Box \mapsto b]}\} & \text{(Por Lema 4.2.2)} \\
= & \{c \mid a \in [\![A]\!], b \in [\![t]\!]_{\rho'[x^A \mapsto a]}, c \in [\![\mathsf{W}]\!]_{\rho'[\Box \mapsto b]}\} & \\
= & \{c \mid a \in [\![A]\!], b \in [\![t\{x^A := y^A\}]\!]_{\rho'[y^A \mapsto a]}, c \in [\![\mathsf{W}]\!]_{\rho'[\Box \mapsto b]}\} & \text{(Por Lema 4.2.5} \\
& & \text{y Lema 4.2.1)} \\
= & \{c \mid a \in [\![A]\!], b \in [\![t\{x^A := y^A\}]\!]_{\rho'[y^A \mapsto a]}, c \in [\![\mathsf{W}]\!]_{\rho'[y^A \mapsto a][\Box \mapsto b]}\} & \text{(Por Lema 4.2.1)} \\
= & \{c \mid a \in [\![A]\!], c \in [\![\mathsf{W}\langle t\{x^A := y^A\}\rangle]\!]_{\rho'[y^A \mapsto a]}\} & \text{(Por Lema 4.2.2)} \\
= & [\![\mathsf{W}\langle t\{x^A := y^A\}\rangle]\!]^{y^A}_{\rho'} &
\end{array}$$

5. `unif`: Nuestro objetivo es probar que $[\![P_1 \oplus \mathsf{W}\langle \mathtt{v} \overset{\bullet}{=} \mathtt{w}\rangle \oplus P_2]\!]^{\Phi}_{\rho} = [\![P_1 \oplus \mathsf{W}\langle \mathbf{ok}\rangle^{\sigma} \oplus P_2]\!]^{\Phi'}_{\rho}$, donde $\sigma = \mathsf{mgu}(\{\mathtt{v} \overset{\bullet}{=} \mathtt{w}\})$. Notemos que Φ' es un subconjunto de Φ, entonces supongamos que $\Phi = \Phi', \vec{y}^{\vec{B}}$ y $\Phi' = \vec{x}^{\vec{A}}$. Notemos también que $\sigma = \mathsf{mgu}(\mathtt{v} \overset{\bullet}{=} \mathtt{w})$ existe, entonces $\{\mathtt{v} \overset{\bullet}{=} \mathtt{w}\} \rightsquigarrow^* \{x_1 \overset{\bullet}{=} \mathtt{v}_1, \ldots, x_n \overset{\bullet}{=} \mathtt{v}_n\}$ tal que $x_i \notin \mathsf{fv}(\mathtt{v}_j)$ for all i, j, y el unificador más general es $\sigma = \{x_1 \mapsto \mathtt{v}_1, \ldots, x_n \mapsto \mathtt{v}_n\}$. Más aún, recordemos que la interpretación de un valor siempre es un conjunto unitario (Lema 4.2.4), por lo que para cada asignación fija ρ' escribamos $b^{\rho'}_i$ para el único elemento en $[\![\mathtt{v}_i]\!]_{\rho'}$. Además, sea $\vec{z}^{\vec{C}} = \vec{x}^{\vec{A}}, \vec{y}^{\vec{B}}$. Por Composicionalidad (Lema 4.2.2) e Irrelevancia (Lema 4.2.1), basta notar que:

$$\begin{array}{lll}
& [\![\mathsf{W}\langle \mathtt{v} \overset{\bullet}{=} \mathtt{w}\rangle]\!]^{\vec{z}^{\vec{C}}}_{\rho} & \\
= & \{a \mid \vec{c} \in [\![\vec{C}]\!], a \in [\![\mathsf{W}\langle \mathtt{v} \overset{\bullet}{=} \mathtt{w}\rangle]\!]_{\rho[\vec{z} \mapsto \vec{c}]}\} & \\
= & \{a \mid \vec{c} \in [\![\vec{C}]\!], b \in [\![\mathtt{v} \overset{\bullet}{=} \mathtt{w}]\!]_{\rho[\vec{z} \mapsto \vec{c}]}, a \in [\![\mathsf{W}]\!]_{\rho[\vec{z} \mapsto \vec{c}][\Box \mapsto b]}\} & \text{(Lema 4.2.2)} \\
= & \{a \mid \vec{c} \in [\![\vec{C}]\!], b \in [\![\mathtt{v} \overset{\bullet}{=} \mathtt{w}]\!]_{\rho[\vec{z} \mapsto \vec{c}]}, a \in [\![\mathsf{W}]\!]_{\rho[\vec{z} \mapsto \vec{c}][\Box \mapsto \mathsf{R}_{\mathbf{ok}}]}\} & \\
= & \{a \mid \vec{c} \in [\![\vec{C}]\!], \vec{c} \vDash_{\rho, \vec{z}} \{\mathtt{v} \overset{\bullet}{=} \mathtt{w}\}, a \in [\![\mathsf{W}]\!]_{\rho[\vec{z} \mapsto \vec{c}][\Box \mapsto \mathsf{R}_{\mathbf{ok}}]}\} & \\
= & \{a \mid \vec{c} \in [\![\vec{C}]\!], \vec{c} \vDash_{\rho, \vec{z}} \{x_1 \overset{\bullet}{=} \mathtt{v}_1, \ldots, x_n \overset{\bullet}{=} \mathtt{v}_n\}, a \in [\![\mathsf{W}]\!]_{\rho[\vec{z} \mapsto \vec{c}][\Box \mapsto \mathsf{R}_{\mathbf{ok}}]}\} & \text{(Lema 4.2.7)} \\
= & \{a \mid \vec{c} \in [\![\vec{C}]\!], \rho[\vec{z} \mapsto \vec{c}](x_i) = b^{\rho[\vec{z} \mapsto \vec{c}]}_i \text{ for all } i,\ a \in [\![\mathsf{W}\langle \mathbf{ok}\rangle]\!]_{\rho[\vec{z} \mapsto \vec{c}]}\} & \text{(Lema 4.2.2)} \\
= & \{a \mid \vec{c} \in [\![\vec{C}]\!], a \in [\![\mathsf{W}\langle \mathbf{ok}\rangle]\!]_{\rho[\vec{z} \mapsto \vec{c}][x_1 \mapsto b^{\rho[\vec{z} \mapsto \vec{c}]}_1] \ldots [x_n \mapsto b^{\rho[\vec{z} \mapsto \vec{c}]}_n]}\} & (\star) \\
= & \{a \mid \vec{c} \in [\![\vec{C}]\!], a \in [\![\mathsf{W}\langle \mathbf{ok}\rangle^{\sigma}]\!]_{\rho[\vec{z} \mapsto \vec{c}]}\} & \text{(Lema 4.2.5)} \\
= & [\![\mathsf{W}\langle \mathbf{ok}\rangle^{\sigma}]\!]^{\vec{z}^{\vec{C}}}_{\rho} & \\
= & [\![\mathsf{W}\langle \mathbf{ok}\rangle^{\sigma}]\!]^{\vec{x}^{\vec{A}}}_{\rho} & \text{(Lema 4.2.1)}
\end{array}$$

Para justificar el paso ($\star$) notemos que $\rho[\vec{z} \mapsto \vec{c}](x_i) = \{b^{\rho[\vec{z} \mapsto \vec{c}]}_i\}$ para todo $i = 1..n$. Por lo tanto, podemos escribir $\rho[\vec{z} \mapsto \vec{c}]$ como $\rho[\vec{z} \mapsto \vec{c}][x_1 \mapsto b^{\rho[\vec{z} \mapsto \vec{c}]}_1] \ldots [x_n \mapsto b^{\rho[\vec{z} \mapsto \vec{c}]}_n]$.

6. `fail`: Nuestro objetivo es probar que:

$$[\![P_1 \oplus \mathsf{W}\langle \mathtt{v} \overset{\bullet}{=} \mathtt{w}\rangle \oplus P_2]\!]^{\Phi}_{\rho} \supseteq [\![P_1 \oplus P_2]\!]^{\Phi'}_{\rho}$$

que es inmediato por definición.

