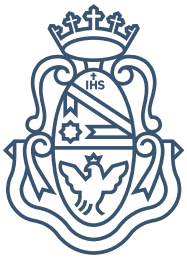
BIORTOGONALIDAD PARA CORRECCIÓN DE COMPILADORES Y ADECUACIÓN COMPUTACIONAL

ALEJANDRO E. GADEA

DIRECTOR: MIGUEL M. PAGANO

En cumplimiento de los requisitos para adquirir el grado
de Doctor en Ciencias de la Computación



Facultad de Matemática, Astronomía, Física y Computación
Universidad Nacional de Córdoba

RESUMEN

Definir un lenguaje de programación implica definir su sintaxis (abstracta y alguna concreta), junto con una interpretación de las expresiones (en el caso de lenguajes declarativos) o sentencias (para lenguajes imperativos). Hoy en día la mayor parte de los lenguajes de programación definen esa interpretación o bien de manera informal, con prosa en lenguaje natural, o bien de forma implícita mediante un compilador o intérprete del lenguaje. La carencia de una definición formal de la semántica dificulta en gran medida el uso de métodos formales para razonar acerca del lenguaje, por ejemplo para probar que una determinada expresión computa una cierta función matemática. Los dos enfoques comunes para definir formalmente la semántica de un lenguaje de programación son la semántica denotacional iniciada por Dana Scott y Christopher Strachey, y la semántica operacional popularizada por Gordon Plotkin. La semántica denotacional permite justificar más fácilmente que una cierta expresión es la implementación de una función matemática mientras que la semántica operacional resulta más apropiada para definir máquinas abstractas.

La programación es realizada por lo general en un lenguaje de alto nivel el cual es conveniente ya que dispone de abstracciones que facilitan el razonamiento, luego un compilador traduce ese programa fuente a un lenguaje de bajo nivel (por ejemplo, de una máquina abstracta) y se asume que el programa objeto resultante se comporta como el programa original. La única manera de tener certeza es probando la corrección del compilador; es decir, que la semántica del programa fuente es preservada por el proceso de compilación.

Los primeros esfuerzos en el área de certificación de compiladores comienzan en 1967 con la corrección de un compilador para expresiones aritméticas de John McCarthy y James Painter, desde entonces el área a estado en constante crecimiento, en particular en el año 2009 se realiza probablemente uno de los trabajos más importantes en el área al certificar un compilador para un fragmento del lenguaje C a un lenguaje ensamblador, desarrollado por el proyecto CompCert. Respecto a la corrección de compiladores de lenguajes funcionales, en 2007 Adam Chlipala presenta la certificación de un compilador del cálculo lambda simplemente tipado hacia código ensamblador, recientemente en 2016 Yong Kiam Tan et al. desarro-

llaron un compilador certificado para el lenguaje CakeML con respecto a diferentes arquitecturas. La corrección de compiladores para lenguajes con evaluación lazy parece un área menos explorada; los trabajos más relacionados involucran la prueba de adecuación entre una semántica denotacional y operacional desarrollado en 1993 por John Launchbury y su reciente mecanización llevada a cabo por Joachim Breitner en Isabelle.

Durante el doctorado hemos estudiado en profundidad los métodos de biortogonalidad y step-indexing para probar tanto adecuación computacional como corrección de compiladores. Un primer aporte es la prueba de corrección de una semántica denotacional con respecto a una operacional para un lenguaje lazy definido por John Launchbury; la prueba original propuesta contiene ciertas irregularidades que corregimos dando nuevas definiciones que nos permiten dar una prueba exhaustiva. Otro aporte es la prueba de adecuación computacional de una semántica operacional con respecto a una denotacional para un lenguaje funcional con un sistema de tipos simple extendido con subtipado y estrategia de evaluación call-by-value. Para este mismo lenguaje probamos la coincidencia entre una semántica denotacional extrínseca y una intrínseca. Este aporte incluye la mecanización completa en Coq de todos los resultados; hasta donde sabemos realizamos la primera mecanización del teorema de bracketing enunciado por John Reynolds. Finalmente damos una prueba de corrección de un compilador para un lenguaje lazy con recursión generando código para una máquina abstracta, este aporte extiende significativamente un trabajo previo desarrollado por Leonardo Rodríguez. Incluimos también la mecanización completa en Coq.

Clasificación (ACM CCS 2012)

- **Theory of computation∼Logic and verification**
- **Theory of computation∼Denotational semantics**
- **Theory of computation∼Operational semantics**
- Theory of computation∼Categorical semantics

Palabras claves

castellano: Adecuación Computacional, Corrección de Compiladores, Coherencia, Bracketing, Biortogonalidad, Relaciones Lógicas, Mecanización.

inglés: Computational Adequacy, Compiler Correctness, Coherence, Bracketing, Biorthogonality, Logical Relations, Mechanization.

AGRADECIMIENTOS

El doctorado fue para mi un increíble viaje que estoy sorprendido de haber podido realizar y sin ninguna duda esto fue posible gracias a todas las personas que me acompañaron de alguna manera: ¡gracias!

Esperando no olvidarme de nadie (disculpas de antemano en tal caso) quisiera intentar agradecer particularmente y sin ningún orden particular, salvo claramente en el caso de mis compañeros Miguel y Ema quienes fueron los grandes acompañantes del viaje y por lo tanto los primeros a los que quiero agradecer.

Principalmente agradecer a Miguel por ser un enorme líder y compañero lleno de paciencia con el que fue más que un placer recorrer semejante camino para mi.

Al compañero de aventuras Ema, fue siempre un placer.

Al hermano mayor de doctorado Leo, herede un trabajo sobre el cual seguir construyendo ha sido genial.

Si bien ya quizá "inactivo", al grupo entero Theona. La oportunidad de trabajar con ellos fue el puntapié inicial de toda esta aventura del doctorado y mucho más importante fue la culpable de que conociera a personas maravillosas.

A los miembros del tribunal: Marcos, Naza y Javi. Gracias por someterse a leer esta tesis y los comentarios con la mejor onda.

A mi amigo de la vida Eze, hay demasiado por agradecer y en resumen es un gracias por tanto.

A los compañeros y las compañeras de la oficina 229-230, fueron fantásticas todas las charlas y risas compartidas dentro y fuera de la oficina.

A mi familia entera por todo el acompañamiento e interés por mis estudios durante el doctorado; charlar sobre las cosas que estudiamos siempre fue muy divertido.

A los compañeros y compañeras de la otra parte de la vida; de la cual forman parte varios de los ya nombrados. Especialmente a Ale, Eze (sí, de nuevo), Juan, Nacho, Marcos, Mati, Manu y Lucas entre tantas cosas para agradecer, gracias por todo el humor.

¡Muchas gracias!

ÍNDICE GENERAL

1

INTRODUCCIÓN

La tarea comúnmente asociada a un programador puede pensarse como *transformar* "ideas" en programas (alguna secuencia de letras y símbolos) escritos en algún lenguaje de programación, que luego son ejecutados por una computadora. Siendo más precisos, un programador escribe un determinado programa (que llamaremos fuente) en algún lenguaje de programación que caracterizamos como de alto nivel, ya que dispone de construcciones que permiten un nivel alto de abstracción. Luego, este programa es *compilado*[1] en un nuevo programa (objeto) escrito en un lenguaje de bajo nivel, que es el ejecutado por una computadora. Evidentemente el programa ejecutado por la computadora nunca es el mismo que escribió el programador, dado que los programas fuente y objeto son distintos en términos de las secuencias de letras y símbolos; sin embargo confiamos que su *semántica*, o comportamiento, "coincida", es decir confiar que el resultado de ejecutar el programa objeto que es fruto de compilar el programa fuente es el esperado. Incluso si el lenguaje fuera *interpretado*, es decir no compilamos el programa fuente a un nuevo programa objeto que es el finalmente ejecutado, sino que ejecutamos el programa original, rara vez el intérprete considera solo el programa y por lo general utiliza información extra obtenida de un análisis estático previo a la ejecución del programa.

Tener fe que el comportamiento de los programas fuente y objeto coincide, no solo parece un poco ingenuo, sino además negligente si consideramos programas críticos que forman parte de computadoras embebidas en vehículos, sistemas biométricos para la seguridad social o sistemas médicos complejos con componentes programados. El comportamiento entre el programa fuente y el objeto no siempre coincide debido a los errores que aparecen en la etapa de compilación como demostraron por ejemplo Yang et al. [51] y Eide et al. [15]. Esta situación puede ser mejorada mediante la *certificación* de compiladores o intérpretes, es decir probando que el programa objeto se comporta como el programa fuente.

1 con la aparición de los asistentes de prueba es curioso remarcar que tal vez uno escriba programas cuya finalidad no es la de ser ejecutados, si no meramente compilados.

Hoy en día cualquier lenguaje de programación popular no cuenta con una certificación de su compilador o intérprete; más aún la definición de su semántica suele estar dada simplemente de manera informal. La carencia de una definición formal de la semántica dificulta en gran medida el uso de métodos formales para razonar acerca del lenguaje y por lo tanto para determinar cuando un compilador es correcto, o la semántica de dos programas coincide.

En este trabajo, utilizamos técnicas conocidas para probar la corrección de compiladores en dos direcciones novedosas: por un lado para probar la corrección de un compilador para un lenguaje funcional lazy, y por otro, para demostrar la coincidencia de dos semánticas para un lenguaje eager.

1.1 DEFINIENDO UN LENGUAJE

Una parte relevante de definir un lenguaje de programación es definir la *gramática concreta* para distinguir entre programas bien y mal escritos. En la gramática concreta nos debemos preocupar por nociones tales como la precedencia de operadores, la distinción entre variables y constantes, cómo identificar bloques en el caso de lenguajes imperativos, cuáles de los operadores son infijos, etc. Sin embargo para la definición y el estudio teórico de un lenguaje no es necesario tanto nivel de detalle, basta con definir una *gramática abstracta*, digamos $\mathcal{L}$, cuyas construcciones sean independientes de cualquier representación posterior. Es importante notar que utilizar una gramática abstracta nos permite concentrarnos plenamente en los aspectos que deseamos para el lenguaje, así por ejemplo si deseamos tener funciones nos concentramos en los aspectos esenciales de la construcción, el nombre de la función, los argumentos y su cuerpo; **fun** $f\ x_1 \ldots x_n\ b$. Luego esta construcción abstracta nos permite representar, a priori, una función en lenguajes bien distintos como pueden ser C y Haskell.

Podemos enriquecer la definición de nuestro lenguaje con un sistema de tipos el cual nos permite detectar errores en etapas tempranas antes de que ocurran en alguna ejecución particular del programa; notar que para un programa lo suficientemente extenso es posible que distintas ejecuciones particulares utilicen diferentes partes del programa, luego algunas ejecuciones podrían fallar y otras no, incluso el error tal vez quede "escondido" en un fragmento de programa que rara vez es ejecutado. Disponer de un sistema de tipos nos permite realizar un análisis estático y exhaustivo del programa antes de ser ejecutado. Los sistemas de tipos populares son con-

siderados un método formal liviano dado que no son demandantes para el programador, sin embargo mientras más complejo es el sistema de tipos esta liviandad va en decremento pero en contraparte más interesantes son las propiedades que se pueden imponer.

Para dar la semántica completa del lenguaje debemos definir el significado de cada construcción de la sintaxis abstracta. A continuación exploramos los dos enfoques más comunes para dar semántica y repasamos uno de los posibles enunciados para demostrar la "coincidencia" entre los diferentes enfoques.

Semántica Operacional

Una posibilidad es dar una *semántica operacional*, o *semántica de transiciones*. Este enfoque fue introducido por Plotkin [37] y la idea es pensar a la ejecución de un programa como la secuencia (posiblemente infinita) de configuraciones que describe la traza de ejecución del programa. El modelo matemático que necesitamos es realmente simple si lo contrastamos con los modelos denotacionales que veremos más adelante.

Una *configuración* $(\mathsf{p}, \mathcal{C})$ es básicamente algún programa $\mathsf{p} \in \mathcal{L}$ a ejecutar junto con algún contexto $\mathcal{C}$ que contiene información sobre la ejecución, luego una *transición* será la transformación de una configuración en otra que determinan como realizar un paso de evaluación; escribimos $(\mathsf{p}, \mathcal{C}) \longmapsto (\mathsf{p}', \mathcal{C}')$, cuando en un paso de ejecución pasamos de la configuración $(\mathsf{p}, \mathcal{C})$ a la configuración $(\mathsf{p}', \mathcal{C}')$. Formalmente, la relación entre configuraciones esta dada por el conjunto de reglas de transición que nos especifica como evaluar un programa utilizando repetidamente las reglas hasta que, eventualmente, alcanzamos un valor. La aplicación sucesiva de la relación $\longmapsto$ da lugar a la clausura transitiva: en general diremos que $(\mathsf{p}, \mathcal{C}) \longmapsto^* (\mathsf{p}', \mathcal{C}')$ si pasamos de la configuración $(\mathsf{p}, \mathcal{C})$ a la configuración $(\mathsf{p}', \mathcal{C}')$ en alguna cantidad finita de pasos. Muchas veces nos centraremos en las ejecuciones que llegan a una configuración terminal, es decir aquella en la cual no podemos aplicar ninguna transición. Sin embargo, también será importante razonar sobre configuraciones que nunca llegan a una configuración terminal, es decir configuración divergentes; usaremos la notación $(\mathsf{p}, \mathcal{C}) \longmapsto^* \infty$ para señalar que la configuración $(\mathsf{p}, \mathcal{C})$ puede realizar una cantidad arbitraria de pasos.

Semántica Denotacional

La semántica denotacional introducida por Scott y Strachey [44] tiene una naturaleza más abstracta: uno elije un modelo matemático, digamos $\mathcal{D}$, y el significado de cada construcción de la sintaxis abstracta es un elemento de ese modelo; por lo general podemos pensar que es necesario definir una función de $\mathcal{L}$ en $\mathcal{D}$. Si nuestro lenguaje tiene variables es necesario contar con un entorno que indique su valor denotacional; podemos pensar que el contexto juega este rol en la semántica operacional. Sea Env el conjunto de todos lo entornos, entonces podemos pensar la semántica denotacional como una función $[\![\_]\!] : \mathcal{L} \to \mathsf{Env} \to \mathcal{D}$.

Es interesante remarcar que si bien no hay restricciones en como definir la función semántica $[\![\_]\!]$, nos son de particular interés aquellas que aseguran que para cada construcción del lenguaje tenemos un único significado y se cumple la propiedad de *composicionalidad*: el significado de cada expresión solo depende del significado de sus sub-expresiones. Esta propiedad es muy importante porque nos permite reemplazar, en una expresión, cualquier sub-expresión por otra siempre y cuando su significado sea el mismo y no cambiar el significado de la expresión completa. Para asegurar estas características exigimos que el conjunto de *ecuaciones semánticas* que definen la función semántica sea *dirigido por sintaxis*, esto significa que el conjunto cumple con las siguientes condiciones:

- Existe una sola ecuación para cada construcción de la gramática abstracta.
- Cada ecuación que define el significado de una expresión, lo hace puramente en función del significado de las sub-expresiones inmediatas.

En presencia de un lenguaje con sistema de tipos tenemos dos formas diferentes de dar semántica a los programas, la versión *intrínseca* en la cual solo los programas bien tipados tienen asociado un significado; en lugar de dar semántica a un programa damos semántica a la derivación (prueba) de que el programa está bien tipado. La otra posibilidad es la *extrínseca* donde la definición de las ecuaciones semánticas es independiente del sistema de tipos pero para el caso de los programas bien tipados podemos asegurar que cumplen ciertas propiedades.