$\square$

Ejemplo 4.2.9. *Consideremos la reducción:*

$$\nu x.\, \Big((\lambda z.\, \nu y.\, ((z \stackrel{\bullet}{=} \mathbf{t}\,\mathbf{1}\,y); (\mathbf{t}\,y\,x)))\,(\mathbf{t}\,x\,\mathbf{2}) \Big) \to^{*} \mathbf{t}\,\mathbf{2}\,\mathbf{1}$$

Si $[\![\mathtt{Nat}]\!] = \mathbb{N}$*,* $[\![\mathtt{Tuple}]\!] = [\![\mathtt{Nat}]\!] \times [\![\mathtt{Nat}]\!] = \mathbb{N} \times \mathbb{N}$*, los constructores* $\mathbf{1} : \mathtt{Nat}$*,* $\mathbf{2} : \mathtt{Nat}$ *se interpretan de la manera obvia y el constructor* $\mathbf{t} : \mathtt{Nat} \to \mathtt{Nat} \to \mathtt{Tuple}$ *es la función constructora de pares, es decir:*

$$\begin{array}{rcl} \mathsf{R}_{\mathbf{1}} & \stackrel{\text{def}}{=} & 1 \\ \mathsf{R}_{\mathbf{2}} & \stackrel{\text{def}}{=} & 2 \\ \mathsf{R}_{\mathbf{t}}(n) & \stackrel{\text{def}}{=} & \{f_n\},\ \textit{donde}\ f_n(m) = \{(n,m)\} \end{array}$$

entonces para cualquier asignación de variables ρ*, si abreviamos* $\rho' := \rho[x \mapsto n][z \mapsto p][y \mapsto m]$*, tenemos que:*

$$\begin{array}{rl} & [\![\nu x.\, \Big((\lambda z.\, \nu y.\, ((z \stackrel{\bullet}{=} \mathbf{t}\,\mathbf{1}\,y); (\mathbf{t}\,y\,x)))\,(\mathbf{t}\,x\,\mathbf{2}) \Big)]\!]_{\rho} \\ = & \{[\![(\lambda z.\, \nu y.\, ((z \stackrel{\bullet}{=} \mathbf{t}\,\mathbf{1}\,y); (\mathbf{t}\,y\,x)))\,(\mathbf{t}\,x\,\mathbf{2})]\!]_{\rho[x\mapsto n]} \mid n \in \mathbb{N}\} \\ = & \{r \mid n \in \mathbb{N}, f \in [\![\lambda z.\, \nu y.\, ((z \stackrel{\bullet}{=} \mathbf{t}\,\mathbf{1}\,y); (\mathbf{t}\,y\,x))]\!]_{\rho[x\mapsto n]}, p \in [\![\mathbf{t}\,x\,\mathbf{2}]\!]_{\rho[x\mapsto n]}, r \in f(p)\} \\ = & \{r \mid n, m \in \mathbb{N}, p \in [\![\mathbf{t}\,x\,\mathbf{2}]\!]_{\rho[x\mapsto n]}, r \in [\![(z \stackrel{\bullet}{=} \mathbf{t}\,\mathbf{1}\,y); (\mathbf{t}\,y\,x)]\!]_{\rho'}\} \\ = & \{r \mid n, m \in \mathbb{N}, p \in \{(n,2)\}, r \in [\![(z \stackrel{\bullet}{=} \mathbf{t}\,\mathbf{1}\,y); (\mathbf{t}\,y\,x)]\!]_{\rho'}\} \\ = & \{r \mid n, m \in \mathbb{N}, p \in \{(n,2)\}, b \in [\![z \stackrel{\bullet}{=} \mathbf{t}\,\mathbf{1}\,y]\!]_{\rho'}, r \in [\![\mathbf{t}\,y\,x]\!]_{\rho'}\} \\ = & \{r \mid n, m \in \mathbb{N}, p \in \{(n,2)\}, p = (1,m), r \in [\![\mathbf{t}\,y\,x]\!]_{\rho'}\} \\ = & \{r \mid n \in \{1\}, m \in \{2\}, p \in \{(1,2)\}, r \in [\![\mathbf{t}\,y\,x]\!]_{\rho'}\} \\ = & \{(2,1)\} \\ = & [\![\mathbf{t}\,\mathbf{2}\,\mathbf{1}]\!]_{\rho} \end{array}$$

5. LIMITACIONES DE LA SEMÁNTICA DENOTACIONAL PROPUESTA

Este capítulo está dedicado a discutir las falencias que presenta la semántica denotacional que proponemos para el cálculo-$\lambda^{\mathtt{U}}$. En particular, mostramos que con la semántica propuesta fallan las propiedades de correctitud y completitud de la semántica operacional con respecto a la denotacional.

Como parte del trabajo de esta tesis, estudiamos dos estrategias alternativas, con la intención de definir una semántica denotacional para la cual valieran las propiedades de correctitud y completitud. Dado que estos dos intentos no fueron exitosos, sólo incluimos una discusión informal, evitando los detalles técnicos.

En primer lugar, una falencia importante de la semántica denotacional propuesta es que no se verifica la propiedad de correctitud en un sentido estricto. Recordemos que una semántica operacional, dada por una relación "$\to$" de reducción *small step* se dice *correcta* con respecto a una semántica denotacional dada por una función de interpretación $[\![-]\!]$ si para cada paso de reducción $P \to Q$ se tiene que $[\![P]\!] = [\![Q]\!]$. En nuestro caso, el teorema de correctitud que enunciamos en el capítulo anterior (Teorema 4.2.8) no logra establecer que si un programa P reduce a otro programa Q entonces las interpretaciones de P y Q son iguales. De hecho, las interpretaciones son iguales cuando el paso de una reducción $P \to Q$ se deriva por cualquiera de las reglas `alloc`, `beta`, `fresh`, `seq` y `unif`, pero no siempre se da la igualdad cuando el paso se deriva por la regla `fail`. En ese caso, sólo se puede asegurar que la interpretación de P es un conjunto que *incluye* a la interpretación de Q. Al no valer la igualdad en todos los casos, el resultado de correctitud de la semántica operacional con respecto a la semántica denotacional no es totalemente satisfactorio, y es por eso que hablamos de *correctitud débil.*

Por otro lado, se presenta la cuestión de si la semántica operacional es completa con respecto a la denotacional. Recordemos que una semántica operacional se dice *completa* con respecto a una semántica denotacional si cada vez que $[\![P]\!] = [\![Q]\!]$ se tiene que P y Q son interconvertibles, es decir, $P \leftrightarrow^* Q$, donde "$\leftrightarrow^*$" denota la clausura reflexiva, simétrica y transitiva de la relación de reescritura $\to$. Como mencionamos anteriormente, la propiedad de completitud no se verifica. Hay varias razones por las cuales esto es así. Listamos algunas de ellas, aunque esta lista no es exhaustiva:

1. **Irrelevancia de las locaciones.** La semántica denotacional ignora las locaciones sobre las abstracciones. Por lo tanto, la interpretación de dos abstracciones alojadas $\lambda^{\ell_1}x.\,P$ y $\lambda^{\ell_2}x.\,P$ con el mismo cuerpo pero diferente locación tiene la misma interpretación bajo cualquier asignación de variables, es decir, $[\![\lambda^{\ell_1}x.\,P]\!]_\rho = [\![\lambda^{\ell_2}x.\,P]\!]_\rho$. Por lo tanto, por el principio de composicionalidad de la semántica denotacional, se verifica que:
$$[\![(\lambda^{\ell_1}x.\,P \overset{\bullet}{=} \lambda^{\ell_1}x.\,P)]\!] = [\![(\lambda^{\ell_1}x.\,P \overset{\bullet}{=} \lambda^{\ell_2}x.\,P)]\!]$$
Pero estos no son interconvertibles, es decir, no vale que $(\lambda^{\ell_1}x.\,P \overset{\bullet}{=} \lambda^{\ell_1}x.\,P) \leftrightarrow^* (\lambda^{\ell_1}x.\,P \overset{\bullet}{=} \lambda^{\ell_2}x.\,P)$, pues $(\lambda^{\ell_1}x.\,P \overset{\bullet}{=} \lambda^{\ell_1}x.\,P) \xrightarrow{\mathtt{unif}} \mathbf{ok}$ mientras que $(\lambda^{\ell_1}x.\,P \overset{\bullet}{=} \lambda^{\ell_2}x.\,P) \xrightarrow{\mathtt{fail}} \mathtt{fail}$.

2. **Incompletitud de la unificación.** Como ya mencionamos, la semántica operacional del cálculo-$\lambda^{\mathtt{U}}$ no resuelve problemas de unificación de orden superior. En particular, las reglas `unif` y `fail` sólo se pueden aplicar cuando la ecuación a unificar es de la forma $\mathtt{v} \stackrel{\bullet}{=} \mathtt{w}$, donde $\mathtt{v}$ y $\mathtt{w}$ deben ser valores. Por ejemplo, el problema de unificación de orden superior $x\,y \stackrel{\bullet}{=} \mathbf{c}$ tiene un unificador más general. Por lo tanto, se tiene que:
$$[\![x\,y \stackrel{\bullet}{=} \mathbf{c}]\!] = \{\mathsf{R}_{\mathbf{ok}}\} = [\![\mathbf{ok}]\!]$$
Sin embargo, el término $x\,y \stackrel{\bullet}{=} \mathbf{c}$ es un término "trabado", es decir, no es un valor pero está en forma normal, y por lo tanto no vale $(x\,y \stackrel{\bullet}{=} \mathbf{c}) \leftrightarrow^* \mathbf{ok}$.