Adecuación computacional

Muchas veces es conveniente disponer de las definiciones de semántica operacional y denotacional, y probar que ambas coinciden. La semántica operacional brinda información precisa sobre como se realizó la ejecución de un programa, por esta razón resulta apropiada para definir máquinas abstractas. En cambio la semántica denotacional mapea programas a elementos en un modelo matemático; es importante notar que un mismo modelo puede ser utilizado para dos estrategias de evaluación distintas (por ejemplo, call-by-name y call-by-need). Sin embargo, la ventaja de este enfoque más abstracto es que nos permite decidir la equivalencia entre programas de manera directa, como así también probar que determinado programa implementa a una función matemática, en el caso de la semántica operacional probar estos conceptos es más complejo.

Diremos que la semántica operacional es *adecuada* con respecto a la semántica denotacional cuando la ejecución operacional de un programa se corresponde con su denotación. El enunciado general de la prueba de adecuación consta de dos aspectos: dado un programa $\mathsf{p} \in \mathcal{L}$ bien tipado, (I) si la semántica denotacional del programa p es denotada por algún elemento $v \in \mathcal{D}$ del modelo matemático, entonces la evaluación de p termina y evalúa a $\overline{v}$ que es la representación abstracta del valor denotacional v; (II) si la denotación del programa p es el elemento del modelo matemático que se corresponde con la no-terminación (es decir, $\bot \in \mathcal{D}$), entonces la evaluación de p puede dar alguna cantidad arbitraria de pasos; en resumen, diremos que $\longmapsto^*$ es adecuada si:

- si $[\![\mathsf{p}]\!] = v$ implica $\mathsf{p} \longmapsto^* \overline{v}$
- si $[\![\mathsf{p}]\!] = \bot$ implica $\mathsf{p} \longmapsto^* \infty$

1.2 COMPILADOR

Un compilador, a priori, no es otra cosa que un programa y por lo tanto está expuesto a errores de implementación como cualquier otro programa. Luego la única manera de evitar errores durante el proceso de compilación es probando su corrección.

El proceso de compilación consiste básicamente en la traducción de un programa escrito en un lenguaje de alto nivel, digamos $\mathcal{H}$, en otro programa escrito en un lenguaje de bajo nivel, digamos $\mathcal{L}$; un compilador será

entonces una función $(\!|\_|\!) : \mathcal{H} \to \mathcal{L}$. En el contexto de este trabajo además ocurrirá que el lenguaje fuente $\mathcal{H}$ tiene descrita su semántica de forma denotacional y el lenguaje objeto $\mathcal{L}$ de manera operacional.

Corrección de un compilador

Es probable que en sistemas críticos uno realice análisis estáticos en el programa escrito en el lenguaje de alto nivel y espera que estas propiedades se mantengan en el programa objeto resultado de la compilación. Esto se puede conseguir probando que el compilador preserva la semántica; es decir, teniendo una prueba de que el programa objeto se comporta como el programa fuente. Definida la semántica denotacional para $\mathcal{H}$ queremos probar que el programa de bajo nivel generado por el compilador implementa el objeto matemático que se corresponde con la denotación del programa de alto nivel compilado. Enunciando esto de forma un poco más precisa tenemos que: dado un programa $p \in \mathcal{L}$ bien tipado, (I) si la semántica denotacional del programa p es denotada por algún elemento $v \in \mathcal{D}$ del modelo matemático, entonces la compilación de p evalúa a $\overline{v}$; (II) si la denotación del programa p es el elemento del modelo matemático que se corresponde con la no-terminación (es decir, $\bot \in \mathcal{D}$), entonces la compilación de p puede dar alguna cantidad arbitraria de pasos; en resumen, diremos que el compilador es correcto si:

- si $[\![p]\!] = v$ implica $(\!|\, p \,|\!) \longmapsto^* \overline{v}$
- si $[\![p]\!] = \bot$ implica $(\!|\, p \,|\!) \longmapsto^* \infty$

1.3 TRABAJOS RELACIONADOS

Los primeros esfuerzos en el área de certificación de compiladores comienzan con los trabajos de McCarthy et al. [29], Morris [30, 31] y Milner et al. [40], desde entonces el área ha estado en constante crecimiento (cf. Linden [28], Necula et al. [20] y Dave [12]). Actualmente el proyecto CompCert [27] es probablemente uno de los trabajos más importantes, en el que se certifica un compilador de un fragmento del lenguaje C a un lenguaje ensamblador.

Existen diferentes trabajos para probar la corrección de compiladores de lenguaje funcionales: Chlipala [8] presenta la certificación de una compilador del cálculo lambda simplemente tipado hacía código ensamblador

utilizando *relaciones lógicas* como estrategia de prueba; Benton et al. [1] prueban la corrección para una variante de la máquina SECD con respecto a un lenguaje call-by-value sumando al esquema de relaciones lógicas el uso de la técnica de *biortogonalidad*. Jaber y Tabareau [22] han aplicado la misma técnica a un lenguaje con tipos polimórficos, al igual que Rodríguez [41, Capítulo 5] para probar la corrección de un compilador para un lenguaje call-by-name. Rodríguez además prueba utilizando *bisimulación* la corrección de un compilador para un calculo lambda extendido con estado y operadores estrictos con estrategia de evaluación call-by-name. Recientemente Tan et al. [48] desarrolló un compilador certificado para CakeML con respecto a diferentes arquitecturas.

La corrección de compiladores para lenguajes con evaluación lazy parece un área menos explorada; los trabajos más relacionados involucran la prueba de adecuación entre una semántica denotacional y operacional desarrollado por Launchbury[25] utilizando *inducción estructural*, luego Sestoft[46] deriva una máquina abstracta y prueba la corrección con respecto a la semántica operacional de Launchbury. Mountjoy[32] modifica la semántica natural propuesta por Launchbury y deriva una máquina abstracta semejante a la máquina STG; posteriormente Encina y Peña-Marí [16] probaron la corrección de una implementación imperativa de la máquina STG con respecto a la semántica de Sestoft. Piróg y Biernacki [33] formalizaron en Coq la derivación de la máquina STG. Recientemente, Breitner[6, 7] mecanizó en Isabelle el desarrollo llevado a cabo por Launchbury. Además de los avances preliminares realizador por Rodríguez en su tesis doctoral[41, Capítulo 7] donde extiende la técnica de biortogonalidad para este tipo de lenguajes sin considerar recursión.

1.4 CONTRIBUCIONES DEL TRABAJO

Durante el doctorado hemos estudiado en profundidad los métodos de biortogonalidad [35] y step-indexing [36] para probar tanto adecuación computacional [18, 19] como corrección de compiladores [10], estos trabajos incluyen además la mecanización completa en Coq de todos los resultados. Para ello, este trabajo está organizado como describimos a continuación. En el capítulo 2 hacemos un resumen de los conceptos matemáticos que utilizamos en el desarrollo.

El primer aporte de este trabajo se realiza en el capitulo 3, el desarrollo de esta tesis comenzó con el estudio del artículo de "A Natural Semantics for Lazy Evaluation" de Launchbury[25] donde se define un lenguaje con

estrategia de evaluación lazy dando semánticas denotacional y operacional, y se prueba su corrección y adecuación. Durante el estudio encontramos ciertas irregularidades que originaron un contra-ejemplo de la prueba de corrección; esto motivó la re-definición de ciertos conceptos y el desarrollo de una prueba corregida y exhaustiva del resultado de corrección.

En el capitulo 4 presentamos dos aportes, el primero es la mecanización en Coq de la prueba de coincidencia entre las semánticas denotacionales intrínseca y extrínseca para un lenguaje con un sistema de tipos enriquecido con subtipado siguiendo la propuesta de Reynolds[38, 39], utilizando una estrategia de prueba mediante relaciones lógicas para un lenguaje call-by-name, nosotros adaptamos la estrategia para un lenguaje call-by-value y realizamos hasta donde sabemos la primera mecanización del teorema de bracketing. El segundo aporte es la mecanización en Coq de la prueba de adecuación computacional de una semántica denotacional extrínseca con respecto a una operacional para el mismo lenguaje con un sistema de tipos enriquecido con subtipado; utilizando como estrategia de prueba biortogonalidad junto con semántica extrínseca, donde el desarrollo de Berger[4] parece ser el único trabajo relacionado. La prueba de coincidencia entre las semánticas denotacionales nos permitió además transferir los resultados de adecuación con respecto a una semántica en base a otra. El desarrollo de este capítulo se realizó en dos etapas: en la primera definimos un lenguaje muy simple para el cual probamos los resultados antes mencionados, luego en la segunda etapa, utilizando la modularidad de las técnicas matemáticas involucradas extendimos el lenguaje enriqueciendo el sistema de tipos con nuevas construcciones y subtipado: ejemplificando que la extensión solo extiende las pruebas sin interferir con las pruebas originales para el lenguaje simple.

La ultima contribución se presenta en el capítulo 5 donde extendemos significativamente los trabajos [10, 41], mecanizando en Coq la corrección de un compilador para un lenguaje lazy con recursión generando código para una máquina abstracta, extendiendo por lo tanto la técnica de biortogonalidad para este tipo de lenguajes. Hasta donde sabemos este es el primer desarrollo que utiliza esta técnica para un lenguaje con estrategia de evaluación lazy.

Finalmente, en el capitulo 6 presentamos las conclusiones y los posibles trabajos futuros. Las mecanizaciones completas de los capítulos 4 y 5 se pueden descargar en `http://www.famaf.unc.edu.ar/~gadea/phd-thesis/`, donde también se encuentra la documentación sobre la mecanización y una versión preliminar de este manuscrito.

2

TEORÍA DE DOMINIOS Y BIORTOGONALIDAD

En este capítulo presentamos un resumen de los modelos y conceptos matemáticos que utilizaremos a lo largo de la tesis, la finalidad es presentar las definiciones y enunciados que aparecen a lo largo del trabajo, y de esta manera hacer la lectura de la tesis auto-contenida; por lo tanto este capítulo no presenta ninguna novedad.

2.1 TEORÍA DE DOMINIOS

En 1970 Dana Scott [42] presenta la solución a la *ecuación de dominios*

$$\mathcal{D} \cong [\mathcal{D} \to \mathcal{D}]$$

y de esta manera da el primer modelo matemático para el cálculo lambda sin tipos. El principal problema para encontrar un modelo no trivial (que ocurrió aproximadamente treinta años después de que Church [9] introdujera por primera vez el lenguaje) radicaba en encontrar una estructura matemática $\mathcal{D}$ tal que la interpretación de las expresiones del cálculo lambda sean elementos de $\mathcal{D}$ y además la aplicación del lenguaje se correspondiera con la noción de aplicación matemática: consideremos la expresión xx su interpretación debería ser un elemento de $\mathcal{D}$ y como la aplicación es interpretada como la aplicación matemática, la primera ocurrencia de x debería corresponderse con una función de $\mathcal{D}$ en $\mathcal{D}$, mientras que la segunda ocurrencia debería ser un elemento de $\mathcal{D}$. Notar que esto implica que tenemos que poder ver a una función de $\mathcal{D}$ en $\mathcal{D}$ como un elemento en el mismo $\mathcal{D}$, luego cada función tiene que tener un punto fijo. Bajo estas restricciones es difícil encontrar un conjunto $\mathcal{D}$, ya que cualquier conjunto con al menos dos elementos contiene funciones para las que no existe un punto fijo. Scott demostró que existen soluciones no triviales para la ecuación recursiva de dominio trabajando con reticulados completos y funciones continuas. Smyth y Plotkin Smyth y Plotkin [47] propusieron una generalización de la solución propuesta por Scott aplicable a diferentes categorías; en particular no es necesario considerar reticulados completos sino dominios. En resumen, la ecuación recursiva de dominios se puede generalizar definiendo

un endo-functor $F\,X = X \to X$ en la categoría de dominios y resolviendo $\mathcal{D} \cong F\,\mathcal{D}$. Esta generalización es interesante ya que variando el functor asociado uno puede interpretar distintas características de los lenguajes.

Un dominio será un tipo particular de conjunto parcialmente ordenado (*poset* por sus siglas en ingles) donde interpretaremos a la relación de orden $\sqsubseteq$ entre elementos del poset como una relación de extensión en términos de información; $d \sqsubseteq d'$ cuando d' ofrece igual o mayor información que d.

Una *cadena* será una secuencia *numerable* de elementos p_i de un poset P, ordenados totalmente con respecto a $\sqsubseteq_P$. A este tipo de cadenas las llamaremos ω-cadena.

Un *predominio*, o cpo (por "complete partial order"), será un poset tal que cada ω-cadena tiene supremo; es decir dada $p_0 \sqsubseteq p_1 \sqsubseteq p_2 \sqsubseteq p_3 \cdots$ luego existe una menor cota superior que denotamos como $\bigsqcup_{i=0}^{\infty} p_i$. Notar que el conjunto de los naturales ($\mathbb{N}$) con el orden usual no es un predominio. Si en cambio utilizamos el orden discreto, es decir definimos $p \sqsubseteq_{\mathbb{N}} p'$ como $p = p'$ entonces obtenemos trivialmente un predominio, ya que cualquier ω-cadena tiene una cantidad finita de elementos y por lo tanto supremo. Esta definición nos permite ver a cualquier conjunto como un predominio, asumiendo el orden discreto.

Dados dos predominios P y P' diremos que una función $f : P \to P'$ es *continua* cuando preserva los supremos de las ω-cadenas. Además definimos al predominio $P \to P'$ como el conjunto de funciones continuas con el orden punto a punto; $f \sqsubseteq_{P \to P'} f'$ si y solo si para todo $p \in P$ vale $f\,p \sqsubseteq_{P'} f'\,p$.

Un *dominio*, o cppo (por "pointed cpo"), será un predominio con menor elemento; dado un dominio D denotaremos a este menor elemento como $\bot_D$. La *promoción* (o lifting) de un predominio cualquiera P será el dominio $P_\bot = P \uplus \{\bot\}$, donde $\bot$ será el menor elemento. Si D es un dominio y $f : D \to D$ una función continua entonces f tiene un menor punto fijo y esta dado por

$$Y_D : (D \to D) \to D$$
$$Y_D\,f = \bigsqcup_{i=0}^{\infty} (f^i \bot_D)$$

este resultado es importante porque nos permitirá definir la semántica de cualquier operador recursivo.

EJEMPLO El siguiente ejemplo pretende ayudarnos a ganar intuición sobre el orden y las ω-cadenas que nos interesan. Supongamos la familia de funciones continuas f_i entre el predominio $\mathbb{N}$ y el dominio $\mathbb{N}_\perp$ (notar que estamos tomando el orden discreto)

$$f_i : \mathbb{N} \to \mathbb{N}_\perp$$

$$f_i\, n = \begin{cases} 1 & n = 0 \\ n * f_i(n-1) & n \in \{1, \ldots, i\} \\ \perp_{\mathbb{N}} & n \notin \{0, 1, \ldots, i\} \end{cases}$$

Notar que una vez fijado el parámetro i para cualquier natural $n \leqslant i$ tenemos que $f_i\, n = \mathit{factorial}(n)$ y para cualquier otro $n > i$, $f_i\, n = \perp_{\mathbb{N}}$. Claramente $f_i \leqslant f_{i+1}$, ya que para todo $n \leqslant i$, $f_i\, n = \mathit{factorial}(n) = f_{i+1}\, n$, además cuando $n = i + 1$ entonces $\perp_{\mathbb{N}} = f_i\, n \sqsubseteq f_{i+1}\, n = \mathit{factorial}(n)$. Esto nos induce la cadena

$$\perp_{\mathbb{N}\to\mathbb{N}_\perp} \sqsubseteq f_0 \sqsubseteq f_1 \sqsubseteq f_2 \sqsubseteq f_3 \sqsubseteq \cdots \sqsubseteq \bigsqcup_{i=0}^{\infty} f_i = \mathit{factorial}(\_)$$

donde el primer elemento $\perp_{\mathbb{N}\to\mathbb{N}_\perp}$ se corresponde con la función que para cualquier n devuelve $\perp_{\mathbb{N}}$, luego a medida que ascendemos vamos "mejorando" la definición. Notar que el supremo es de hecho el factorial puesto que para cualquier n podemos encontrar un i lo suficientemente grande para que $f_i\, n$ es definida, es decir no sea $\perp$.

Mecanización

Para la mecanización en Coq de los resultados expuestos en los siguientes capítulos utilizamos la formalización constructiva de teoría de dominios desarrollada por Benton, Kennedy y Varming [2]. El hecho que la librería se base en principios constructivos introduce sutilezas que se soslayan en usos habituales de dominios; en particular la construcción de la promoción es sutil y conlleva la imposibilidad de decidir la igualdad de un elemento $x \in P_\perp$ respecto de $\perp$.

Los predominios (cpos) son cerrados bajo productos finitos (el orden se determina componente a componente), suma (solo los elementos del mismo conjunto son comparables) y espacio de funciones (comparando

punto a punto). Las mecanizaciones de estas construcciones en la librería son comparables con las definiciones usuales, algo que no ocurre en el caso de la *promoción* de un predominio; en la construcción clásica promover un predominio P es simplemente el dominio $P_\perp = P \uplus \{\perp\}$, donde $\perp$ será el menor elemento. En la librería, la promoción de un predominio P esta construida utilizando listas infinitas (Streams): un elemento de $P_\perp$ será un valor $\mathit{Val}\, p$ con $p \in P$ o $\mathit{Eps}\, s$ para algún $s \in P_\perp$. Es decir, $P_\perp$ se define co-inductivamente.

Para definir el orden primero necesitamos introducir la siguiente función que remueve alguna cantidad de *Eps* de una lista infinita:

$$
\begin{aligned}
&\_ \upharpoonleft \_ : P_\perp \to \mathbb{N} \to P_\perp \\
&\qquad s \upharpoonleft 0 = s \\
&\mathit{Val}\, p \upharpoonleft n+1 = \mathit{Val}\, p \\
&\mathit{Eps}\, s \upharpoonleft n+1 = s \upharpoonleft n
\end{aligned}
$$

Intuitivamente, $\mathit{Val}\, p \leqslant \mathit{Eps}\, s$ si $s \upharpoonleft n = \mathit{Val}\, p'$ y $p \leqslant p'$, para algún n y p'; por otro lado si $s \leqslant s'$, entonces $\mathit{Eps}\, s \leqslant \mathit{Eps}\, s'$. El menor elemento de $P_\perp$ será el stream tal que tiene una cantidad infinita de *Eps* cuya definición co-inductiva es $\perp = \mathit{Eps}\, \perp$.

Es importante notar que una de las consecuencias de esta definición co-inductiva es que no podemos decidir si un elemento de $P_\perp$ es $\perp$ o $\mathit{Val}\, p$. Por lo tanto, no podemos dar la definición clásica de la promoción estricta de $f : P \to P_\perp$

$$
\begin{aligned}
f_{\perp\!\!\perp} &: P_\perp \to P_\perp \\
f_{\perp\!\!\perp}\ \perp &= \perp \\
f_{\perp\!\!\perp}\ \mathit{Val}\, p &= f\, p
\end{aligned}
$$

Afortunadamente la definición nos induce una mónada donde la unidad es *Val* y el bind (utilizando terminología de Haskell) satisface

$$
\begin{aligned}
&\_ \gg\!= \_ : P_\perp \to (P \to P'_\perp) \to P'_\perp \\
&\mathit{Val}\, p \gg\!= f = f\, p \\
&\mathit{Eps}\, s \gg\!= f = \mathit{Eps}(s \gg\!= f)
\end{aligned}
$$

luego podemos probar que para cualquier $f : P \to P_\perp$ vale $\perp \gg\!= f = \perp$, recuperando de esta manera la promoción estricta de f.

Clausura de Scott

Un tema recurrente en la tesis es el uso de predicados sobre dominios (semánticos); a priori esos predicados pueden no contener ni el menor elemento ni ser cerrados por supremos de ω-cadenas. Dado un predicado S sobre un predominio P, la *Clausura de Scott* de S es la menor extensión de S que cumple con ambos requisitos. A continuación damos la definición formal.

Dado $S \subseteq P$ (visto como un predicado) sobre un predominio P podemos definir cuando es

(I) *cerrado por supremo de cadenas:* si para cada cadena $d_i \in P$ tal que para todo $i \in \mathbb{N}$, $d_i \in S$, entonces $\bigsqcup_{i=0}^{\infty} d_i \in S$.

(II) *decreciente:* si para todo d y d', tal que $d' \in S$ y $d \sqsubseteq d'$, entonces $d \in S$.

Cuando el predicado cumpla con estas dos propiedades diremos que es *cerrado*. Además, diremos que el predicado $S \subseteq D$ sobre el dominio D es *estricto* si $\bot_D \in S$. Además, en el caso de ser S cerrado por supremos de cadenas diremos que es *admisible*.

Estas definiciones pueden ser generalizadas sobre una familias de predominios P_i con $i = 1 \ldots n$ tal que una relación $R \subseteq (P_1 \times P_2 \times \ldots \times P_n)$ cumple con las definiciones si lo hace componente a componente; de igual manera sobre una familia de dominios.

Dado $P' \subseteq P$ (visto como un predicado) la *clausura de Scott*, o *clausura ideal*, denotada por $IC(P')$ esta definida como el menor predominio cerrado que contiene a P'.

$$IC(P') = \bigcap \{P'' \subseteq P \mid P'' \subseteq P' \text{ y } P'' \text{ es cerrado } \}$$

Directo de esta definición tenemos que para cualquier predominio P: (I) $IC(P)$ es cerrado (II) $P \subseteq IC(P)$.

Enunciamos de manera general un resultado importante que será de uso frecuente en los siguientes capítulos; consideremos los predicados sobre los predominios $S \subseteq P$ y $S' \subseteq P'$ tal que S' es cerrado, y sea f una función continua de P en P' tal que $f\,S \subseteq S'$ entonces $f\,IC(S) \subseteq S'$. Intuitivamente esto vale ya que la clausuras de Scott a lo sumo agrega los supremos "que faltaban" en S, luego por la continuidad de la función y S' cerrado, tenemos que el supremo resultado de aplicar la función a otro supremo está en S'.

Lema 1. *Sean* $S \subseteq P$ *y* $S' \subseteq P'$ *tal que* S' *es cerrado y* $f : P \to P'$ *una función continua, luego si vale que para todo* $s \in S$ *implica* $f\,s \in S'$ *entonces*

para todo $s \in IC(S)$ *implica* $f\,s \in S'$.

Por el lema anterior si la imagen de una función f sobre un subconjunto S de P esta contenida en S', entonces la imagen de f sobre la clausura de S estará contenida en la clausura de S'. Esto es directo de recordar que para cualquier predominio P vale $P \subseteq IC(P)$.

Lema 2. *Sean* $S \subseteq P$, $S' \subseteq P'$ *y* $f : P \to P'$ *una función continua, luego si vale que para todo* $s \in S$ *implica* $f\,s \in S'$ *entonces*

para todo $s \in IC(S)$ *implica* $f\,s \in IC(S')$.

Estos lemas se pueden generalizar para cualquier función continua con dominio $(P_1 \times P_2 \times \ldots \times P_n)$ sobre familia de predominios P_i con $i = 1 \ldots n$, tomando la clausura componente a componente. Otro lema que será de mucha utilidad es el que nos asegura que al tomar la clausura de Scott para cualquier dominio con orden llano obtenemos el dominio original.

Lema 3. *Sea* D *un dominio con orden llano, luego* $D = IC(D)$

Dado cualquier predominio $P \to P'$ de funciones continuas podemos construirnos uno nuevo resultado de "aplicarle" el predominio P.

Definición 1. *Sean* $F \subseteq P \to P'$ *y* $S \subseteq P$ *predominios, construimos un nuevo predominio* $F\,S \subseteq P'$ *definido como*

$$F\,S = \{d \mid \textit{existen } f \textit{ y } d \textit{ tal que } f \in F,\ s \in S \textit{ y } f\,s = d\}$$

Esta generalización de la aplicación es interesante por el siguiente lema que caracteriza su clausura:

Lema 4. *Sean* $F \subseteq P \to P'$ *y* $S \subseteq P$ *predominios,*

$$IC(F)\,IC(S) \subseteq IC(F\,S)$$

2.2 BIORTOGONALIDAD

En esta sección presentamos las definiciones y explicamos el uso de realizabilidad y biortogonalidad para probar la corrección de un compilador.

La noción básica en el uso de realizabilidad para probar la corrección de un compilador es que una porción de código (generalmente escrito en un lenguaje de bajo nivel) es un *realizador* de un tipo de alto nivel si al ejecutar el código, este termina en un valor del mismo tipo de alto nivel. Por ejemplo, a una porción de código que computa a un numeral se la puede pensar como un realizador de tipo **int**; utilizando tipos refinados [21], se puede obtener realizadores más informativos y decidir además si un código de bajo nivel computa a un numeral particular. En la realizabilidad de Krivine uno además tiene en cuenta el contexto en el que el código de bajo nivel es ejecutado (es decir, no solo la porción de código si no también la pila de ejecución, variables globales, etc); ahora un código será un realizador de cierto tipo si al ejecutarlo junto con cualquier contexto apropiado termina en un valor del mismo tipo de alto nivel: en ese sentido, al contexto se lo considera un *test* y el realizador debe satisfacer todos los tests. En nuestro caso, un realizador satisface un test si la configuración que se obtiene al combinarlos computa el valor de una expresión de alto nivel.

En lugar de empezar con los tests, Benton y Hur [1] identifican lo que ellos llaman *realizadores primitivos* y definen los tests y realizadores a través de los operadores ortogonales asociados con cualquier relación binaria [5, p. 122]. Además, los realizadores primitivos que definen no son realizadores de algún tipo, si no que son realizadores de valores denotacionales. Lo interesante es que las nociones de realizadores primitivos, tests y realizadores se define inductivamente sobre los tipos.

Como mencionamos antes, la combinación de un realizador (código) con un test (contexto de ejecución) da lugar a un configuración completa del entorno de ejecución de bajo nivel. Para probar la corrección del compilador elegimos una *observación*, que será un subconjunto de configuraciones particulares (por ejemplo, la configuración computa a determinado valor). Luego la noción de *satisfactibilidad* está definida mediante una relación entre realizadores y tests; $\_ \models \_ \subseteq \mathsf{R} \times \mathsf{T}$, la cual esta principalmente determinada por una *observación* $\perp\!\!\!\perp$, donde los elementos de este conjunto están caracterizados por alguna combinación de realizadores y tests. Esta relación nos induce los operadores ortogonales $\_^{\perp} : \mathcal{P}(\mathsf{R}) \to \mathcal{P}(\mathsf{T})$ y $\_^{\top} : \mathcal{P}(\mathsf{T}) \to \mathcal{P}(\mathsf{R})$.

Definición 2. *Sea* $\perp\!\!\!\perp \subseteq Conf$ *una observación, que nos determina la relación* $\_ \models \_$.

$$\begin{aligned}
&_^{\perp} : \mathcal{P}(R) \to \mathcal{P}(T)\\
&X^{\perp} = \{ t \mid \textit{para todo } r \in X,\ r \models t \}\\
&_^{\top} : \mathcal{P}(T) \to \mathcal{P}(R)\\
&Y^{\top} = \{ r \mid \textit{para todo } t \in Y,\ r \models t \}\ .
\end{aligned}$$

En general podemos probar que $X \subseteq Y^{\top}$ si y solo si $Y \subseteq X^{\perp}$; es decir, que existe una conexión de Galois antítona entre los posets $\mathcal{P}(R)$ y $\mathcal{P}(T)$, los cuales inducen un operador de clausura $\_^{\perp\top} : \mathcal{P}(R) \to \mathcal{P}(R)$. Dado $X \subseteq R$, decimos que $X^{\perp\top}$ es su biortogonal.

2.3 RELACIONES INDEXADAS

La estrategia de biortogonalidad nos sirve para probar la corrección de un compilador, o la adecuación computacional, cuando los programas terminan. Step-Indexing [36] complementa biortogonalidad para lidiar con los programas divergentes.

Un conjunto indexado $A' \subseteq \mathbb{N} \times A$ es *decreciente* (o *cerrado por abajo*) si lo es cada conjunto A'_i para cada $i \in \mathbb{N}$:

$$(i, a) \in A' \text{ y } j \leqslant i \text{ entonces } (j, a) \in A'$$

Una familia de predicados (indexada por los naturales) $R = \{ R_n \mid n \in \mathbb{N} \}$ sobre un conjunto A es *step-indexed* si satisface

$$A = R_0 \supseteq R_1 \supseteq R_2 \supseteq \dots$$

A partir de ahora nos tomaremos la licencia de hablar de relación indexada para referirnos a una familia de relaciones step-indexed. Dada una relación indexada sobre $A \times B$ nos será conveniente pensarla como una relación sobre conjuntos indexados.

Definición 3. *Dada una relación* $R_n \subseteq (A \times B)$, *podemos construirnos una relación* $\hat{R} \subseteq (\mathbb{N} \times A) \times (\mathbb{N} \times B)$ *definida como*

$$((i, a), (j, b)) \in \hat{R} \textit{ si y solo si } (a, b) \in R_{\min(i,j)}$$

La motivación para la definición anterior es poder aplicar biortogonalidad a partir de una relación indexada; ahora tenemos operadores ortogonales asociados a $N \times A$ y $N \times B$. Para decidir si un elemento (j, b) es un

tests en $A^{\perp}$ equivale a probar (luego de desplegar la definición) que para todo elemento $(i, a) \in A$ vale $(a, b) \in R_{\min(i,j)}$. Como lo demuestra el siguiente lema, si el conjunto A es cerrado por abajo alcanza con probar $(a, b) \in R_{\min(i,j)}$ sólo para los (i, a) tales que $i <= j$. Obviamente como las nociones de tests y realizadores son duales, tenemos un resultado análogo para realizadores.

Lema 5. *Sea* $A' \subseteq \mathbb{N} \times A$, $\hat{R} \subseteq (\mathbb{N} \times A) \times (\mathbb{N} \times B)$. *Si* A' *es cerrado por abajo, entonces*

$$A'^{\perp} = \{\, (j, b) \mid \textit{para todo } (i, a) \in A', \textit{ si } i \leqslant j \textit{ entonces } (a, b) \in R_i \,\}$$

Demostración. Demostramos la doble inclusión.

($\subseteq$) supongamos $(j, b) \in A'^{\perp}$, luego queremos probar que si tomamos un $(i, a) \in A'$ tal que $i \leqslant j$ entonces $(a, b) \in R_i$. Por la definición de $\_^{\perp}$ tenemos que $((i, a), (j, b)) \in \hat{R}$, además es directo que $\min(i, j) = i$ luego $(a, b) \in R_i$.