3. **Redexes bloqueados.** Otro problema, relacionado con el anterior, es que algunas reglas de la semántica operacional requieren que un subtérmino llegue a ser un valor para poder proceder con el cómputo, en tanto que la semántica denotacional no impone este requerimiento.

   Por ejemplo, $[\![((x\,y); \lambda^\ell z.\, z)\,\mathbf{c}]\!] = \{\mathsf{R}_{\mathbf{c}}\}$ pero $((x\,y); \lambda^\ell z.\, z)\,\mathbf{c}$ es un término trabado debido a que $x\,y$ no es un valor. En particular $((x\,y); \lambda^\ell z.\, z)\,\mathbf{c}$ no es interconvertible con $\mathbf{c}$.

   Como otro ejemplo, $[\![(\lambda^\ell z.\, \mathbf{c})(x\,y)]\!] = \{\mathsf{R}_{\mathbf{c}}\}$ pero, igual que antes, $(\lambda^\ell z.\, \mathbf{c})(x\,y)$ es un término trabado que no es interconvertible con $\mathbf{c}$.

4. **Multiplicidad de resultados.** La semántica operacional admite multiplicidad de resultados, mientras que este fenómeno no ocurre en la semántica denotacional. Por ejemplo, sucede que $[\![\mathbf{c} \oplus \mathbf{c}]\!] = \{\mathsf{R}_{\mathbf{c}}\}$, y sin embargo $\mathbf{c} \oplus \mathbf{c}$ no es interconvertible con $\mathbf{c}$. Esta diferencia que se da es porque modelamos el dominio de interpretación usando conjuntos, los cuales no admiten repetidos.

5. **Extensionalidad.** La propiedad de extensionalidad indica que para dos abstracciones t_1 y t_2, si $t_1\,s = t_2\,s$, entonces $t_1 = t_2$. De esta propiedad se extiende la η-equivalencia, que indica que los términos $\lambda x.\, t\,x$ y t son equivalentes. En la semántica denotacional que proponemos se cumple que $[\![\lambda x.\, t\,x]\!] = [\![t]\!]$. Sin embargo, en la semántica operacional los términos $\lambda x.\, t\,x$ y t no son interconvertibles.

En las dos subsecciones que siguen, describimos brevemente las dos maneras alternativas de definir la semántica denotacional que ensayamos, con el objeto de recuperar las propiedades de correctitud y completitud. Desafortunadamente, ninguna de las dos variantes de la semántica resulta verificar las propiedades deseadas.

5.1. Variante 1: semántica denotacional con memoria

En esta primera alternativa, proponemos una semántica denotacional para el cálculo-$\lambda^{\mathtt{U}}$ en la que incluimos una noción de *memoria* con el fin de distinguir entre las interpretaciones de las abstracciones y las abstracciones alojadas. Intuitivamente, una expresión $\lambda x.\, P$ debería interpretarse como un comando que reserva espacio libre en la memoria, donde almacena una *clausura*[1], en tanto que una expresión $\lambda^\ell x.\, P$ debería interpretarse como un puntero a una posición de memoria ℓ en la que ya se encuentra almacenada una clausura.

[1] Recordemos que una clausura es código asociado a la función $\lambda x.\, P$, posiblemente acompañado de un entorno que liga las variables libres a valores.

Pasamos a detallar más precisamente la semántica propuesta:

- **Locaciones.** Para cada tipo de la forma $A \to B$ suponemos dado un conjunto infinito de locaciones $(\ell_1^{A\to B}, \ell_2^{A\to B}, \ldots)$. Notamos $\mathsf{Loc}_{A\to B}$ al conjunto de locaciones de tipo $A \to B$.

- **Dominios de interpretación.** La interpretación de los tipos base es la misma, $[\![\alpha]\!] = \mathsf{S}_\alpha$, donde S_α es un conjunto fijo. Las funciones se interpretan ahora como locaciones de memoria, es decir que $[\![A \to B]\!] = \mathsf{Loc}_{A\to B}$ se define como el conjunto de locaciones de tipo $A \to B$.

- **Memoria.** El conjunto de las memorias se nota Mem. Una memoria $\mu \in \mathsf{Mem}$ es un objeto que le asocia valores a las locaciones de memoria. Más precisamente, para cada par de tipos A, B, se tiene que $\mu_{A,B}$ es una función parcial, donde, para cada locación $\ell^{A\to B}$, se tiene que o bien $\mu_{A,B}(\ell^{A\to B})$ está indefinido o bien vale $\mu_{A,B}(\ell^{A\to B}) : [\![A]\!] \to \mathsf{Mem} \to \mathcal{P}(\mathsf{Mem} \times [\![B]\!])$. Es decir, si $\mu_{A,B}(\ell^{A\to B})$ está definido, resulta ser una función que recibe una entrada del dominio de interpretación de A y un estado inicial de la memoria y, de manera no determinística, devuelve un estado final de la memoria y un resultado del dominio de interpretación de B. Cuando puede deducirse del contexto, omitimos superíndices y subíndices. Notamos $\ell \in \mu$ si $\mu(\ell)$ está definido y $\ell \notin \mu$ si no. Decimos que dos memorias μ_1 y μ_2 son *consistentes* si para cada locación ℓ definida en μ_1 y μ_2, tenemos que $\mu_1(\ell) = \mu_2(\ell)$, y escribimos $\mu_1 \uplus \mu_2$ para denotar su *unión*.

- **Asignaciones de variables.** Las asignaciones de variables se definen de manera análoga a la de la semántica denotacional del capítulo previo, y notamos Asg al conjunto de todas las asignaciones de variables.

- **Interpretación de los términos.** Sea $\vdash t : A$ ($\vdash P : A$ respectivamente), entonces su interpretación tiene tipo

  $$[\![t]\!] : \mathsf{Asg} \times \mathsf{Mem} \to \mathcal{P}(\mathsf{Mem} \times [\![A]\!]) \qquad [\![P]\!] : \mathsf{Asg} \times \mathsf{Mem} \to \mathcal{P}(\mathsf{Mem} \times [\![A]\!])$$

  es decir, la interpretación de un término (o programa) de tipo A es una función que a cada asignación de variables y a cada estado inicial de la memoria, le asocia no determinísticamente un estado final de la memoria y un resultado del dominio de interpretación de A.

  La interpretación de los términos y programas se define recursivamente como sigue.

Por brevedad, notamos $[\![t]\!]\,\rho\,\mu$ en lugar de $[\![t]\!](\rho,\mu)$:

$$
\begin{array}{rcl}
[\![x^A]\!]\,\rho\,\mu & \stackrel{\text{def}}{=} & \{(\mu, \rho(x^A))\} \\
[\![\mathbf{c}]\!]\,\rho\,\mu & \stackrel{\text{def}}{=} & \{(\mu, \mathsf{R}_{\mathbf{c}})\} \\
[\![\lambda x^A.\, P]\!]\,\rho\,\mu & \stackrel{\text{def}}{=} & \{(\mu \uplus \{\ell \mapsto f\}, \ell)\} \\
 & & \text{donde } \ell \notin \mu \text{ y } f(a,\mu') = [\![P]\!]\,\rho[x^A \mapsto a]\,\mu' \\
[\![\lambda^{\ell^{A\to B}} x^A.\, P]\!]\,\rho\,\mu & \stackrel{\text{def}}{=} & \{(\mu, \ell)\} \\
[\![t\,s]\!]\,\rho\,\mu & \stackrel{\text{def}}{=} & \{(\mu_3, b) \mid (\mu_1, \ell) \in [\![t]\!]\,\rho\,\mu, (\mu_2, a) \in [\![s]\!]\,\rho\,\mu, (\mu_3, b) \in \mu_1(\ell)(a, \mu_1 \uplus \mu_2)\} \\
[\![t \stackrel{\bullet}{=} s]\!]\,\rho\,\mu & \stackrel{\text{def}}{=} & \{(\mu_1 \uplus \mu_2, \mathsf{R}_{\mathbf{ok}}) \mid (\mu_1, a) \in [\![t]\!]\,\rho\,\mu, (\mu_2, b) \in [\![s]\!]\,\rho\,\mu, a = b\} \\
[\![t; s]\!]\,\rho\,\mu & \stackrel{\text{def}}{=} & \{(\mu_1 \uplus \mu_2, a) \mid (\mu_1, b) \in [\![t]\!]\,\rho\,\mu, (\mu_2, a) \in [\![s]\!]\,\rho\,\mu\} \\
[\![\nu x^A.\, t]\!]\,\rho\,\mu & \stackrel{\text{def}}{=} & \{(\mu', b) \mid a \in [\![A]\!], (\mu', b) \in [\![t]\!]\,[x^A \mapsto a]\,\mu\} \\
[\![\mathtt{fail}^A]\!]\,\rho\,\mu & \stackrel{\text{def}}{=} & \varnothing \\
[\![t \oplus P]\!]\,\rho\,\mu & \stackrel{\text{def}}{=} & [\![t]\!]\,\rho\,\mu \cup [\![P]\!]\,\rho\,\mu
\end{array}
$$

Una característica digna de mencionar es que en esta variante de la semántica la memoria es *monótona*, es decir, una vez que se almacena un valor en una locación, este valor no cambia. Como observación técnica, en la interpretación de la aplicación, la unificación y la secuencia, la operación de unión de memorias $\mu_1 \uplus \mu_2$ asume que las memorias son consistentes. Intuitivamente, esto significa que al evaluar t y s por separado en una memoria común μ, se debe asegurar que las locaciones frescas que se asignan en la evaluación de t son disjuntas de las locaciones frescas que se asignan en la evaluación de s.