($\supseteq$) supongamos $(i, a) \in A'$ y queremos demostrar que $(a, b) \in R_{\min(i,j)}$, procedemos por casos:

- Si $\min(i, j) = i$, entonces es porque $i \leqslant j$ luego la prueba es directa.
- Si $\min(i, j) = j$, entonces tenemos que probar $(a, b) \in R_j$. Como $(i, a) \in A'$ y además A' es cerrado por abajo entonces $(j, a) \in A'$, luego podemos completar la prueba haciendo uso de nuestra hipótesis principal.

■

Lema 6. *Sea* $B' \subseteq \mathbb{N} \times B$, $\hat{R} \subseteq (\mathbb{N} \times A) \times (\mathbb{N} \times B)$. *Si* B' *es cerrado por abajo, entonces*

$$B'^{\top} = \{\, (i, a) \mid \textit{para todo } (j, b) \in B', \textit{ si } i \leqslant j \textit{ entonces } (a, b) \in R_i \,\}$$

3

CORRECCIÓN OPERACIONAL PARA UN LENGUAJE LAZY

En este capítulo probaremos la corrección de una semántica operacional con respecto a una semántica denotacional. El siguiente desarrollo surgió durante la etapa inicial de estudio del doctorado, en la cual se estudió el artículo "A Natural Semantics for Lazy Evaluation" de John Launchbury [25] con la finalidad principal de adentrarse en la definición de un lenguaje lazy, las diferentes formas de darle semántica y las pruebas de corrección y adecuación computacional que corroboren que estas semánticas "coinciden". En particular, el artículo mencionado define semánticas operacional y denotacional, y presenta una prueba de corrección de la semántica operacional con respecto de la semántica denotacional. En un punto del estudio descubrimos que la semántica de una de las categorías sintácticas del lenguaje tenía un ligero error. En este capítulo presentamos una prueba "corregida" de la corrección entre las semánticas, para esto realizamos dos cambios principales en las definiciones de Launchbury; re-definimos la semántica denotacional de heaps y utilizamos la definición de semántica operacional presentada por Sestoft [46]. Joachim Breitner [7] realiza un desarrollo semejante en el segundo capítulo de su tesis doctoral.

3.1 LENGUAJE

El lenguaje presentado por Launchbury es un calculo lambda extendido con la construcción *let* recursiva al cual le impone dos restricciones: todas las variables ligadas de una expresión son distintas y la aplicación es restringida en el sentido de que el operando es siempre una variable.

Para modelar el lazyness en la semántica operacional se utiliza un heap que mantiene una referencia a cada expresión introducida por el *let*; cuando se requiere computar la expresión se desreferencia el puntero y luego, si la evaluación converge a un valor, se actualiza el puntero con el valor obtenido de tal computación. Notar que sin realizar la actualización del puntero capturamos la estrategia de evaluación call-by-name. Un heap entonces serán mapas finitos y estará representado por el heap vacío $\square$ o el

heap $\Gamma[\mathsf{p} \mapsto \mathsf{e}]$ cuyo puntero p referencia a la expresión e y cualquier otro puntero q (distinto de p) referencia a $\Gamma\,\mathsf{q}$. Además, dado un heap Γ, denotamos el conjunto de punteros referenciadores de Γ como $\mathtt{dom}(\Gamma)$ y el conjunto de expresiones referenciadas por punteros de Γ como $\mathtt{img}(\Gamma)$.

La decisión de tener una aplicación restringida se debe principalmente a esta estrategia de evaluación operacional en la cual se utiliza el heap para suspender computaciones. Si consideramos la computación de una aplicación no restringida, computar $\mathsf{e}\,\mathsf{e}'$ debería ser equivalente a computar e donde en el heap almacenamos un puntero *fresco* referenciando a e', para simplificar esto podemos utilizar la expresión let que introduce en el heap punteros referenciando a expresiones, delegando de esta manera la creación de punteros a una sola regla operacional.

Por lo tanto la decisión de dar semántica a una aplicación restringida no nos recorta expresividad ya que usamos la expresión let para traducir una aplicación no restringida en una que si lo está. Decimos que $\mathbb{V}$ es un conjunto numerable de variables $(\mathsf{x}, \mathsf{y}, \mathsf{z}, \ldots)$ y punteros $(\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots)$.

Definición 4 (Expresiones)**.**

$$\mathsf{e}, \mathsf{e}' \in \mathsf{Exp} \ ::= \ \mathsf{x} \mid \lambda\mathsf{x}.\,\mathsf{e} \mid \mathsf{e}\,\mathsf{x} \mid \textbf{\textit{let}}\ \mathsf{x}_\mathsf{i}{=}\mathsf{e}_\mathsf{i}\ \textbf{\textit{in}}\ \mathsf{e}$$

Definición 5 (Heaps)**.**

$$\Gamma, \Delta, \Theta \in \mathit{Heap} \ ::= \ [] \mid \Gamma[\mathsf{p} \mapsto \mathsf{e}]$$

3.2 SEMÁNTICA OPERACIONAL

La semántica operacional esta definida en términos de configuraciones, que son pares $\Gamma : \mathsf{e}$, compuesta por un heap y una expresión, Launchbury define una semántica operacional como una relación entre configuraciones, denotada como $\Gamma : \mathsf{e} \Downarrow \Delta : \mathsf{e}'$; la configuración $\Gamma : \mathsf{e}$ computa a $\Delta : \mathsf{e}'$.

Definición 6 (Semántica operacional de Launchbury)**.**

$$\frac{}{\Gamma : \lambda\mathsf{x}.\,\mathsf{e} \Downarrow \Gamma : \lambda\mathsf{x}.\,\mathsf{e}}\ \textsc{Lam} \qquad \frac{\Gamma : \mathsf{e} \Downarrow \Delta : \mathsf{z}}{\Gamma[\mathsf{x} \mapsto \mathsf{e}] : \mathsf{x} \Downarrow \Delta[\mathsf{x} \mapsto \mathsf{z}] : \hat{\mathsf{z}}}\ \textsc{Var}$$

$$\frac{\Gamma : \mathsf{e} \Downarrow \Theta : \lambda\mathsf{y}.\,\mathsf{e}' \qquad \Theta : (\mathsf{e}'/[\mathsf{x}/\mathsf{y}]) \Downarrow \Delta : \mathsf{z}}{\Gamma : \mathsf{e}\,\mathsf{x} \Downarrow \Delta : \mathsf{z}}\ \textsc{App}$$

$$\frac{\Gamma[\mathsf{x}_\mathsf{i} \mapsto \mathsf{e}_\mathsf{i}] : \mathsf{e} \Downarrow \Delta : \mathsf{z}}{\Gamma : \textbf{\textit{let}}\ \mathsf{x}_\mathsf{i}{=}\mathsf{e}_\mathsf{i}\ \textbf{\textit{in}}\ \mathsf{e} \Downarrow \Delta : \mathsf{z}}\ \textsc{Let}$$

Particularmente para la prueba de corrección, pero también en general, queremos que toda configuración involucrada en una evaluación cumpla con la propiedad que Launchbury define como distintivamente nombrada; una configuración $\Gamma : e$ es *distintivamente nombrada* cuando cada ligador de variable tanto en Γ como en e liga nombres de variable distintos. Luego enuncia su Teorema 1: Si $\Gamma : e$ es distintivamente nombrada y tenemos que $\Gamma : e \Downarrow \Delta : e'$ entonces toda configuración que ocurre en la derivación es distintivamente nombrada. La prueba de este resultado tiene como caso interesante al de la regla VAR en el cual Launchbury asegura que al renombrar la expresión z (es decir, $\hat{z}$), resultado de la evaluación $\Gamma : e \Downarrow \Delta : z$, basta para asegurar la propiedad sobre la nueva configuración $\Delta[x \mapsto z] : \hat{z}$.

En su articulo Sestoft deriva una máquina abstracta tomando como base la semántica operacional definida por Launchbury. Sestoft nota que la estrategia de renombre utilizada en la regla VAR no asegura que la configuración $\Delta[x \mapsto z] : \hat{z}$ sea distintivamente nombrada; tomar una variable fresca con respecto a toda la derivación es imposible desde la localidad de la regla. Este análisis proporciona un contra-ejemplo para el teorema 1 de Launchbury. El otro aspecto que considera es que renombrar todas las variables ligadas resulta poco económico desde el punto de vista de una implementación.

Por lo tanto, Sestoft redefine la semántica operacional como una relación entre configuraciones indexada por un conjunto de variables, este conjunto contendrá los punteros que están siendo evaluados en un estado particular de la evaluación; por lo tanto, en el contexto de lazyness este conjunto contiene a los punteros a actualizar. Notar que entre la configuración y el conjunto tenemos todas las variables involucradas hasta el momento en la derivación, por lo tanto es seguro tomar una variable (verdaderamente) fresca en cualquier paso de la evaluación. Esta nueva definición además desplaza el renombre de variables de la regla VAR a la regla LET; en la máquina abstracta derivada por Sestoft, este es el único punto en el que se "crean" nuevos punteros.

A partir de este punto la semántica operacional que utilizamos será la redefinición de Sestoft.

Definición 7 (Semántica operacional de Sestoft)**.**

$$\frac{}{\Gamma : \lambda x.\, e \Downarrow_A \Gamma : \lambda x.\, e}\ \textsc{Lam} \qquad \frac{\Gamma : e \Downarrow_{A \cup \{p\}} \Delta : z}{\Gamma[p \mapsto e] : p \Downarrow_A \Delta[p \mapsto z] : z}\ \textsc{Var}$$

$$\frac{\Gamma : e \Downarrow_A \Theta : \lambda y.\, e' \qquad \Theta : (e'/[p/y]) \Downarrow_A \Delta : z}{\Gamma : e\,p \Downarrow_A \Delta : z}\ \textsc{App}$$

$$\frac{\Gamma[p_i \mapsto \tilde{e}_i] : \tilde{e} \Downarrow_A \Delta : z}{\Gamma : \textbf{\textit{let}}\ x_i{=}e_i\ \textbf{\textit{in}}\ e \Downarrow_A \Delta : z}\ \textsc{Let}$$

con p_i *que no ocurren en* A, Γ *y* **let** $x_i{=}e_i$ ***in*** e *y donde* $\tilde{e} = e/[x_i : p_i]$.

La regla operacional Lam define que las expresiones canónicas serán las abstracciones. En el caso de la regla Var, evaluamos la expresión referenciada por el puntero p, quitando la referencia del heap y agregando el puntero al conjunto A, luego con el resultado de tal evaluación actualizamos el puntero en el nuevo heap. Si la configuración es de la forma $\Gamma : e\,p$ entonces la regla App evalúa a una configuración $\Delta : z$ siempre que la expresión e evalúa a una abstracción de la cual luego el cuerpo e' evalúa a z donde substituimos la variable ligadora y de la abstracción por el puntero p. Notar que en esta ultima regla substituimos un puntero p por una variable y, lo cual induce que punteros $(p, q, r, \ldots)$ y variables $(x, y, z, \ldots)$ pertenecen al conjunto numerable $\mathbb{V}$. La diferenciación entonces entre punteros y variables está dada por su utilización, dada una configuración cualquiera, las variables serán aquellas que actúen como variables ligadoras o ligadas en la expresión. Los punteros por otro lado serán las variables libres para la expresión, las cuales fueron introducidas siempre por la regla Let.

Teniendo en cuenta esta diferencia entre variables y punteros, Sestoft introduce cuando una configuración es *A-good* para algún conjunto A de punteros. Esta propiedad captura varias de las nociones mencionadas anteriormente; todas las variables libres (punteros) de $\Gamma : e$ deben estar contenidas en el conjunto de punteros por actualizar A o referenciando a alguna expresión en Γ, además cada variable ligadora o ligada de la configuración no debe encontrarse en A unión el dominio de Γ. Finalmente todo puntero que será actualizado (es decir pertenece a A) no puede estar referenciando a una expresión en Γ. Definimos al conjunto de variables libres y ligadas para configuraciones como

$$FV(\Gamma : e) = FV(e) \cup \{FV(e) \mid e \in \mathrm{img}(\Gamma)\}$$
$$BV(\Gamma : e) = BV(e) \cup \{BV(e) \mid e \in \mathrm{img}(\Gamma)\}$$

respectivamente, utilizando las definiciones de conjunto de variables libres y ligadas de expresiones.

Definición 8 (Conjuntos de variables libres y ligadas)**.**

Conjunto de variables libres

$$\begin{aligned} FV(x) &= \{x\} \\ FV(\lambda x.\, e) &= FV(e) - \{x\} \\ FV(e\ x) &= FV(e) \cup \{x\} \\ FV(\textbf{\textit{let}}\ x_i{=}e_i\ \textbf{\textit{in}}\ e) &= FV(e) \cup FV(e_i) - \{x_i\} \end{aligned}$$

Conjunto de variables ligadas

$$\begin{aligned} BV(x) &= \emptyset \\ BV(\lambda x.\, e) &= BV(b) \cup \{x\} \\ BV(e\ x) &= BV(e) \\ BV(\textbf{\textit{let}}\ x_i{=}e_i\ \textbf{\textit{in}}\ e) &= BV(e) \cup BV(e_i) \cup \{x_i\} \end{aligned}$$

Definición 9 (Definición de A-good)**.**
Sea A *un conjunto de punteros, diremos que* $\Gamma : e$ *es A-good si:*

- $A \cap \mathrm{dom}(\Gamma) = \emptyset$
- $FV(\Gamma : e) \subseteq A \cup \mathrm{dom}(\Gamma)$
- $BV(\Gamma : e) \cap (A \cup \mathrm{dom}(\Gamma)) = \emptyset$

Sestoft enuncia y da la estrategia de prueba para un lema similar al que presentamos a continuación y para el cual completamos la prueba.

Lema 7. *Si* $\Gamma : e$ *es A-good y tenemos una derivación* $\Gamma : e \Downarrow_A \Delta : z$*, entonces* $\Delta : z$ *es A-good y* $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\Delta)$*.*

Demostración. La prueba procede por inducción estructural en las reglas operacionales.

CASO ABS: Directo.

CASO VAR: Dada la siguiente derivación

$$\frac{\Gamma : e \Downarrow_{A \cup \{p\}} \Delta : z}{\Gamma[p \mapsto e] : p \Downarrow_A \Delta[p \mapsto z] : z}\ \textsc{Var}$$

Suponemos $\Gamma[p \mapsto e] : p$ es A-good y queremos probar que $\Delta[p \mapsto z] : z$ es A-good. La prueba de que $\Gamma : e$ es $(A \cup \{p\})$-good es directa de notar que $A \cup \{p\} \cup dom(\Gamma) = A \cup dom(\Gamma[p \mapsto e])$ y utilizar que tenemos como hipótesis $\Gamma[p \mapsto e] : p$ es A-good. Luego podemos utilizar la hipótesis inductiva y concluir que $\Delta : z$ es $(A \cup \{p\})$-good. Por lo tanto, probar que $\Delta[p \mapsto z]$ es A-good es directo. Finalmente por monotonía de la unión e hipótesis inductiva tenemos que $(dom(\Gamma) \cup \{p\}) \subseteq (dom(\Delta) \cup \{p\})$.

CASO APP: En el caso de la aplicación tenemos la siguiente derivación

$$\frac{\Gamma : e \Downarrow_A \Theta : \lambda y.\, e' \qquad \Theta : (e'/[p/y]) \Downarrow_A \Delta : z}{\Gamma : e, p \Downarrow_A \Delta : z}\ \textsc{App}$$

Notar que si $\Gamma : e, p$ es A-good entonces $\Gamma : e$ es A-good, luego utilizando una de nuestras hipótesis inductivas tenemos que $\Theta : \lambda y.\, e'$ es A-good. Luego queremos probar $\Theta : (e'/[p/y])$ es A-good, probar $A \cap dom(\Theta) = \emptyset$ es directo. Para probar $FV(\Theta : (e'/[p/y])) \subseteq A \cup dom(\Theta)$ notemos que el puntero p es el único que no tienen en común los conjuntos $FV(\Theta : \lambda y.\, e')$ y $FV(\Theta : (e'/[p/y]))$. Por lo tanto restaría probar $p \in (A \cup dom(\Theta))$, pero por hipótesis tenemos que $p \in (A \cup dom(\Gamma))$ y además $dom(\Gamma) \subseteq dom(\Theta)$. Finalmente, probar $BV(\Theta : (e'/[p/y])) \cap (A \cup dom(\Theta)) = \emptyset$ es directo de notar que $BV(\Theta : (e'/[p/y])) \subseteq BV(\Theta : \lambda y.\, e')$.

Por último probar $dom(\Gamma) \subseteq dom(\Delta)$ es directo de las hipótesis inductivas y utilizando la transitividad de $\subseteq$.

CASO LET: En el caso de la regla Let, tenemos la derivación

$$\frac{\Gamma[p_i \mapsto \tilde{e}_i] : \tilde{e} \Downarrow_A \Delta : z}{\Gamma : \textbf{let } x_i{=}e_i \textbf{ in } e \Downarrow_A \Delta : z}\ \textsc{Let}$$

Probemos que $\Gamma[p_i \mapsto \tilde{e}_i] : \tilde{e}$ es A-good. Probar que A y $dom(\Gamma) \cup \{p_i\}$ no tienen punteros en común es directo de notar que por hipótesis tenemos que $A \cap dom(\Gamma) = \emptyset$ y además los p_i son seleccionados de manera tal que no ocurran en A. Para probar que todas las variables libres de la configuración $\Gamma[p_i \mapsto \tilde{e}_i] : \tilde{e}$ están contenidas en $A \cup dom(\Gamma) \cup \{p_i\}$, por hipótesis tenemos que $(FV(\Gamma : e) \cup FV(e_i)) - \{x_i\} \subseteq A \cup dom(\Gamma)$ tenemos además que $FV(\Gamma[p_i \mapsto \tilde{e}_i] : \tilde{e}) = ((FV(\Gamma : e) \cup FV(e_i)) - \{x_i\}) \cup \{p_i\}$ y por lo tanto este último conjunto de variables está contenido en $A \cup dom(\Gamma) \cup \{p_i\}$. Por último, probamos que todas las variables ligadoras y ligadas de la configuración $\Gamma[p_i \mapsto \tilde{e}_i] : \tilde{e}$

no se encuentran en el conjunto de punteros $A \cup \mathrm{dom}(\Gamma)$, pero esto es directo de notar que el conjunto de variables $\mathrm{BV}(\Gamma, p_i \mapsto \tilde{e}_i : \tilde{e})$ tendrá las mismas variables que $\mathrm{BV}(\Gamma : \textbf{let } x_i{=}e_i \textbf{ in } e)$ salvo probablemente por las $\{x_i\}$, luego por hipótesis es directo. Resta probar $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\Delta)$, notar que por hipótesis inductiva tenemos $(\mathrm{dom}(\Gamma) \cup \{p_i\}) \subseteq \mathrm{dom}(\Delta)$, luego claramente podemos concluir lo que buscamos probar.

■

Utilizando el lema anterior podemos probar que si tenemos una derivación cuya configuración inicial es A-good entonces para toda configuración que aparece en la derivación existe un conjunto A' tal que esta es A'-good. El siguiente lema también se desprende directamente del lema anterior.

Lema 8. *Si* $\Gamma : e$ *es A-good y tenemos una derivación* $\Gamma : e \Downarrow_A \Delta : z$*, entonces*

$$p \in A \Rightarrow p \notin \mathrm{dom}(\Delta)$$

Demostración. Tomemos una configuración $\Gamma : e$ tal que es A-good y la derivación $\Gamma : e \Downarrow_A \Delta : z$, luego por lema 7 podemos concluir que $\Delta : z$ es A-good. Por lo tanto, para cualquier variable $p \in A$ tenemos que $p \notin \mathrm{dom}(\Delta)$, ya que $A \cap \mathrm{dom}(\Delta) = \emptyset$. ■

3.3 SEMÁNTICA DENOTACIONAL

Launchbury propone una semántica denotacional utilizando como modelo matemático la solución a la ecuación recursiva de dominio $Value \cong (Value \to Value)_\perp$, de la cual podemos caracterizar dos valores posible; $\perp$ como la computación que diverge, o una función continua de $Value$ en $Value$. Además, denota a las proyecciones e inyecciones relacionadas con la ecuación de dominio como $(\_) \downarrow_{Fn}$: $Value \to (Value \to Value)$ y $Fn(\_) : (Value \to Value) \to Value$ respectivamente. Un entorno estará definido como una función entre el conjunto de variables $\mathbb{V}$ y el dominio $Value$, al conjunto de entornos lo denotamos como Env.

Dado un entorno $\rho \in Env$ y un heap Γ podemos definir una función de *actualización* del entorno ρ con respecto al heap Γ; para cualquier asignación $x \mapsto e \in \Gamma$, la semántica de la variable x en el entorno actualizado debe ser igual a la semántica de la expresión e también en este entorno actualizado. Es importante notar que la semántica de las expresiones contenidas en $\mathrm{img}(\Gamma)$ debe ser calculada con respecto al entorno actualizado ya que nos

interesa cubrir la posibilidad de recursión mutua, debido a que la expresión let permite definiciones mutuamente recursivas. Launchbury define esta función semántica como:

$$\begin{aligned} \{\!|\_|\!\} &: \mathit{Heap} \to \mathrm{Env} \to \mathrm{Env} \\ \{\!|[]|\!\}\rho &= \rho \\ \{\!|\Gamma[p \mapsto e]|\!\}\rho &= \mu\rho'.\, \{\!|\Gamma|\!\}\rho' \sqcup (p \mapsto [\![e]\!]\rho') \sqcup \rho \end{aligned}$$

donde la función $[\![\_]\!] : \mathrm{Exp} \to \mathrm{Env} \to \mathit{Value}$ define la semántica de las expresiones. En los casos de la variable, abstracción y aplicación no hay novedad, utilizamos el entorno, la inyección y la proyección respectivamente para dar las respectivas definiciones. La denotación de la expresión let será la denotación de su cuerpo evaluada en el entorno actualizado con respecto al heap formado por las variables y expresiones introducidas por el let.

Definición 10 (Semántica denotacional)**.**

$$\begin{aligned} [\![\_]\!] &: \mathrm{Exp} \to \mathrm{Env} \to \mathit{Value} \\ [\![p]\!]\rho &= \rho\, p \\ [\![\lambda x.\, e]\!]\rho &= \mathrm{Fn}(\lambda\, d.\, [\![e]\!]([\rho \mid x : d])) \\ [\![e\ p]\!]\rho &= ([\![e]\!]\rho) \downarrow_{\mathrm{Fn}} ([\![p]\!]\rho) \\ [\![\textit{let}\ x_i{=}e_i\ \textit{in}\ e]\!]\rho &= [\![e]\!](\{\!|x_i \mapsto e_i|\!\}\rho) \end{aligned}$$

CONTRA-EJEMPLO DE CORRECCIÓN Launchbury define una relación entre los entornos, de manera que $\rho \leqslant \rho'$ cuando para todo x, si $\rho\, x \neq \bot$ entonces $\rho x = \rho' x$. Luego enuncia el teorema de corrección: dada una derivación $\Gamma : e \Downarrow \Delta : z$ entonces para cualquier entorno ρ vale $[\![e]\!](\{\!|\Gamma|\!\}\rho) = [\![z]\!](\{\!|\Delta|\!\}\rho)$ y $\{\!|\Gamma|\!\}\rho \leqslant \{\!|\Delta|\!\}\rho$. El problema que se presenta durante la prueba de este enunciado está dado por la definición semántica de los heaps, en particular la prueba del teorema falla para el caso de la variable y puntualmente el error está en asegurar que vale $\{\!|\Delta[x \mapsto z]|\!\}\rho(x) = [\![z]\!](\{\!|\Delta[x \mapsto z]|\!\}\rho)$. Notar que si tomamos $\Delta = []$ y $z = \lambda x.$ **let** y=y **in** y entonces sin importar que entorno ρ tomemos $[\![z]\!](\{\!|\Delta[x \mapsto z]|\!\}\rho) = \mathrm{Fn}(\lambda\, d.\, \bot)$. Por otro lado, utilizando las definiciones tenemos:

$$\begin{aligned} \{\!|\Delta[x \mapsto z]|\!\}\rho(x) &= (\mu\rho'.\, (x \mapsto [\![z]\!]\rho') \sqcup \rho)(x) \\ &= ((x \mapsto \mathrm{Fn}(\lambda\, d.\, \bot)) \sqcup \rho)(x) \\ &= \mathrm{Fn}(\lambda\, d.\, \bot) \sqcup \rho(x) \end{aligned}$$

notar que el resultado dependerá de qué entorno ρ consideremos, si tomamos un entorno tal que $\rho(x) = Fn(\lambda\, d.\, Fn(\lambda\, d'.\, \bot))$ entonces hemos construido un contra-ejemplo de que la propiedad no es válida.

A continuación presentamos una definición corregida para la semántica de los heaps que captura exactamente la noción original pretendida. Nuestra definición es diferente a la propuesta por Breitner [7], sin embargo se puede notar su equivalencia mediante el lema 4 del capítulo 2 de su tesis.

La definición corregida toma el punto fijo de una función auxiliar cuya "tarea" será redefinir explícitamente el valor de los punteros contenidos en el heap.

Definición 11 (Semántica de heaps corregida)**.**

$$(\!|\_|\!) : \mathit{Heap} \to Env \to Env \to Env$$

$$(\!|x_i \mapsto e_i|\!)_\rho\ \rho' x = \begin{cases} \rho\, x & x \neq x_i \\ [\![e_i]\!]\rho' & x = x_i \end{cases}$$

$$\{\!|\_|\!\} : \mathit{Heap} \to Env \to Env$$

$$\{\!|x_i \mapsto e_i|\!\}\rho = \mu\rho'.\, (\!|x_i \mapsto e_i|\!)_\rho\ \rho'$$

Esta definición nos permite enunciar y probar la propiedad que anteriormente falló y que invalidaba la prueba del teorema de corrección. A partir de este punto muchas de las pruebas asumen la continuidad de varias de las funciones definidas.

Lema 9.

$$(\{\!|\Gamma[p \mapsto e]|\!\}\rho)p = [\![e]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho)$$

Demostración.

$$\begin{aligned}
(\{\!|\Gamma[p \mapsto e]|\!\}\rho)p &= (\mu\rho'.(\!|\Gamma[p \mapsto e]|\!)_\rho\rho')p \\
&= (\bigsqcup_{n=0}^{\infty}(\!|\Gamma[p \mapsto e]|\!)_\rho^n\bot_{Env})p \\
&= \bigsqcup_{n=0}^{\infty}((\!|\Gamma[p \mapsto e]|\!)_\rho^n\bot_{Env}p) \\
&= \bigsqcup_{n=1}^{\infty}\begin{cases} \rho\,p & p \notin (dom(\Gamma) \cup p) \\ [\![e]\!](\!|\Gamma[p \mapsto e]|\!)_\rho^{n-1}\bot_{Env} & p = p \\ [\![\Gamma q]\!](\!|\Gamma[p \mapsto e]|\!)_\rho^{n-1}\bot_{Env} & p \in dom(\Gamma) \end{cases} \\
&= \bigsqcup_{n=1}^{\infty}([\![e]\!](\!|\Gamma[p \mapsto e]|\!)_\rho^{n-1}\bot_{Env}) \\
&= [\![e]\!](\bigsqcup_{n=1}^{\infty}(\!|\Gamma[p \mapsto e]|\!)_\rho^{n-1}\bot_{Env}) \\
&= [\![e]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho)
\end{aligned}$$

■

Un resultado interesante que probamos es el enunciado de *Coincidencia* el cual refleja que la parte interesante de un entorno es su definición para las variables libres de la expresión. Más allá del interés general, en particular utilizamos Coincidencia a lo largo de algunas pruebas. Este resultado no aparece mencionado por Launchbury aunque es importante mencionar que tampoco le es necesario.

Lema 10 (Coincidencia)**.**

Si para todo $x \in FV(e)$ *vale* $\rho\,x = \rho'\,x$*, entonces* $[\![e]\!]\rho = [\![e]\!]\rho'$

Demostración. La prueba procede por inducción estructural sobre e, donde el caso interesante es el de la expresión let.

CASO VAR: Directo por la hipótesis del lema.

CASO ABS: Supongamos que para toda v tal que $v \in FV(\lambda x.\ e)$, $\rho v = \rho' v$ luego

$$\begin{aligned}[\![\lambda x.\ e]\!]\rho &= Fn(\lambda d.\ [\![e]\!][\ \rho \mid x : d\ ]) \\ &= Fn(\lambda d.\ [\![e]\!][\ \rho' \mid x : d\ ]) \\ &= [\![\lambda x.\ e]\!]\rho'\end{aligned}$$

donde el segundo paso de igualdad se justifica utilizando la hipótesis inductiva, para lo cual probamos que para toda $v \in FV(e)$ vale $[\ \rho \mid x : d\ ](v) = [\ \rho' \mid x : d\ ](v)$ utilizando la hipótesis del lema.

CASO APP: Suponemos para toda $v \in FV(e\ p) = FV(e) \cup \{p\}$ vale $\rho v = \rho' v$, de lo cual se desprende inmediatamente que $[\![p]\!]\rho = [\![p]\!]\rho'$. El resto de la prueba es directa utilizando la hipótesis inductiva.

$$\begin{aligned}[\![e\ p]\!]\rho &= ([\![e]\!]\rho) \downarrow_{Fn} ([\![p]\!]\rho) \\ &= ([\![e]\!]\rho') \downarrow_{Fn} ([\![p]\!]\rho') \\ &= [\![e\ p]\!]\rho'\end{aligned}$$

CASO LET Suponemos para toda $v \in (FV(e) \cup FV(e_i)) - \{x_i\}$ vale $\rho v = \rho' v$ y queremos probar

$$[\![\textbf{let } x_i = e_i \textbf{ in } e]\!]\rho = [\![\textbf{let } x_i = e_i \textbf{ in } e]\!]\rho'$$

para lo cual, usando la definición, alcanza con probar

$$[\![e]\!](\{\!|x_i \mapsto e_i|\!\}\rho) = [\![e]\!](\{\!|x_i \mapsto e_i|\!\}\rho')$$

Si probamos que para toda variable v libre en e vale $\{\!|x_i \mapsto e_i|\!\}\rho(v) = \{\!|x_i \mapsto e_i|\!\}\rho'(v)$ entonces podemos utilizar la hipótesis inductiva y completar la prueba. Para probar esto probaremos que las cadenas asociadas a estos dos supremos son iguales. Probaremos que para todo $n \in \mathbb{N}$ e $v \in FV(e_i)$ vale

$$(\!(x_i \mapsto e_i)\!)^n_\rho \bot_{Env})(v) = (\!(x_i \mapsto e_i)\!)^n_{\rho'} \bot_{Env})(v)$$

Procedemos por inducción en n, donde el caso de $n = 0$ es directo. Nos resta probar que $(\!(x_i \mapsto e_i)\!)^{n+1}_\rho \bot_{Env})(v) = (\!(x_i \mapsto e_i)\!)^{n+1}_{\rho'} \bot_{Env})(v)$. Si $v \notin \{x_i\}$ entonces es directo ya que por hipótesis $\rho v = \rho' v$. Si $v = x_i$ para algún i, entonces necesitamos probar que

$$[\![e_i]\!](\!(x_i \mapsto e_i)\!)^n_\rho \bot_{Env}) = [\![e_i]\!](\!(x_i \mapsto e_i)\!)^n_{\rho'} \bot_{Env})$$

luego podemos completar la prueba utilizando la hipótesis inductiva sobre los e_i donde por hipótesis inductiva sobre n tenemos que vale $(\!|x_i \mapsto e_i|\!)^n_\rho \bot_{Env}(v) = (\!|x_i \mapsto e_i|\!)^n_{\rho'} \bot_{Env}(v)$.