Además, podemos observar que la segunda componente de la interpretación de cada uno de los constructores de términos es similar al de la semántica denotacional del capítulo anterior, a excepción de las abstracciones, donde se distinguen las abstracciones alojadas de las que no lo están, y la aplicación. Como mencionamos más arriba, la interpretación de una abstracción no alojada produce un *efecto secundario*, que es reservar en la memoria una locación ℓ fresca y almacenar una función f en dicha locación, en tanto que una abstracción alojada no tiene efectos secundarios (es decir, no modifica el estado de la memoria) y representa simplemente un puntero a una locación ℓ de la memoria μ original. La interpretación de la aplicación tiene como peculiaridad que, una vez que se encontró la locación ℓ donde se encuentra almacenada la función asociada a t, hace un acceso a memoria para encontrar la función correspondiente ($\mu_1(\ell)$).

Esta variante de la semántica denotacional soluciona el problema ya mencionado de correctitud de la regla `fail`. Podemos observar que si $\ell \neq \ell'$ entonces $(\lambda^{\ell} x.\, P \stackrel{\bullet}{=} \lambda^{\ell'} x.\, P) \xrightarrow{\mathtt{fail}}$ `fail` y en efecto se tiene que $[\![(\lambda^{\ell} x.\, P \stackrel{\bullet}{=} \lambda^{\ell'} x.\, P)]\!]\,\mu\,\rho = \varnothing = [\![\mathtt{fail}]\!]\,\mu\,\rho$.

Sin embargo, esta variante de la semántica denotacional tiene un problema severo que hace que no sea correcta. El problema es que a veces deberían darse dependencias cíclicas entre las memorias, que nuestra propuesta no logra capturar. Por ejemplo, el siguiente término:

$$((x \stackrel{\bullet}{=} \lambda z.\, z); y\,\mathbf{c})\,((y \stackrel{\bullet}{=} \lambda z.\, z); x\,\mathbf{c})$$

reduce a $\mathbf{c}\,\mathbf{c}$:

$$
\begin{array}{rcl}
((x \stackrel{\bullet}{=} \lambda z.\, z); y\,\mathbf{c})\,((y \stackrel{\bullet}{=} \lambda z.\, z); x\,\mathbf{c}) & \xrightarrow{\texttt{alloc}} & ((x \stackrel{\bullet}{=} \lambda^{\ell} z.\, z); y\,\mathbf{c})\,((y \stackrel{\bullet}{=} \lambda z.\, z); x\,\mathbf{c}) \\
& \xrightarrow{\texttt{alloc}} & ((x \stackrel{\bullet}{=} \lambda^{\ell} z.\, z); y\,\mathbf{c})\,((y \stackrel{\bullet}{=} \lambda^{\ell'} z.\, z); x\,\mathbf{c}) \\
& \xrightarrow{\texttt{unif}} & (\mathbf{ok}; y\,\mathbf{c})\,((y \stackrel{\bullet}{=} \lambda^{\ell'} z.\, z); (\lambda^{\ell} z.\, z)\,\mathbf{c}) \\
& \xrightarrow{\texttt{unif}} & (\mathbf{ok}; (\lambda^{\ell'} z.\, z)\,\mathbf{c})\,(\mathbf{ok}; (\lambda^{\ell} z.\, z)\,\mathbf{c}) \\
& \xrightarrow{\texttt{seq}} & ((\lambda^{\ell'} z.\, z)\,\mathbf{c})\,(\mathbf{ok}; (\lambda^{\ell} z.\, z)\,\mathbf{c}) \\
& \xrightarrow{\texttt{seq}} & ((\lambda^{\ell'} z.\, z)\,\mathbf{c})\,((\lambda^{\ell} z.\, z)\,\mathbf{c}) \\
& \xrightarrow{\texttt{beta}} & \mathbf{c}\,((\lambda^{\ell} z.\, z)\,\mathbf{c}) \\
& \xrightarrow{\texttt{beta}} & \mathbf{c}\,\mathbf{c}
\end{array}
$$

Una observación importante es que, en este ejemplo, la evaluación del subtérmino de la izquierda sólo puede progresar una vez que la evaluación del subtérmino de la derecha instancia x en $\lambda^{\ell'} z.\, z$. Simétricamente, la evaluación del subtérmino de la derecha sólo puede progresar una vez que la evaluación del subtérmino de la izquierda instancia y en $\lambda^{\ell} z.\, z$. Intuitivamente, esto quiere decir que la formulación composicional de la semántica propuesta arriba no alcanza para capturar la interdependencia entre la evaluación del término de la izquierda y el término de la derecha.

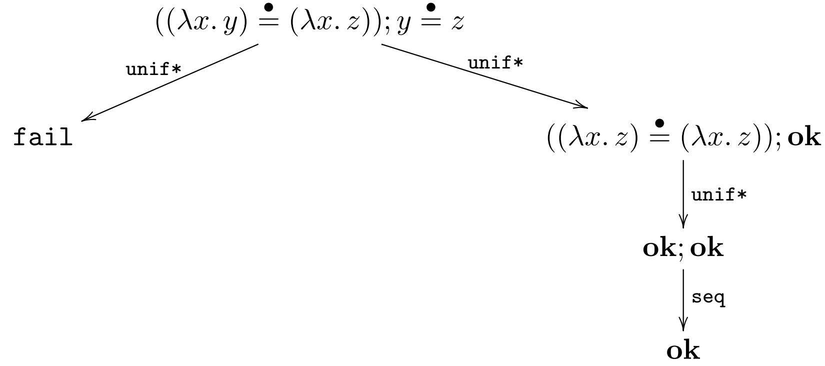
Desde el punto de vista técnico, se puede observar que la interpretación del término $((x \stackrel{\bullet}{=} \lambda z.\, z); y\,\mathbf{c})\,((y \stackrel{\bullet}{=} \lambda z.\, z); x\,\mathbf{c})$ en una memoria inicial μ_0 se define en función de las interpretaciones de $(x \stackrel{\bullet}{=} \lambda z.\, z); y\,\mathbf{c}$ y de $(y \stackrel{\bullet}{=} \lambda z.\, z); x\,\mathbf{c}$, ambos en la misma memoria inicial μ_0, en la cual las variables x e y no tienen asociado un valor aún. No resulta evidente cómo solucionar este desfasaje entre la semántica operacional y la denotacional.

5.2. Variante 2: revisión del cálculo para incorporar clausuras

El desfasaje entre la semántica operacional y la denotacional está dado por la presencia de locaciones sobre las abstracciones. La primera alternativa que mencionamos en la sección anterior consistió en tratar de ajustar la definición de la semántica denotacional para aproximarla a la operacional. La segunda alternativa, que describimos en esta sección, va más atrás y se basa en revisar la sintaxis y la semántica operacional del cálculo.

La razón para incorporar locaciones en el cálculo-$\lambda^{\mathtt{U}}$ es que dos abstracciones sintácticamente diferentes pueden convertirse en abstracciones sintácticamente iguales después de un paso de sustitución. Por ejemplo, consideremos una variante de la regla `unif`, que llamamos `unif*`, que tiene éxito al unificar dos abstracciones sintácticamente iguales (módulo α-equivalencia) y falla al unificar dos abstracciones sintácticamente diferentes. Esto con-

tradiría la propiedad de confluencia[2], como se ve en el siguiente ejemplo:



Es decir, dependiendo del orden de evaluación, el mismo programa puede arrojar dos resultados diferentes.