Hemos probado que $\{\!|x_i \mapsto e_i|\!\}\rho(v) = \{\!|x_i \mapsto e_i|\!\}\rho'(v)$ para todo $v \in FV(e_i)$, pero el resultado que necesitamos para completar la prueba debe ser para cualquier $v \in FV(e)$. Pero esto es directo de notar que si $v \notin \{x_i\}$ ocurre como antes que vale por hipótesis $\rho v = \rho' v$ y si $v = x_i$ para algún i entonces por el lema 9

$$\{\!|x_i \mapsto e_i|\!\}\rho(v) = [\![e_i]\!](\{\!|x_i \mapsto e_i|\!\}\rho)$$

$$\{\!|x_i \mapsto e_i|\!\}\rho'(v) = [\![e_i]\!](\{\!|x_i \mapsto e_i|\!\}\rho')$$

luego utilizando este ultimo resultado y la hipótesis inductiva sobre e_i hemos completado la prueba.

■

Los siguientes lemas nos muestran la equivalencia entre distintas formas de actualizar un entorno con respecto a un heap. Dado un entorno ρ y un heap $\Gamma[p \mapsto e]$ probaremos que el entorno actualizado $\{\!|\Gamma, p \mapsto e|\!\}\rho$ cumple

- $\{\!|\Gamma, p \mapsto e|\!\}\rho = \{\!|\Gamma|\!\}(\mu\rho'.\,[\,\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')\,])$
- $\{\!|\Gamma, p \mapsto e|\!\}\rho = \{\!|\Gamma|\!\}([\,\rho \mid p : [\![e]\!](\{\!|\Gamma, p \mapsto e|\!\}\rho)\,])$

las dos igualdades nos exponen que podemos actualizar un entorno considerando de a una asignación del heap por vez. La estrategia general en la demostración de estos resultados será utilizar que el supremo es la menor de las cotas superiores; de esa manera lo que probaremos será que determinado supremo es cota superior de alguna cadena y por lo tanto es mayor o igual que el supremos de tal cadena.

Lema 11. *Si para todo* $v \notin dom(\Gamma)$ *vale* $\overline{\rho}(v) \sqsubseteq \{\!|\Gamma[p \mapsto e]|\!\}\rho(v)$, *entonces* $\{\!|\Gamma|\!\}\overline{\rho} \sqsubseteq \{\!|\Gamma[p \mapsto e]|\!\}\rho$

Demostración. Supongamos que para toda variable $v \notin dom(\Gamma)$ vale $\overline{\rho}(v) \sqsubseteq \{\!|\Gamma[p \mapsto e]|\!\}\rho(v)$ y probemos por inducción en $n \in \mathbb{N}$

$$(\!|\Gamma|\!)^n_{\overline{\rho}} \bot_{Env} \sqsubseteq \{\!|\Gamma[p \mapsto e]|\!\}\rho$$

el caso de $n = 0$ es directo. Luego probemos que $(\!|\Gamma|\!)^{n+1}_{\overline{\rho}} \bot_{Env} \sqsubseteq \{\!|\Gamma[p \mapsto e]|\!\}\rho$. Tomemos una variable v cualquiera y probemos

$$((\!|\Gamma|\!)^{n+1}_{\overline{\rho}} \bot_{Env})(v) \sqsubseteq (\{\!|\Gamma[p \mapsto e]|\!\}\rho)(v)$$

Si $v \notin \mathrm{dom}(\Gamma)$, entonces por definición e hipótesi respectivamente vale $(\!|\Gamma|\!)^{n+1}_{\overline{\rho}} \bot_{Env}(v) = \overline{\rho}(v) \sqsubseteq \{\!|\Gamma[p \mapsto e]|\!\}\rho(v)$.

Si $v \in \mathrm{dom}(\Gamma)$, entonces utilizando la definición semántica de heaps tenemos,

$$(\!|\Gamma|\!)^{n+1}_{\overline{\rho}} \bot_{Env})(v) = (\!|\Gamma|\!)_{\overline{\rho}}((\!|\Gamma|\!)^{n}_{\overline{\rho}} \bot_{Env})(v) = [\![\Gamma v]\!]((\!|\Gamma|\!)^{n}_{\overline{\rho}} \bot_{Env})$$

Por lo tanto utilizando la hipótesis inductiva y la continuidad de la función $[\![\Gamma v]\!] : Env \to Value$ obtenemos

$$\begin{aligned}(\!|\Gamma|\!)^{n+1}_{\overline{\rho}} \bot_{Env})(v) &= [\![\Gamma v]\!]((\!|\Gamma|\!)^{n}_{\overline{\rho}} \bot_{Env}) \\ &\sqsubseteq [\![\Gamma v]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho) = (\{\!|\Gamma[p \mapsto e]|\!\}\rho)(v)\end{aligned}$$

donde la última igualdad se justifica utilizando el lema 9.

■

Lema 12. *Para todo* $n \in \mathbb{N}$, *si* $\forall v \notin \mathrm{dom}(\Gamma)$, $(\!|\Gamma[p \mapsto e]|\!)^{n}_{\rho} \bot_{Env}(v) \sqsubseteq \overline{\rho}(v)$ *entonces* $(\!|\Gamma[p \mapsto e]|\!)^{n}_{\rho} \bot \sqsubseteq \{\!|\Gamma|\!\}\overline{\rho}$

Demostración. Procedemos por inducción en $n \in \mathbb{N}$. El caso de $n = 0$ es directo. Para probar el caso inductivo suponemos para toda variable $v \notin \mathrm{dom}(\Gamma)$ vale $((\!|\Gamma[p \mapsto e]|\!)^{n+1}_{\rho} \bot_{Env})(v) \sqsubseteq \overline{\rho}(v)$ y probemos

$$(\!|\Gamma[p \mapsto e]|\!)^{n+1}_{\rho} \bot_{Env} \sqsubseteq \{\!|\Gamma|\!\}\overline{\rho}$$

Notar que $(\!|\Gamma[p \mapsto e]|\!)^{n}_{\rho} \bot_{Env} \sqsubseteq (\!|\Gamma[p \mapsto e]|\!)^{n+1}_{\rho} \bot_{Env}$ y por lo tanto vale que para toda $v \notin \mathrm{dom}(\Gamma)$, $((\!|\Gamma[p \mapsto e]|\!)^{n}_{\rho} \bot_{Env})(v) \sqsubseteq \overline{\rho}(v)$. Además utilizando la hipótesis inductiva podemos concluir $(\!|\Gamma[p \mapsto e]|\!)^{n}_{\rho} \bot_{Env} \sqsubseteq \{\!|\Gamma|\!\}\overline{\rho}$.

Probemos ahora $((\!|\Gamma[p \mapsto e]|\!)^{n+1}_{\rho} \bot_{Env})(v) \sqsubseteq (\{\!|\Gamma|\!\}\overline{\rho})(v)$, tomemos un variable v cualquiera

Si $v \notin \mathrm{dom}(\Gamma)$, nuestra hipótesis que nos asegura $((\!|\Gamma[p \mapsto e]|\!)^{n+1}_{\rho} \bot_{Env})(v) \sqsubseteq \overline{\rho}(v)$ y utilizando la definición de la semántica de heaps podemos obtener que $\overline{\rho}(y) = \{\!|\Gamma|\!\}\overline{\rho}(y)$.

Si $v \in \mathrm{dom}(\Gamma)$, entonces

$$\begin{aligned}(\!|\Gamma[p \mapsto e]|\!)^{n+1}_{\rho} \bot_{Env})(v) &= (\!|\Gamma[p \mapsto e]|\!)_{\overline{\rho}}((\!|\Gamma[p \mapsto e]|\!)^{n}_{\overline{\rho}} \bot_{Env})(v) \\ &= [\![\Gamma[p \mapsto e]\, v]\!]((\!|\Gamma[p \mapsto e]|\!)^{n}_{\overline{\rho}} \bot_{Env}) \\ &= [\![\Gamma v]\!]((\!|\Gamma[p \mapsto e]|\!)^{n}_{\overline{\rho}} \bot_{Env})\end{aligned}$$

Además utilizando el lema 9 tenemos $(\{\!|\Gamma|\!\}\overline{\rho})(v) = [\![\Gamma v]\!](\{\!|\Gamma|\!\}\overline{\rho})$. Por lo tanto utilizando que anteriormente probamos $(\!|\Gamma[p \mapsto e]|\!)^n_\rho \bot_{Env} \sqsubseteq \{\!|\Gamma|\!\}\overline{\rho}$ y la continuidad de la función semántica $[\![\Gamma v]\!]$ podemos concluir

$$\begin{aligned}(\!|\Gamma[p \mapsto e]|\!)^{n+1}_\rho \bot_{Env})(v) &= [\![\Gamma v]\!]((\!|\Gamma[p \mapsto e]|\!)^n_{\overline{\rho}} \bot_{Env}) \\ &\sqsubseteq [\![\Gamma v]\!](\{\!|\Gamma|\!\}\overline{\rho}) = (\{\!|\Gamma|\!\}\overline{\rho})(v)\end{aligned}$$

■

El siguiente lema prueba la primera igualdad que enunciamos y para la demostración utilizamos los lemas anteriores.

Lema 13.

$$\{\!|\Gamma, p \mapsto e|\!\}\rho = \{\!|\Gamma|\!\}(\mu\rho'.\,[\,\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')\,])$$

Demostración. La estrategia de prueba será probar que valen las siguientes desigualdades que claramente nos permiten concluir el resultado original.

1. $\{\!|\Gamma|\!\}(\mu\rho'.\,[\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')]) \sqsubseteq \{\!|\Gamma[p \mapsto e]|\!\}\rho$
2. $\{\!|\Gamma[p \mapsto e]|\!\}\rho \sqsubseteq \{\!|\Gamma|\!\}(\mu\rho'.\,[\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])$

Para probar la primera desigualdad utilizamos el lema 11, por lo tanto nos resta con probar que si $v \notin dom(\Gamma)$, entonces $(\mu\rho'.\,[\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])(v) \sqsubseteq \{\!|\Gamma[p \mapsto e]|\!\}\rho(v)$. Luego resta probar que $\{\!|\Gamma[p \mapsto e]|\!\}\rho(v)$ es cota superior de la cadena asociada al supremo del lado izquierdo de la ultima desigualdad. Por lo tanto, probemos por inducción en $n \in \mathbb{N}$ que

$$(\lambda\rho'.\,[\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])^n \bot_{Env}(v) \sqsubseteq \{\!|\Gamma[p \mapsto e]|\!\}\rho(v)$$

El caso de $n = 0$ es directo. En el caso inductivo separamos en dos casos, cuando $v = p$ y $v \neq p$, recordando que $v \notin dom(\Gamma)$.

Si $v = p$, por definición el lado izquierdo lo podemos reescribir como

$$\begin{aligned}&(\lambda\rho'.\,[\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])^{n+1}(v) \\ =\,&[\![e]\!](\{\!|\Gamma|\!\})((\lambda\rho'.\,[\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])^n \bot_{Env})\end{aligned}$$

además, utilizando el lema 9 tenemos que

$$[\![e]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho) = \{\!|\Gamma[p \mapsto e]|\!\}\rho(v)$$

Luego podemos utilizar la hipótesis inductiva sobre n en conjunto con el lema 11 y obtener

$$(\{\!|\Gamma|\!\})((\lambda\rho'.\ [\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])^n \bot_{Env}) \sqsubseteq \{\!|\Gamma[p \mapsto e]|\!\}\rho$$

Por lo tanto, utilizando la continuidad de la función $[\![e]\!]$ podemos completar la prueba.

Si $v \neq p$, además $v \notin dom(\Gamma)$ luego por definición

$$(\lambda\rho'.\ [\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')]))^{n+1}(v) = \rho(v) = (\{\!|\Gamma, p \mapsto e|\!\}\rho)(v)$$

La estrategia de prueba para la segunda desigualdad será probar que cada elemento de la cadena asociada al supremo $\{\!|\Gamma[p \mapsto e]|\!\}\rho$ es acotado superiormente por $\{\!|\Gamma|\!\}(\mu\rho'.\ [\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])$; para todo $n \in \mathbb{N}$

$$(\!|\Gamma[p \mapsto e]|\!)^n_\rho \bot_{Env} \sqsubseteq \{\!|\Gamma|\!\}(\mu\rho'.\ [\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])$$

Luego utilizando el lema 12 solo nos resta probar que para todo $n \in \mathbb{N}$ y toda variable $v \notin dom(\Gamma)$ vale

$$(\!|\Gamma[p \mapsto e]|\!)^n_\rho \bot_{Env}(v) \sqsubseteq (\mu\rho'.\ [\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])(v)$$

Procedemos por inducción en $n \in \mathbb{N}$, donde el caso base es directo. El caso inductivo lo separamos en dos casos, $v = p$ y $v \neq p$. Probaremos entonces que para todo $v \notin dom(\Gamma)$ vale

$$(\!|\Gamma[p \mapsto e]|\!)^{n+1}_\rho \bot_{Env}(v) \sqsubseteq (\mu\rho'.\ [\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])(v)$$

Si $v = p$, por definición $(\!|\Gamma[p \mapsto e]|\!)^{n+1}_\rho \bot_{Env}(v) = [\![e]\!]((\!|\Gamma[p \mapsto e]|\!)^n_\rho \bot)$. Además utilizando la hipótesis inductiva sobre n y el lema 12 tenemos que vale

$$(\!|\Gamma[p \mapsto e]|\!)^n_\rho \bot_{Env} \sqsubseteq \{\!|\Gamma|\!\}(\mu\rho'.\ [\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])$$

usando la continuidad de $[\![e]\!]$ y definiciones tenemos

$$\begin{aligned}(\!|\Gamma[p \mapsto e]|\!)^{n+1}_\rho \bot_{Env}(v) =& [\![e]\!]((\!|\Gamma[p \mapsto e]|\!)^n_\rho \bot_{Env}) \\ \sqsubseteq& [\![e]\!](\{\!|\Gamma|\!\}(\mu\rho'.\ [\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')])) \\ =& (\{\!|\Gamma|\!\}(\mu\rho'.\ [\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')]))(v)\end{aligned}$$

Si $v \neq p$, luego $(\!|\Gamma, p \mapsto e|\!)^{k+1}_\rho \bot(v) = \rho(v) = (\{\!|\Gamma, p \mapsto e|\!\}\rho)(v)$

Finalmente, probamos que $\{\!|\Gamma|\!\}(\mu\rho'.[\ \rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')\ ])$ es cubierto por $\{\!|\Gamma, p \mapsto e|\!\}\rho$ y viceversa, por lo tanto hemos probado que son iguales. ■

Lema 14.

$$\{\!|\Gamma, p \mapsto e|\!\}\rho = \{\!|\Gamma|\!\}([\ \rho \mid p : [\![e]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho)\ ])$$

Demostración. La prueba es análoga a la prueba del lema anterior, probando que

1. $\{\!|\Gamma, p \mapsto e|\!\}\rho \sqsubseteq \{\!|\Gamma|\!\}([\ \rho \mid p : [\![e]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho)\ ])$
2. $\{\!|\Gamma|\!\}([\ \rho \mid p : [\![e]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho)\ ]) \sqsubseteq \{\!|\Gamma, p \mapsto e|\!\}\rho$

■

3.4 CORRECCIÓN

El enunciado de Corrección que probaremos en esta sección es un enunciado comparable al propuesto por Launchbury; más allá de la definición de semántica operacional utilizada (nuestra prueba usa la versión revisada de Sestoft), definimos una nueva relación entre heaps que reemplazará en el enunciado original la relación entre entornos.