Los términos se mantienen igual, con la excepción que ahora en lugar de abstracciones y abstracciones alojadas hay una única forma de abstracción llamada *clausura*, que notamos $(\lambda x.\, P)[x_1\backslash \mathtt{v}_1][x_2\backslash \mathtt{v}_2]\ldots[x_n\backslash \mathtt{v}_n]$ donde cada una de las $[x_i\backslash \mathtt{v}_i]$ se llama una *sustitución explícita* y se asume que cada uno de los valores $\mathtt{v}_i$ es cerrado. El punto clave de este cálculo es que dos clausuras $(\lambda x.\, P)[x_1\backslash \mathtt{v}_1][x_2\backslash \mathtt{v}_2]\ldots[x_n\backslash \mathtt{v}_n]$ y $(\lambda y.\, Q)[y_1\backslash \mathtt{w}_1][y_2\backslash \mathtt{w}_2]\ldots[y_m\backslash \mathtt{w}_m]$ unifican si y sólo si son sintácticamente iguales, salvo α-equivalencia y permutación de las sustituciones explícitas.

En la Sección 5.2.1, describimos la sintaxis y semántica operacional de una variante del cálculo-$\lambda^{\mathtt{U}}$ con clausuras, que llamamos cálculo-$\lambda^{\mathtt{UC}}$. En la Sección 5.2.4, describimos algunas dificultades que se encuentran al tratar de proponer una semántica denotacional para este cálculo.

5.2.1. Sintaxis y semántica operacional del cálculo-$\lambda^{\mathtt{UC}}$

Definición 5.2.1 (Sintaxis)**.** Supongamos dados conjuntos infinitos numerables de *variables* $\mathsf{Var} = \{x, y, z, \ldots\}$ y *constructores* $\mathsf{Con} = \{\mathbf{c}, \mathbf{d}, \mathbf{e}, \ldots\}$. Asumimos que existe un constructor distinguido **ok**. Los conjuntos de *términos* $(\{t, s, \ldots\})$, *valores* $(\mathsf{Val} = \{\mathtt{v}, \mathtt{w}, \ldots\})$, *entornos* $(\{\mathtt{E}, \mathtt{E}', \ldots\})$, *programas* $(\{P, Q, \ldots\})$, y *contextos débiles* $(\{\mathsf{W}, \mathsf{W}', \ldots\})$ se definen de manera mutuamente inductiva como sigue.

Términos:

$$
\begin{array}{rcll}
t & ::= & x & \text{variable} \\
 & | & \mathbf{c} & \text{constructor} \\
 & | & (\lambda x.\, P)\mathtt{E} & \text{clausura} \\
 & | & t\, s & \text{aplicación} \\
 & | & t \stackrel{\bullet}{=} s & \text{unificación} \\
 & | & t; s & \text{secuencia} \\
 & | & \nu x.\, t & \text{declaración de variable fresca}
\end{array}
$$

Valores:

$$
\begin{array}{rcl}
\mathtt{v} & ::= & x \\
 & | & (\lambda x.\, P)\mathtt{E} \\
 & | & \mathbf{c}\, \mathtt{v}_1 \ldots \mathtt{v}_n
\end{array}
$$

[2] Esta propiedad afirma esencialmente que el resultado de un cómputo es independiente del orden de evaluación.

Entornos:

$$\begin{array}{rcl} \mathtt{E} & ::= & [\cdot] \\ & | & \mathtt{E}[x\backslash \mathtt{v}] \end{array}$$

Programas:

$$\begin{array}{rcll} P & ::= & \mathtt{fail} & \text{programa vacío} \\ & | & t \oplus P & \text{alternativa no determinística} \end{array}$$

El *dominio* de un entorno se define como $\mathrm{dom}([x_1\backslash \mathtt{v}_1]\dots[x_n\backslash \mathtt{v}_n]) = \{x_1, \dots, x_n\}$. Suponemos que las variables en el dominio de un entorno son **distintas dos a dos**.

Contextos de evaluación:

$$\begin{array}{rcll} \mathsf{W} & ::= & \square & \text{contexto vacío} \\ & | & \mathsf{W}\,t & \text{izquierda de una aplicación} \\ & | & t\,\mathsf{W} & \text{derecha de una aplicación} \\ & | & \mathsf{W} \overset{\bullet}{=} t & \text{izquierda de una unificación} \\ & | & t \overset{\bullet}{=} \mathsf{W} & \text{derecha de una unificación} \\ & | & \mathsf{W}; t & \text{izquierda de una secuencia} \\ & | & t; \mathsf{W} & \text{derecha de una secuencia} \end{array}$$

Definición 5.2.2 (Variables libres)**.** El conjunto de *variables libres* de un término ($\mathsf{fv}(t)$), de un programa ($\mathsf{fv}(P)$), y de un entorno ($\mathsf{fv}(\mathtt{E})$) se definen de manera mutuamente inductiva como sigue:

$$\begin{array}{rclrcl} \mathsf{fv}(x) & \overset{\text{def}}{=} & \{x\} & & & \\ \mathsf{fv}(\mathbf{c}) & \overset{\text{def}}{=} & \varnothing & \mathsf{fv}(\mathtt{fail}) & \overset{\text{def}}{=} & \varnothing \\ \mathsf{fv}((\lambda x.\, P)\mathtt{E}) & \overset{\text{def}}{=} & \mathsf{fv}(\mathtt{E}) & \mathsf{fv}(t \oplus P) & \overset{\text{def}}{=} & \mathsf{fv}(t) \cup \mathsf{fv}(P) \\ \mathsf{fv}(t\,s) & \overset{\text{def}}{=} & \mathsf{fv}(t) \cup \mathsf{fv}(s) & & & \\ \mathsf{fv}(t \overset{\bullet}{=} s) & \overset{\text{def}}{=} & \mathsf{fv}(t) \cup \mathsf{fv}(s) & \mathsf{fv}([\cdot]) & \overset{\text{def}}{=} & \varnothing \\ \mathsf{fv}(t; s) & \overset{\text{def}}{=} & \mathsf{fv}(t) \cup \mathsf{fv}(s) & \mathsf{fv}(\mathtt{E}[x\backslash \mathtt{v}]) & \overset{\text{def}}{=} & \mathsf{fv}(\mathtt{E}) \cup \mathsf{fv}(\mathtt{v}) \\ \mathsf{fv}(\nu x.\, t) & \overset{\text{def}}{=} & \mathsf{fv}(t) \setminus \{x\} & & & \end{array}$$

Asumimos que, para cada clausura, las variables libres de la función están ligadas por el entorno, es decir, en cada subtérmino de la forma $(\lambda x.\, P)\mathtt{E}$, asumimos que $\mathsf{fv}(P) \subseteq \{x\} \cup \mathrm{dom}(\mathtt{E})$. Los términos los consideramos módulo α-equivalencia, es decir, renombrando todas las variables ligadas, y también módulo orden en que se ligan las variables en un entorno. Por ejemplo, $(\lambda x.\, yz)[y\backslash \mathbf{c}][z\backslash \mathbf{d}] = (\lambda x.\, yz)[z\backslash \mathbf{d}][y\backslash \mathbf{c}] = (\lambda x.\, zy)[y\backslash \mathbf{d}][z\backslash \mathbf{c}]$. Sin embargo, notemos que $(\lambda x.\, \mathbf{cc})[\cdot] \neq (\lambda x.\, yy)[y\backslash \mathbf{c}] \neq (\lambda x.\, yz)[y\backslash \mathbf{c}][z\backslash \mathbf{c}]$.

Las operaciones $\mathsf{W}\langle t\rangle$, $\mathsf{W}\langle P\rangle$ y $P \oplus Q$ se definen igual que en el caso del cálculo-$\lambda^{\mathtt{U}}$. Las sustituciones se definen, igual que en el caso del cálculo-$\lambda^{\mathtt{U}}$, como funciones $\sigma : \mathsf{Var} \to \mathsf{Val}$ con soporte finito.

Definición 5.2.3 (Operaciones con sustituciones)**.** Sea $\sigma : \mathsf{Var} \to \mathsf{Val}$ una sustitución cualquiera. La operación que aplica la sustitución σ a un término (resp. programa) evitando

captura de variables libres se nota t^σ (resp. P^σ) y se define de la siguiente forma:

$$\begin{array}{rcll}
x^\sigma & \stackrel{\text{def}}{=} & \sigma(x) & \\
\mathbf{c}^\sigma & \stackrel{\text{def}}{=} & \mathbf{c} & \\
((\lambda x.\, P)\mathtt{E})^\sigma & \stackrel{\text{def}}{=} & (\lambda x.\, P)(\mathtt{E}^\sigma) & \\
(\nu x.\, t)^\sigma & \stackrel{\text{def}}{=} & \nu x.\, t^\sigma & \text{si no hay captura, } \textit{i.e.}\ \forall y \in \mathrm{supp}(\sigma).\ x \notin \{y\} \cup \mathsf{fv}(\sigma(y)) \\
(t\, s)^\sigma & \stackrel{\text{def}}{=} & t^\sigma\, s^\sigma & \\
(t; s)^\sigma & \stackrel{\text{def}}{=} & t^\sigma; s^\sigma & \\
(t \stackrel{\bullet}{=} s)^\sigma & \stackrel{\text{def}}{=} & t^\sigma \stackrel{\bullet}{=} s^\sigma & \\
& & & \\
\mathtt{fail}^\sigma & \stackrel{\text{def}}{=} & \mathtt{fail} & \\
(t \oplus P)^\sigma & \stackrel{\text{def}}{=} & t^\sigma \oplus P^\sigma &
\end{array}$$

donde definimos una sustitución en un entorno como:

$$([x_1\backslash\mathtt{v}_1]\dots[x_n\backslash\mathtt{v}_n])^\sigma \stackrel{\text{def}}{=} [x_1\backslash{\mathtt{v}_1}^\sigma]\dots[x_n\backslash{\mathtt{v}_n}^\sigma]$$

Un entorno $\mathtt{E} = [x_1\backslash\mathtt{v}_1]\dots[x_n\backslash\mathtt{v}_n]$ puede entenderse como una sustitución $\{x_1 \mapsto \mathtt{v}_1, \dots, x_n \mapsto \mathtt{v}_n\}$. En particular, $P^{\mathtt{E}}$ se usa para notar la sustitución de cada ocurrencia libre de x_i por $\mathtt{v}_i$ en P, sin captura de variables libres. Las sustituciones también pueden aplicarse a contextos débiles, tomando $\Box^\sigma \stackrel{\text{def}}{=} \Box$.

Igual que en el caso del cálculo-$\lambda^{\mathtt{U}}$, la *composición* de sustituciones se nota $\rho \cdot \sigma$ y se define como $(\rho \cdot \sigma)(x) \stackrel{\text{def}}{=} \rho(x)^\sigma$, y se definen igual que antes las nociones de sustitución *idempotente* y la relación de ser *más general* ($\sigma \lesssim \rho$).

Algunas propiedades básicas de la sustitución son las siguientes:

Lema 5.2.4 (Propiedades de la sustitución)**.** *Si σ es una sustitución, entonces:*

1. *Si W es un contexto de evaluación débil, entonces W^σ es un contexto de evaluación débil.*

2. *Si $\mathtt{E}$ es un entorno, entonces $\mathtt{E}^\sigma$ es un entorno.*

3. *Si $\mathtt{v}$ es un valor, entonces $\mathtt{v}^\sigma$ es un valor.*

4. $\mathsf{W}\langle t\rangle^\sigma = \mathsf{W}^\sigma\langle t^\sigma\rangle$

5. *Si X es un término o un programa, entonces vale que $X\{x := \mathtt{v}\}^\sigma = X^\sigma\{x := \mathtt{v}^\sigma\}$ tanto como no haya captura de variables libres, es decir, $x \notin \mathrm{supp}(\sigma)$ y que para toda $y \in \mathrm{supp}(\sigma)$ tengamos que $x \notin \mathsf{fv}(\sigma(y))$.*

5.2.2. Algoritmo de unificación

Igual que en el caso del cálculo-$\lambda^{\mathtt{U}}$ se definen las nociones de *problema de unificación*, *unificador* y *unificador más general.*

Teorema 5.2.5 (Cómputo de unificadores más generales)**.** *Se tiene:*

1. *La relación $\rightsquigarrow$ es fuertemente normalizante.*

2. *Las formas normales de* $\rightsquigarrow$ *son* FAIL *y conjuntos de ecuaciones de la forma* $\{x_1 \stackrel{\bullet}{=} \mathtt{v}_1, \dots, x_n \stackrel{\bullet}{=} \mathtt{v}_n\}$ *donde* $x_i \neq x_j$ *y* $x_i \notin \mathsf{fv}(\mathtt{v}_j)$ *para cada* $i, j \in 1..n$.

   Si la forma normal de $\mathcal{E}$ *es* $\{x_1 \stackrel{\bullet}{=} \mathtt{v}_1, \dots, x_n \stackrel{\bullet}{=} \mathtt{v}_n\}$*, decimos que el* $\mathsf{mgu}(\mathcal{E})$ *existe, y* $\mathsf{mgu}(\mathcal{E}) = \{x_1 \mapsto \mathtt{v}_1, \dots, x_n \mapsto \mathtt{v}_n\}$*. Si la forma normal es* FAIL*, entonces decimos que* $\mathsf{mgu}(\mathcal{E})$ *falla.*

3. *La sustitución* $\sigma = \mathsf{mgu}(\mathcal{E})$ *existe si y sólo si existe un unificador para* $\mathcal{E}$*. Cuando existe,* $\mathsf{mgu}(\mathcal{E})$ *es un* unificador más general idempotente.

Demostración. Se omite la demostración de este teorema. Puede demostrarse de manera similar a como se demuestra un teorema análogo para el calculo-$\lambda^{\mathtt{U}}$ [3], que a su vez se basa en las técnicas usuales de unificación de primer orden (ver por ejemplo [2, Sección 4.5]). □

Definición 5.2.6 (Algoritmo de unificación)**.** El siguiente algoritmo es una variante del algoritmo de unificación de Martelli–Montanari. Escribimos $(\lambda x.\, P)\mathtt{E} \approx (\lambda x.\, P)\mathtt{E}'$ si $\mathtt{E} = [x_1 \backslash \mathtt{v}_1] \dots [x_n \backslash \mathtt{v}_n]$ y $\mathtt{E}' = [x_1 \backslash \mathtt{w}_1] \dots [x_n \backslash \mathtt{w}_n]$. En ese caso, abreviamos con $\mathtt{E} \stackrel{\bullet}{=} \mathtt{E}'$ al conjunto de ecuaciones $\{\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{w}_1, \dots, \mathtt{v}_n \stackrel{\bullet}{=} \mathtt{w}_n\}$. Notemos que esto depende de los nombres de las variables ligadas y del orden en que se ligan dentro del entorno. Más concretamente, decimos que dos clausuras $(\lambda x.\, P)\mathtt{E}$ y $(\lambda y.\, Q)\mathtt{E}'$ *tienen la misma forma* si hay un renombre σ tal que $\sigma(x) = y$ y $P^\sigma = Q$. Notemos que si $(\lambda x.\, P)[x_1 \backslash \mathtt{v}_1] \dots [x_n \backslash \mathtt{v}_n]$ y $(\lambda y.\, Q)[y_1 \backslash \mathtt{w}_1] \dots [y_n \backslash \mathtt{w}_n]$ tienen la misma forma, entonces hay una correspondencia 1–1 entre los conjuntos $\{x_1, \dots, x_n\}$ y $\{y_1, \dots, y_m\}$, por lo que en particular $n = m$ y hay una permutación $\pi : \{1, \dots, n\} \to \{1, \dots, n\}$ tal que $x_i = y_{\pi(i)}$ para todo $i = 1..n$. Esto significa que $(\lambda y.\, Q)[y_1 \backslash \mathtt{w}_1] \dots [y_n \backslash \mathtt{w}_n]$ puede escribirse como $(\lambda x.\, P)[x_1 \backslash \mathtt{w}_{\pi(1)}] \dots [x_n \backslash \mathtt{w}_{\pi(n)}] \approx (\lambda x.\, P)[x_1 \backslash \mathtt{v}_1] \dots [x_n \backslash \mathtt{v}_n]$.

Decimos que dos valores $\mathtt{v}, \mathtt{w}$ *colisionan* si vale cualquiera de las siguientes condiciones:

1. Colisión de constructores: $\mathtt{v} = \mathbf{c}\, \mathtt{v}_1 \dots \mathtt{v}_n$ y $\mathtt{w} = \mathbf{d}\, \mathtt{w}_1 \dots \mathtt{w}_m$ con $\mathbf{c} \neq \mathbf{d}$.

2. Colisión de aridad: $\mathtt{v} = \mathbf{c}\, \mathtt{v}_1 \dots \mathtt{v}_n$ y $\mathtt{w} = \mathbf{c}\, \mathtt{w}_1 \dots \mathtt{w}_m$ con $n \neq m$.

3. Colisión de tipos: $\mathtt{v} = \mathbf{c}\, \mathtt{v}_1 \dots \mathtt{v}_n$ y $\mathtt{w} = (\lambda x.\, P)\mathtt{E}$, o viceversa.

4. Colisión de formas: $\mathtt{v} = (\lambda x.\, P)\mathtt{E}$ y $\mathtt{w} = (\lambda y.\, Q)\mathtt{E}'$ donde $(\lambda x.\, P)\mathtt{E}$ y $(\lambda y.\, Q)\mathtt{E}'$ **no** tienen la misma forma.