Recordemos, dados ρ y ρ' diremos que $\rho \leqslant \rho'$ cuando para cualquier variable x, si $\rho(x) \neq \bot$ entonces $\rho(x) = \rho'(x)$. Notar esto implica que un entorno estará más definido que otro cuando coincidiendo en el valor definido de todas las variables de este ultimo, además define el valor denotacional de nuevas variables; si $\rho \leqslant \rho'$ y para alguna variable x, $\rho(x) = \bot$ entonces tal vez $\rho'(x) \neq \bot$. Nuestra definición que reemplazará a esta relación contempla explícitamente cuales son las nuevas variables definidas ya que la relación es sobre los heaps que realizan las actualizaciones sobre un determinado entorno. Notar que claramente esta relación es reflexiva y transitiva.

Definición 12 (Relación de orden entre heaps)**.** *Diremos que Δ agrega nuevas referencias a Γ con respecto al entorno ρ cuando:*

$$\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\Delta) \text{ y para toda } x \in \mathrm{dom}(\Gamma),\ \{\!|\Gamma|\!\}\rho(x) = \{\!|\Delta|\!\}\rho(x)$$

Denotamos a esta relación como $\Gamma \preceq_\rho \Delta$.

Lema 15. *La relación $\preceq_\rho$ es reflexiva y transitiva.*

El siguiente lema demuestra que efectivamente el entorno ρ actualizado con respecto al heap $\Gamma[p \mapsto e]$ está más definido al actualizarlo con respecto a Γ.

Lema 16. *Dados Γ y ρ, tal que p no ocurre en Γ entonces*

$$\Gamma \preceq_\rho \Gamma[p \mapsto e]$$

Demostración. Por definición tenemos que probar que para toda variable $v \in dom(\Gamma)$ vale

$$\{\!|\Gamma|\!\}\rho(v) = \{\!|\Gamma[p \mapsto e]|\!\}\rho(v)$$

lo que probaremos es que las cadenas asociadas a estos supremos son iguales punto a punto; recordemos que semántica de los heaps esta definida tomando el supremo de las siguientes cadenas. Para todo $n \in \mathbb{N}$ y $v \in (\{FV(e) \mid e \in img(\Gamma)\} \cup dom(\Gamma))$

$$(\!|\Gamma|\!)^n_\rho \bot_{Env})(v) = (\!|\Gamma[p \mapsto e]|\!)^n_\rho \bot_{Env})(v)$$

Procedemos por inducción en $n \in \mathbb{N}$, donde el caso base es directo. Luego probamos que para toda $v \in (\{FV(e) \mid e \in img(\Gamma)\} \cup dom(\Gamma))$

$$(\!|\Gamma|\!)^{n+1}_\rho \bot_{Env})(v) = (\!|\Gamma[p \mapsto e]|\!)^{n+1}_\rho \bot_{Env})(v)$$

Si $v \notin dom(\Gamma)$, entonces la prueba es directa de notar que ambos lados de la igualdad son iguales por definición a $\rho(v)$. En cambio si $v \in dom(\Gamma)$ entonces

$$\begin{aligned}
(\!|\Gamma|\!)^{n+1}_\rho \bot_{Env}(v) &= [\![\Gamma v]\!](\!|\Gamma|\!)^n_\rho \bot_{Env}) \\
(\!|\Gamma[p \mapsto e]|\!)^{n+1}_\rho \bot_{Env}(v) &= [\![\Gamma[p \mapsto e]v]\!](\!|\Gamma[p \mapsto e]|\!)^n_\rho \bot_{Env}) \\
&= [\![\Gamma v]\!](\!|\Gamma[p \mapsto e]|\!)^n_\rho \bot_{Env})
\end{aligned}$$

notar que $v \neq p$ ya que p se tomo fresco con respecto a Γ. Por hipótesis inductiva tenemos que para toda variable $w \in (\{FV(e) \mid e \in img(\Gamma)\} \cup dom(\Gamma)$ entonces $(\!|\Gamma|\!)^n_\rho \bot_{Env})(w) = (\!|\Gamma[p \mapsto e]|\!)^n_\rho \bot_{Env})(w)$. Luego tenemos que para toda variable $w \in FV(\Gamma v)$ vale esta ultima igualdad. Por lo tanto utilizando el lema 10 (de Coincidencia) completamos la prueba, ya que del lema obtenemos

$$[\![\Gamma v]\!](\!|\Gamma|\!)^n_\rho \bot_{Env}) = [\![\Gamma v]\!](\!|\Gamma[p \mapsto e]|\!)^n_\rho \bot_{Env})$$

■

El siguiente resultado se desprende del último lema y la propiedad transitiva de la relación entre heaps.

Lema 17. *Dados* Γ *y* ρ*, tal que* p_i *no ocurre en* Γ *entonces*

$$\Gamma \preceq_\rho \Gamma[p_1 \mapsto e_1] \dots [p_n \mapsto e_n]$$

Dados dos heaps podemos construir un nuevo heap que será el resultado de unir todas las referencias de uno y otro, esto solo será valido si los dominios de ambos heaps son disjuntos.

Definición 13 (Unión de Heaps)**.** *Sean* Γ *y* Δ *tal que* $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$ *entonces* $(\Gamma \cup \Delta)$ *denotará la unión tal que*

$$(\Gamma \cup \Delta)\, x = \begin{cases} \Gamma x & x \in \mathrm{dom}(\Gamma) \\ \Delta x & x \in \mathrm{dom}(\Delta) \end{cases}$$

En el caso particular tengamos dos heaps cuyos dominios sean totalmente disjuntos, la actualización de un entorno con respecto a estos heaps se puede realizar mediante dos actualizaciones o una actualización en la cual unimos los heaps. Notar que en el siguiente lema, nada prohíbe que expresiones contenidas en la $\mathrm{img}(\Gamma)$ contengan como variables libres punteros de $\mathrm{dom}(\Delta)$; sin embargo sí prohíbe que expresiones en $\mathrm{img}(\Delta)$ tengan variables libres contenidas en $\mathrm{dom}(\Gamma)$.

Lema 18. *Dados* Γ*,* Δ *heaps y un entorno* ρ *tal que* $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$ *y* $\mathrm{dom}(\Gamma) \cap \{FV(e) \mid e \in \mathrm{img}(\Delta)\} = \emptyset$ *entonces*

$$\{\!|\Gamma|\!\}(\{\!|\Delta|\!\}\rho) = \{\!|\,(\Gamma \cup \Delta)\,|\!\}\rho$$

Demostración. Sea $\rho_\Delta = \{\!|\Delta|\!\}\rho$. La prueba estará separada en dos casos:

1. Para todo $n \in \mathbb{N}$, $(\!|\Gamma|\!)^n_{\rho_\Delta} \bot_{Env} \sqsubseteq \{\!|\,(\Gamma \cup \Delta)\,|\!\}\rho$
2. Para todo $n \in \mathbb{N}$, $(\!|(\Gamma \cup \Delta)|\!)^n_\rho \bot_{Env} \sqsubseteq \{\!|\Gamma|\!\}\rho_\Delta$

Luego podemos concluir $\{\!|\Gamma|\!\}\rho_\Delta \sqsubseteq \{\!|\,(\Gamma \cup \Delta)\,|\!\}\rho$ y $\{\!|\,(\Gamma \cup \Delta)\,|\!\}\rho \sqsubseteq \{\!|\Gamma|\!\}\rho_\Delta$ respectivamente para completar la prueba por antisimetría.

Ambas pruebas procederán por inducción en los naturales donde el caso base es directo. Probamos entonces ambos casos inductivos en conjunto, es decir probamos que para toda variable v,

1. $(\!|\Gamma|\!)^{n+1}_{\rho_\Delta} \bot_{Env})(v) \sqsubseteq (\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho)(v)$

2. $(\!|(\Gamma \cup \Delta)|\!)^{n+1}_{\rho} \bot_{Env})(v) \sqsubseteq (\{\!|\Gamma|\!\}\rho_\Delta)(v)$

Si $v \notin (\mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Delta))$, entonces es directo por definición que

1. $$\begin{aligned}(\!|\Gamma|\!)^{n+1}_{\rho_\Delta} \bot_{Env})(v) &= (\{\!|\Delta|\!\}\rho)(v)\\ &= \rho(v)\\ &= (\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho)(v)\end{aligned}$$

2. $$\begin{aligned}(\!|(\Gamma \cup \Delta)|\!)^{n+1}_{\rho} \bot_{Env})(v) &= \rho(v)\\ &= (\{\!|\Delta|\!\}\rho)(v)\\ &= (\{\!|\Gamma|\!\}\rho_\Delta)(v)\end{aligned}$$

Si $v \in \mathrm{dom}(\Delta)$, luego utilizando el lema 9 y el lema 10 de coincidencia; para el cual para el primer punto tenemos que probar que para toda variable $w \in \mathrm{FV}(\Delta v)$ vale $\{\!|\Delta|\!\}\rho(w) = \{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho(w)$ y para el segundo punto probamos que para toda variable $w \in \mathrm{FV}(\Delta v)$ vale $(\{\!|\Gamma|\!\}\rho_\Delta)(w) = (\{\!|\Delta|\!\}\rho)(w)$. Ambas son resultado directo de notar que por hipótesis vale $(\mathrm{dom}(\Gamma) \cap \mathrm{FV}(\Delta w)) = \emptyset$,

1. $$\begin{aligned}(\!|\Gamma|\!)^{n+1}_{\rho_\Delta} \bot_{Env})(v) &= (\{\!|\Delta|\!\}\rho)(v) && \text{(Def. 11)}\\ &= [\![\Delta\, v]\!](\{\!|\Delta|\!\}\rho) && \text{(Lema 9)}\\ &= [\![\Delta\, v]\!](\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho) && \text{(Lema 10)}\\ &= [\![(\Gamma \cup \Delta)\, v]\!](\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho)\\ &= (\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho)(v) && \text{(Lema 9)}\end{aligned}$$

2. $$\begin{aligned}(\!|(\Gamma \cup \Delta)|\!)^{n+1}_{\rho} \bot_{Env})(v) &= (\!|(\Gamma \cup \Delta)|\!)_\rho((\!|(\Gamma \cup \Delta)|\!)^{n}_{\rho} \bot_{Env}))(v) && \text{(Def.)}\\ &= [\![\Delta\,]\!](\!|(\Gamma \cup \Delta)|\!)^{n}_{\rho} \bot_{Env}) && \text{(Def. 11)}\\ &\sqsubseteq [\![\Delta\, v]\!](\{\!|\Gamma|\!\}\rho_\Delta) && \text{(HI)}\\ &= [\![\Delta\, v]\!](\{\!|\Delta|\!\}\rho) && \text{(Lema 10)}\\ &= (\{\!|\Delta|\!\}\rho)(v) && \text{(Lema 9)}\\ &= (\{\!|\Gamma|\!\}\rho_\Delta)(v) && \text{(Def. 11)}\end{aligned}$$

Si $v \in \mathrm{dom}(\Gamma)$, utilizando el lema 9 y la hipótesis inductiva sobre n en conjunto con la continuidad de la función semántica $[\![\Gamma v]\!]$ tenemos que vale

1.
$$\begin{aligned}
(\!(\Gamma)\!)^{n+1}_{\rho_\Delta}\bot_{Env})(v) &= ((\!(\Gamma)\!)_{\rho_\Delta}((\!(\Gamma)\!)^{n}_{\rho_\Delta}\bot_{Env}))(v) && \text{(Def.)}\\
&= [\![\Gamma v]\!]((\!(\Gamma)\!)^{n}_{\rho_\Delta}\bot_{Env}) && \text{(Def. 11)}\\
&\sqsubseteq [\![\Gamma v]\!](\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho) && \text{(HI)}\\
&= [\![(\Gamma \cup \Delta)\, v]\!](\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho)\\
&= (\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho)(v) && \text{(Lema 9)}
\end{aligned}$$

2.
$$\begin{aligned}
(\!(\Gamma \cup \Delta)\!)^{n+1}_{\rho}\bot_{Env})(v) &= ((\!(\Gamma \cup \Delta)\!)_{\rho}((\!(\Gamma \cup \Delta)\!)^{n}_{\rho}\bot_{Env}))(v) && \text{(Def.)}\\
&= [\![\Gamma\, v]\!]((\!(\Gamma \cup \Delta)\!)^{n}_{\rho}\bot_{Env}) && \text{(Def. 11)}\\
&\sqsubseteq [\![\Gamma\, v]\!](\{\!|\Gamma|\!\}\rho_\Delta) && \text{(HI)}\\
&= (\{\!|\Gamma|\!\}\rho_\Delta)(v) && \text{(Lema 9)}
\end{aligned}$$

■

Notar que la restricción $dom(\Gamma) \cap dom(\Delta) = \emptyset$ del lema previo es un requerimiento con el fin de poder unir los heaps. Por otro lado la restricción $dom(\Gamma) \cap \{FV(e) \mid e \in img(\Delta)\} = \emptyset$ se debe a que si ocurriera que en Δ se referencia a alguna expresión cuya variable libre, digamos x, puede ser referenciada por Γ entonces notar $\{\!|\Gamma|\!\}(\{\!|\Delta|\!\}\rho)(p) = \{\!|\Delta|\!\}\rho(p)$ donde no interviene Γ a diferencia de $\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho(p)$ donde si lo hace; consideremos el siguiente ejemplo, tomemos ρ tal que $\rho(x) = \bot$, $\Delta = \{p \mapsto x\}$ y $\Gamma = \{x \mapsto \lambda y.\ \textbf{let}\ w{=}w\ \textbf{in}\ w\}$. Claramente aplicando las definiciones obtenemos diferente significado para el puntero p.

$$\begin{aligned}
\{\!|\Gamma|\!\}(\{\!|\Delta|\!\}\rho)(p) &= \{\!|\Delta|\!\}\rho(p)\\
&= [\![x]\!](\{\!|\Delta|\!\}\rho)\\
&= \{\!|\Delta|\!\}\rho(x)\\
&= \rho(x)\\
&= \bot
\end{aligned}
\qquad
\begin{aligned}
\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho(p) &= [\![x]\!](\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho)\\
&= (\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho)(x)\\
&= [\![\Gamma x]\!](\{\!|\, (\Gamma \cup \Delta) \,|\!\}\rho)\\
&= Fn(\lambda\, d.\, \bot)
\end{aligned}$$

Un resultado tan importante como el lema de Coincidencia es el lema de Substitución. Launchbury utiliza substitución en su prueba de corrección, pero nunca presenta un enunciado ni prueba. Nosotros probamos un enunciado no tan general de Substitución que solo considera la substitución de variables por variables; este es el único tipo de substitución que realizamos en la semántica operacional y permite aplicar las hipótesis inductivas correspondientes en la prueba de corrección para los casos de la aplicación y el let.