Definimos un sistema de reescritura cuyos objetos son problemas de unificación $\mathcal{E}$ y el símbolo FAIL. La relación binaria de reescritura $\rightsquigarrow$ está dada por la unión de las siguientes reglas. Notar que "$\uplus$" representa la unión disjunta de conjuntos:

$$
\begin{array}{rlll}
\{x \stackrel{\bullet}{=} x\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-delete}} & \mathcal{E} & \\
\{\mathtt{v} \stackrel{\bullet}{=} x\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-orient}} & \{x \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E} & \text{si } \mathtt{v} \notin \mathsf{Var} \\
\{(\lambda x.\, P)\mathtt{E} \stackrel{\bullet}{=} (\lambda x.\, P)\mathtt{E}'\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-match-clo}} & (\mathtt{E} \stackrel{\bullet}{=} \mathtt{E}') \uplus \mathcal{E} & \text{si } (\lambda x.\, P)\mathtt{E} \approx (\lambda x.\, P)\mathtt{E}' \\
\{\mathbf{c}\, \mathtt{v}_1 \dots \mathtt{v}_n \stackrel{\bullet}{=} \mathbf{c}\, \mathtt{w}_1 \dots \mathtt{w}_n\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-match-cons}} & \{\mathtt{v}_1 \stackrel{\bullet}{=} \mathtt{w}_1, \dots, \mathtt{v}_n \stackrel{\bullet}{=} \mathtt{w}_n\} \uplus \mathcal{E} & \\
\{\mathtt{v} \stackrel{\bullet}{=} \mathtt{w}\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-clash}} & \text{FAIL} & \text{si } \mathtt{v} \text{ y } \mathtt{w} \text{ colisionan} \\
\{x \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-eliminate}} & \{x \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E}^{\{x:=\mathtt{v}\}} & \text{if } x \in \mathsf{fv}(\mathcal{E}) \setminus \mathsf{fv}(\mathtt{v}) \\
\{x \stackrel{\bullet}{=} \mathtt{v}\} \uplus \mathcal{E} & \rightsquigarrow_{\texttt{u-occurs-check}} & \text{FAIL} & \text{si } x \neq \mathtt{v} \text{ y } x \in \mathsf{fv}(\mathtt{v})
\end{array}
$$

Ejemplo 5.2.7. *Sea el conjunto de ecuaciones* $\{(\lambda x.\,\mathbf{c})[\cdot] \doteq x, \mathbf{d}\,x\,((\lambda z.\,z)[z\backslash \mathbf{e}\,w]) \doteq \mathbf{d}\,y\,((\lambda z.\,z)[z\backslash \mathbf{e}\,w'])\}$*, aplicando el algoritmo de unificación obtenemos:*

$$
\begin{array}{ll}
 & \{(\lambda x.\,\mathbf{c})[\cdot] \doteq x, \mathbf{d}\,x\,((\lambda z.\,z)[z\backslash \mathbf{e}\,w]) \doteq \mathbf{d}\,y\,((\lambda z.\,z)[z\backslash \mathbf{e}\,w'])\} \\
\rightsquigarrow_{\texttt{u-orient}} & \{x \doteq (\lambda x.\,\mathbf{c})[\cdot], \mathbf{d}\,x\,((\lambda z.\,z)[z\backslash \mathbf{e}\,w]) \doteq \mathbf{d}\,y\,((\lambda z.\,z)[z\backslash \mathbf{e}\,w'])\} \\
\rightsquigarrow_{\texttt{u-match-cons}} & \{x \doteq (\lambda x.\,\mathbf{c})[\cdot], x \doteq y, ((\lambda z.\,z)[z\backslash \mathbf{e}\,w]) \doteq ((\lambda z.\,z)[z\backslash \mathbf{e}\,w'])\} \\
\rightsquigarrow_{\texttt{u-eliminate}} & \{x \doteq (\lambda x.\,\mathbf{c})[\cdot], (\lambda x.\,\mathbf{c})[\cdot] \doteq y, ((\lambda z.\,z)[z\backslash \mathbf{e}\,w]) \doteq ((\lambda z.\,z)[z\backslash \mathbf{e}\,w'])\} \\
\rightsquigarrow_{\texttt{u-orient}} & \{x \doteq (\lambda x.\,\mathbf{c})[\cdot], y \doteq (\lambda x.\,\mathbf{c})[\cdot], ((\lambda z.\,z)[z\backslash \mathbf{e}\,w]) \doteq ((\lambda z.\,z)[z\backslash \mathbf{e}\,w'])\} \\
\rightsquigarrow_{\texttt{u-match-clo}} & \{x \doteq (\lambda x.\,\mathbf{c})[\cdot], y \doteq (\lambda x.\,\mathbf{c})[\cdot], \mathbf{e}\,w \doteq \mathbf{e}\,w'\} \\
\rightsquigarrow_{\texttt{u-match-cons}} & \{x \doteq (\lambda x.\,\mathbf{c})[\cdot], y \doteq (\lambda x.\,\mathbf{c})[\cdot], w \doteq w'\}
\end{array}
$$

El algoritmo de unificación termina, y retorna $\mathsf{mgu}(\{(\lambda x.\,\mathbf{c})[\cdot] \doteq x, \mathbf{d}\,x\,((\lambda z.\,z)[z\backslash \mathbf{e}\,w]) \doteq \mathbf{d}\,y\,((\lambda z.\,z)[z\backslash \mathbf{e}\,w'])\}) = \{x \mapsto (\lambda x.\,\mathbf{c})[\cdot], y \mapsto (\lambda x.\,\mathbf{c})[\cdot], w \mapsto w'\}$

5.2.3. Semántica operacional del cálculo $\lambda^{\texttt{UC}}$

Definición 5.2.8 (Reglas de reducción)**.** Las reglas de reducción del cálculo-$\lambda^{\texttt{UC}}$ están dadas por:

$$
\begin{array}{rcll}
P_1 \oplus \mathsf{W}\langle(\lambda x.\,Q)\texttt{E}\,\texttt{v}\rangle \oplus P_2 & \xrightarrow{\texttt{beta}} & P_1 \oplus \mathsf{W}\langle Q^{\texttt{E}}\{x := \texttt{v}\}\rangle \oplus P_2 & \\
P_1 \oplus \mathsf{W}\langle\texttt{v}; t\rangle \oplus P_2 & \xrightarrow{\texttt{seq}} & P_1 \oplus \mathsf{W}\langle t\rangle \oplus P_2 & \\
P_1 \oplus \mathsf{W}\langle\nu x.\,t\rangle \oplus P_2 & \xrightarrow{\texttt{fresh}} & P_1 \oplus \mathsf{W}\langle t\{x := y\}\rangle \oplus P_2 & \text{donde } y \notin \mathsf{fv}(\mathsf{W}). \\
P_1 \oplus \mathsf{W}\langle\texttt{v} \doteq \texttt{w}\rangle \oplus P_2 & \xrightarrow{\texttt{unif}} & P_1 \oplus \mathsf{W}\langle\mathbf{ok}\rangle^{\sigma} \oplus P_2 & \text{donde } \sigma = \mathsf{mgu}(\{\texttt{v} \doteq \texttt{w}\}). \\
P_1 \oplus \mathsf{W}\langle\texttt{v} \doteq \texttt{w}\rangle \oplus P_2 & \xrightarrow{\texttt{fail}} & P_1 \oplus P_2 & \text{si } \mathsf{mgu}(\{\texttt{v} \doteq \texttt{w}\}) \text{ falla.}
\end{array}
$$

Las reglas son similares a las del cálculo-$\lambda^{\texttt{U}}$, con las siguientes diferencias:

1. No hay regla `alloc`, ya que no hay abstracciones alojadas.

2. La regla `beta` no aplica una abstracción alojada $\lambda^{\ell}x.\,Q$, sino una *clausura* $(\lambda x.\,Q)\texttt{E}$ a un argumento `v`, aplicando simultáneamente todas las sustituciones del entorno `E` y la sustitución de x por `v` en el cuerpo Q.

3. Las reglas `unif` y `fail` se basan en el algoritmo de unificación modificado como en Def. 5.2.6.

Definición 5.2.9 (Equivalencia estructural)**.** La equivalencia estructural, escrita $P \equiv Q$, es la clausura reflexiva y transitiva de los siguientes dos axiomas:

1. $\equiv$-`swap`: $P \oplus t \oplus s \oplus Q \equiv P \oplus s \oplus t \oplus Q$.

2. $\equiv$-`var`: Si $y \notin \mathsf{fv}(t)$ entonces $P \oplus t \oplus Q \equiv P \oplus t\{x := y\} \oplus Q$.

Resumiendo, la regla $\equiv$-`swap` significa que los programas principales pueden reordenarse arbitrariamente, mientras que la regla $\equiv$-`var` significa que las variables simbólicas son locales a cada programa principal.

Notemos que los axiomas son simétricos, por lo que $\equiv$ es una relación de equivalencia.

Lema 5.2.10 (La equivalencia estructural es una bisimulación fuerte)**.** *La equivalencia estructural* $\equiv$ *es una bisimulación fuerte con respecto a* $\to$*. Más precisamente, si* $P \equiv P'$ *y* $P \xrightarrow{\mathtt{x}} Q$ *para cada regla* $\mathtt{x} \in \{\mathtt{beta}, \mathtt{seq}, \mathtt{fresh}, \mathtt{unif}, \mathtt{fail}\}$*, entonces existe* Q' *tal que* $P' \xrightarrow{\mathtt{x}} Q'$ *y* $Q \equiv Q'$*.*

Demostración. La demostración es similar a la del cálculo-$\lambda^{\mathtt{U}}$ que se encuentra en [3]. □

Teorema 5.2.11 (Confluencia)**.** *La relación de reducción* $\to$ *es confluente módulo* $\equiv$*. Precisamente, si* $P_1 \to^* P_2$ *y* $P_1 \to^* P_3$*, entonces hay un programa* P_4 *tal que* $P_2 \to^* \equiv P_4$ *y* $P_3 \to^* \equiv P_4$*.*

Demostración. La demostración es análoga a la demostración de Confluencia del cálculo-$\lambda^{\mathtt{U}}$, que forma parte del trabajo de [3]. Esta demostración fue estudiada con detalle como parte del trabajo de esta tesis, pero omitimos los detalles. □

5.2.4. Dificultades

El cálculo-$\lambda^{\mathtt{UC}}$ que describimos en la sección anterior tiene un aspecto más sencillo que el cálculo-$\lambda^{\mathtt{U}}$, que es la ausencia de locaciones. Sin embargo, la presencia de clausuras dificulta la formulación de una semántica denotacional.

Un problema proviene de que las clausuras son expresiones de la forma:

$$(\lambda x.\, P)[x_1 \backslash \mathtt{v}_1][x_2 \backslash \mathtt{v}_2] \dots [x_n \backslash \mathtt{v}_n]$$

donde los valores $\mathtt{v}_i$ son "observables". Esto es porque el algoritmo de unificación propuesto (Def. 5.2.6) requiere recuperar el valor de cada uno de los $\mathtt{v}_i$ a partir de la clausura, según lo indica la regla `u-match-clo`. La dificultad es que, si la clasura es de tipo $A \to B$, cada uno de los valores $\mathtt{v}_i$ puede tener un tipo C_i arbitrario, posiblemente más grande que $A \to B$. Concretamente, esto hace que el dominio de interpretación $[\![D]\!]$ de un tipo arbitrario D no se pueda definir inductivamente sobre la estructura de D, porque esto daría lugar a ecuaciones de la siguiente forma, en las que la recursión no está bien fundada:

$$[\![A \to B]\!] = \dots [\![C_1]\!] \dots [\![C_n]\!] \dots$$

Para lidiar con esta dificultad, tratamos de dar una semántica denotacional interpretando los tipos como *órdenes parciales completos*, en lugar de como conjuntos[3], y definiendo $[\![A \to B]\!]$ como un límite filtrante. Esta manera de definir la semántica resulta ser técnicamente compleja y dejamos como trabajo futuro estudiarla con profundidad.

[3] Un orden parcial completo es un orden parcial $(X, \leq)$ que tiene un elemento mínimo y tal que todo conjunto dirigido $\Delta \subseteq X$ tiene una mínima cota superior en X.

6. CONCLUSIONES Y TRABAJO FUTURO

En esta tesis presentamos un sistema de tipos simples para el cálculo-$\lambda^{\mathtt{U}}$, demostramos propiedades fundamentales del sistema (en particular, la propiedad de preservación), proponemos una semántica denotacional para el cálculo-$\lambda^{\mathtt{U}}$, y probamos una forma débil de *correctitud* de la semántica operacional con respecto a la denotacional. El resultado de correctitud no es totalmente satisfactorio, como mencionamos en el capítulo anterior. Además, analizamos dos alternativas para tratar de subsanar esta limitación, que por el momento no resultaron fructíferas. La propiedad de *completitud* de la semántica operacional con respecto a la denotacional parece encontrarse aún más lejos de verificarse.

Como trabajo futuro, queda pendiente poder dar para el cálculo-$\lambda^{\mathtt{U}}$ (o alguna de sus variantes) una semántica denotacional adecuada, para la cual valgan las propiedades de correctitud y completitud. Como posibles aplicaciones, la semántica denotacional se podría usar como herramienta para razonar acerca del comportamiento de los programas. Por ejemplo, podría servir para demostrar que un programa es correcto con respecto a una especificación. Podría servir también para justificar que distintos tipos de transformaciones de los programas son correctas, en el sentido de que preservan el comportamiento, lo que se podría usar como componente en un optimizador. Por ejemplo, si tuviéramos que $[\![(t; s)\, u]\!] = [\![t; (s\, u)]\!]$, esto podría justificar un reemplazo de $((t; s)\, u)$ por $(t; (s\, u))$ como parte de un proceso de optimización.

Además, otras líneas de trabajo relacionadas con la semántica del cálculo-$\lambda^{\mathtt{U}}$, aunque no necesariamente con el aspecto denotacional, pueden ser: establecer una relación entre el cálculo-$\lambda^{\mathtt{U}}$ y cálculos de patrones como el Pure Pattern Calculus de Kesner y Jay [9], reformular las reglas de reducción del cálculo-$\lambda^{\mathtt{U}}$ para que adopten una estrategia de evaluación call-by-need (*lazy*) en lugar de call-by-value y estudiar sistemas de tipos para el cálculo-$\lambda^{\mathtt{U}}$ que puedan expresar *modedness*, que permitan expresar si los datos se encuentran o no instanciados.

Bibliografía

[1] Elvira Albert, Michael Hanus, Frank Huch, Javier Oliver, and Germán Vidal. Operational semantics for functional logic languages. *Electron. Notes Theor. Comput. Sci.*, 76:1–19, 2002.

[2] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1999.

[3] Pablo Barenbaum, Federico Lochbaum, and Mariana Milicich. Semantics of a relational λ-calculus. In Violet Ka I Pun, Volker Stolz, and Adenilso Simão, editors, *Theoretical Aspects of Computing - ICTAC 2020 - 17th International Colloquium, Macau, China, November 30 - December 4, 2020, Proceedings*, volume 12545 of *Lecture Notes in Computer Science*, pages 242–261. Springer, 2020.

[4] Claudia Faggian and Simona Ronchi Della Rocca. Lambda calculus and probabilistic computation. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019.

[5] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theor. Comput. Sci.*, 13:225–230, 1981.

[6] C.A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. Foundations of computing. MIT Press, 1992.

[7] Michael Hanus, Herbert Kuchen, and J J Moreno-Navarro. Curry: A truly functional logic language. 1995.

[8] Gérard P. Huet. The undecidability of unification in third order logic. *Inf. Control.*, 22(3):257–267, 1973.

[9] C. Barry Jay and Delia Kesner. Pure pattern calculus. In Peter Sestoft, editor, *Programming Languages and Systems, 15th European Symposium on Programming, ESOP 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 27-28, 2006, Proceedings*, volume 3924 of *Lecture Notes in Computer Science*, pages 100–114. Springer, 2006.

[10] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In Peter Schroeder-Heister, editor, *Extensions of Logic Programming*, pages 253–281, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[11] Dale Miller. Unification of simply typed lamda-terms as logic programming. In Koichi Furukawa, editor, *Logic Programming, Proceedings of the Eigth International Conference, Paris, France, June 24-28, 1991*, pages 255–269. MIT Press, 1991.

[12] Gopalan Nadathur and Dale Miller. An overview of lambda-prolog. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, pages 810–827. MIT Press, 1988.

[13] Benjamin C. Pierce. *Types and programming languages.* MIT Press, 2002.

[14] Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 154–165, 2002.

[15] Dmitry Rozplokhas, Andrey Vyatkin, and Dmitry Boulytchev. Certified semantics for relational programming. In Bruno C. d. S. Oliveira, editor, *Programming Languages and Systems - 18th Asian Symposium, APLAS 2020, Fukuoka, Japan, November 30 - December 2, 2020, Proceedings*, volume 12470 of *Lecture Notes in Computer Science*, pages 167–185. Springer, 2020.

[16] Manfred Schmidt-Schauß and Michael Huber. A lambda-calculus with letrec, case, constructors and non-determinism. *CoRR*, cs.PL/0011008, 2000.

[17] Antonis Stampoulis and Adam Chlipala. The makam metalanguage. 2014.