Lema 19 (Substitución). *Si* $\rho(\delta v) = \rho' v$ *para* $v \in FV(e)$ *donde* $\delta : \mathbb{V} \to \mathbb{V}$, *entonces* $[\![e/\delta]\!]\rho = [\![e]\!]\rho'$

Demostración. Procedemos por inducción estructural en la expresión e.

CASO VAR: Directo de aplicar la hipótesis.

CASO ABS: Probemos que $[\![(\lambda x.\ e)/\delta]\!]\rho = [\![\lambda x.\ e]\!]\rho'$ suponiendo que para toda variable $v \in (FV(e) - \{x\})$ se cumple $\rho(\delta v) = \rho' v$. Comenzamos probando que para toda $v \in FV(e)$ vale

$$[\rho \mid y : d]([\delta \mid x : y]v) = [\rho' \mid x : d]v$$

esta prueba es directa considerando los casos $v = x$ y $v \neq x$, notando que la variable y es fresca. Por lo tanto, por hipótesis inductiva vale

$$[\![e/[\delta \mid x : y]]\!][\rho \mid y : d] = [\![e]\!][\rho' \mid x : d] \tag{HI'}$$

luego podemos completar la prueba

$$\begin{aligned}
[\![(\lambda x.\ e)/\delta]\!]\rho &= [\![\lambda y.\ e/[\delta \mid x : y]]\!]\rho && \text{(DEF. SUBST.)}\\
&= Fn(\lambda\, d.\ [\![e/[\delta \mid x : y]]\!][\rho \mid y : d]) && \text{(DEF. 10)}\\
&= Fn(\lambda\, d.\ [\![e]\!][\rho' \mid x : d]) && \text{(HI')}\\
&= [\![\lambda x.\ e]\!]\rho' && \text{(DEF. 10)}
\end{aligned}$$

CASO APP: Supongamos para toda variable $v \in (FV(e) \cup \{p\})$ vale $\rho(\delta v) = \rho' v$ y probemos $[\![(e\ p)/\delta]\!]\rho = [\![e\ p]\!]\rho'$. Utilizando la hipótesis inductiva es directo obtener $[\![e/\delta]\!]\rho = [\![e]\!]\rho'$ y $[\![p/\delta]\!]\rho = [\![p]\!]\rho'$, por lo tanto

$$\begin{aligned}
[\![(e\ p)/\delta]\!]\rho &= [\![e/\delta\ p/\delta]\!]\rho && \text{(DEF. SUBST.)}\\
&= ([\![e/\delta]\!]\rho) \downarrow_{Fn} ([\![p/\delta]\!]\rho) && \text{(DEF. 10)}\\
&= ([\![e]\!]\rho') \downarrow_{Fn} ([\![p]\!]\rho') && \text{(HI)}\\
&= [\![e\ p]\!]\rho' && \text{(DEF. 10)}
\end{aligned}$$

CASO LET: Este es el caso interesante y más complejo, suponemos para toda variable $v \in ((FV(e) \cup FV(e_i)) - \{x_i\})$ vale $\rho(\delta v) = \rho' v$ y probemos

$$\begin{aligned}
[\![(\textbf{let}\ x_i{=}e_i\ \textbf{in}\ e)/\delta]\!]\rho &= [\![\textbf{let}\ y_i{=}e_i/\delta'\ \textbf{in}\ e/\delta']\!]\rho && \text{(DEF. SUBST.)}\\
&= [\![e/\delta']\!](\{\!|y_i \mapsto e_i/\delta'|\!\}\rho) && \text{(DEF. 10)}\\
&= [\![e]\!](\{\!|x_i \mapsto e_i|\!\}\rho') && \text{(HI')}\\
&= [\![\textbf{let}\ x_i{=}e_i\ \textbf{in}\ e]\!]\rho' && \text{(DEF. 10)}
\end{aligned}$$

donde $\delta' = [\delta \mid x_i : y_i]$, por lo tanto el paso relevante que nos resta probar es

$$[\![e/\delta']\!](\{\!|y_i \mapsto e_i/\delta'|\!\}\rho) = [\![e]\!](\{\!|x_i \mapsto e_i|\!\}\rho') \tag{HI'}$$

Para utilizar la hipótesis inductiva con respecto a e debemos probar que para toda variable libre $v \in FV(e)$ vale $(\{\!|y_i \mapsto e_i/\delta'|\!\}\rho)(\delta' v) = (\{\!|x_i \mapsto e_i|\!\}\rho')(v)$. Probaremos entonces que las cadenas de estos supremos son iguales punto a punto; para todo $n \in \mathbb{N}$ y $v \in (FV(e) \cup FV(e_i))$ vale

$$(\!|y_i \mapsto e_i/\delta'|\!)^n_\rho \bot_{Env})(\delta' v) = (\!|x_i \mapsto e_i|\!)^n_{\rho'} \bot_{Env})(v)$$

el caso $n = 0$ es directo, probemos el caso inductivo separando en los casos $v \neq x_i$ y $v = x_i$.

Si $v \notin \{x_i\}$, luego por definición de la semántica de heaps y la hipótesis principal de substitución

$$\begin{aligned}
(\!|y_i \mapsto e_i/\delta'|\!)^{n+1}_\rho \bot_{Env})(\delta' v) &= \rho(\delta' v) && \text{(Def. 11)}\\
&= \rho(\delta v) && (v \neq x_i)\\
&= \rho' v && \text{(Hip.)}\\
&= (\!|x_i \mapsto e_i|\!)^{n+1}_{\rho'} \bot_{Env})(v) && \text{(Def. 11)}
\end{aligned}$$

Si $v = x_i$ para algún i, utilizamos la hipótesis inductiva sobre n en conjunto con la hipótesis inductiva respecto de e_i

$$\begin{aligned}
(\!|y_i \mapsto e_i/\delta'|\!)^{n+1}_\rho \bot_{Env})(\delta' v) &= (\!|y_i \mapsto e_i/\delta'|\!)^{n+1}_\rho \bot_{Env})(y_i) && (v = x_i)\\
&= [\![e_i/\delta']\!](\!|y_i \mapsto e_i/\delta'|\!)^n_\rho \bot_{Env}) && \text{(Def. 11)}\\
&= [\![e_i]\!](\!|x_i \mapsto e_i|\!)^n_{\rho'} \bot_{Env}) && \text{(HI)}\\
&= (\!|x_i \mapsto e_i|\!)^{n+1}_{\rho'} \bot_{Env})(v) && \text{(Def. 11)}
\end{aligned}$$

Por lo tanto, probamos que toda variable libre $v \in FV(e)$ vale

$$(\{\!|y_i \mapsto e_i/\delta'|\!\}\rho)(\delta' v) = (\{\!|x_i \mapsto e_i|\!\}\rho')(v)$$

y podemos completar la prueba utilizando la hipótesis inductiva respecto de e.

■

Finalmente ya tenemos todas las definiciones y lemas necesarios para enunciar y probar nuestra versión del teorema de Corrección. Este enunciado es equivalente mediante definiciones al enunciado de Corrección Generalizada probado por Breitner [7].

Teorema 1 (Corrección). *Si* $\Gamma : e \Downarrow_A \Delta : z$ *entonces para todo entorno* ρ,

$$[\![e]\!]\{\!|\Gamma|\!\}\rho = [\![z]\!]\{\!|\Delta|\!\}\rho \quad y \quad \Gamma \preceq_\rho \Delta$$

Demostración. Procedemos por inducción estructural sobre la derivación.

CASO ABS: Directo.

CASO APP: En el caso que la última regla aplicada fuera APP

$$\frac{\Gamma : e \Downarrow_A \Theta : \lambda x.\, e' \qquad \Theta : e'/[x : p] \Downarrow_A \Delta : z}{\Gamma : e\ p \Downarrow_A \Delta : z}\ \text{APP}$$

Probar $\Gamma \preceq_\rho \Delta$ es directo por transitividad en las hipótesis inductivas $\Gamma \preceq_\rho \Theta$ y $\Theta \preceq_\rho \Delta$. Tenemos las hipótesis inductivas

1. $[\![e]\!]\{\!|\Gamma|\!\}\rho = [\![\lambda x.\, e']\!]\{\!|\Theta|\!\}\rho = \mathrm{Fn}(\lambda\, d.\, [\![e']\!][\{\!|\Theta|\!\}\rho \mid x : d])$
2. $[\![e'/[x : p]]\!]\{\!|\Theta|\!\}\rho = [\![z]\!]\{\!|\Delta|\!\}\rho$

probaremos $[\![e\ p]\!]\{\!|\Gamma|\!\}\rho = [\![z]\!]\{\!|\Delta|\!\}\rho$. Es fácil probar que $[\![p]\!](\{\!|\Gamma|\!\}\rho) = [\![p]\!](\{\!|\Theta|\!\}\rho)$; si $p \in \mathrm{dom}(\Gamma)$ entonces es directo ya que $\Gamma \preceq_\rho \Theta$, si $p \notin \mathrm{dom}(\Gamma)$ como $\Gamma : e'$ es A-good luego $p \in A$ entonces utilizando el lema 8 tenemos $p \notin \mathrm{dom}(\Theta)$. Por lo tanto $[\![p]\!](\{\!|\Gamma|\!\}\rho) = \rho(y)\ [\![p]\!](\{\!|\Theta|\!\}\rho)$. Aplicando definiciones y la hipótesis inductiva con respecto a e' tenemos

$$\begin{aligned}
[\![e\ p]\!]\{\!|\Gamma|\!\}\rho &= ([\![e]\!]\{\!|\Gamma|\!\}\rho) \downarrow_{\mathrm{Fn}} ([\![p]\!]\{\!|\Gamma|\!\}\rho) && \text{(DEF. 10)}\\
&= (\mathrm{Fn}(\lambda\, d.\, [\![e']\!][\{\!|\Theta|\!\}\rho \mid x : d])) \downarrow_{\mathrm{Fn}} ([\![p]\!]\{\!|\Gamma|\!\}\rho) && \text{(HI 1)}\\
&= [\![e']\!][\{\!|\Theta|\!\}\rho \mid x : ([\![p]\!]\{\!|\Gamma|\!\}\rho)]\\
&= [\![e']\!][\{\!|\Theta|\!\}\rho \mid x : ([\![p]\!]\{\!|\Theta|\!\}\rho)]\\
&= [\![e'/[x : p]]\!]\{\!|\Theta|\!\}\rho && \text{(LEMA 19: SUBSTITUCIÓN)}\\
&= [\![z]\!]\{\!|\Delta|\!\}\rho && \text{(HI 2)}
\end{aligned}$$

Para completar la prueba utilizamos el lema 19 (de Substitución) con la substitución $[x : p]$, para esto notar que es directo probar que para toda variable $v \in \mathrm{FV}(e')$ vale $(\{\!|\Theta|\!\}\rho)([x : p]\, v) = ([\{\!|\Theta|\!\}\rho \mid x : ([\![p]\!]\{\!|\Theta|\!\}\rho)])(v)$.

CASO VAR: En el caso de la regla VAR

$$\frac{\Gamma : e \Downarrow_{A \cup \{p\}} \Delta : z}{\Gamma[p \mapsto e] : p \Downarrow_A \Delta[p \mapsto z] : z}\ \textsc{Var}$$

Las hipótesis inductivas serán: para cualquier entorno ρ vale $\Gamma \preceq_\rho \Delta$ y $[\![e]\!](\{\!|\Gamma|\!\}\rho) = [\![z]\!](\{\!|\Delta|\!\}\rho)$, luego probaremos utilizando los lemas 9, 13 y dos veces la hipótesis inductiva

$$\begin{aligned}
[\![p]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho) &= (\{\!|\Gamma[p \mapsto e]|\!\}\rho)(p) && \text{(DEF. 10)}\\
&= [\![e]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho) && \text{(LEMA 9)}\\
&= [\![e]\!](\{\!|\Gamma|\!\}(\mu\rho'.[\,\rho \mid p : [\![e]\!](\{\!|\Gamma|\!\}\rho')\,])) && \text{(LEMA 13)}\\
&= [\![e]\!](\{\!|\Gamma|\!\}(\mu\rho'.[\,\rho \mid p : [\![z]\!](\{\!|\Delta|\!\}\rho')\,])) && \text{(HI)}\\
&= [\![z]\!](\{\!|\Delta|\!\}(\mu\rho'.[\,\rho \mid p : [\![z]\!](\{\!|\Delta|\!\}\rho')\,])) && \text{(HI)}\\
&= [\![z]\!](\{\!|\Delta[p \mapsto z]|\!\}\rho) && \text{(LEMA 9)}
\end{aligned}$$

Notar que un resultado que se desprende que utilizaremos a continuación es $[\![e]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho) = [\![z]\!](\{\!|\Delta[p \mapsto z]|\!\}\rho)$. Resta probar $\Gamma[p \mapsto e] \preceq_\rho \Delta[p \mapsto z]$; para toda variable $v \in (\mathrm{dom}(\Gamma) \cup \{p\})$ vale $\{\!|\Gamma[p \mapsto e]|\!\}\rho(v) = \{\!|\Delta[p \mapsto z]|\!\}\rho(v)$. Para probar esto transformamos las actualizaciones utilizando el lema 14 y luego completamos la prueba usando lo previamente probado e hipótesis inductiva

$$\begin{aligned}
(\{\!|\Gamma[p \mapsto e]|\!\}\rho)(v) &= (\{\!|\Gamma|\!\}([\,\rho \mid p : [\![e]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho)\,]))(v) && \text{(LEMA 14)}\\
&= (\{\!|\Delta|\!\}([\,\rho \mid p : [\![e]\!](\{\!|\Gamma[p \mapsto e]|\!\}\rho)\,]))(v) && \text{(HI)}\\
&= (\{\!|\Delta|\!\}([\,\rho \mid p : [\![z]\!](\{\!|\Delta[p \mapsto z]|\!\}\rho)\,]))(v)\\
&= (\{\!|\Delta[p \mapsto z]|\!\}\rho)(v) && \text{(LEMA 14)}
\end{aligned}$$

CASO LET: En el ultimo caso de prueba tenemos la regla LET

$$\frac{\Gamma[p_i \mapsto \tilde{e}_i] : \tilde{e} \Downarrow_A \Delta : z}{\Gamma : \mathbf{let}\ x_i{=}e_i\ \mathbf{in}\ e \Downarrow_A \Delta : z}\ \textsc{Let}$$

Lo primero que probamos será $\Gamma \preceq_\rho \Delta$, lo cual se desprende rápidamente de utilizar el lema 17 que nos permite obtener $\Gamma \preceq_\rho \Gamma[p_i \mapsto \tilde{e}_i]$ ya que los punteros p_i son frescos con respecto a Γ, A y **let** $x_i{=}e_i$ **in** e. Además nuestra hipótesis inductiva nos asegura $\Gamma[p_i \mapsto \tilde{e}_i] \preceq_\rho \Delta$, por lo tanto por transitividad completamos la prueba.

La hipótesis inductiva además nos garantiza que para todo entorno ρ vale $[\![\tilde{e}]\!](\{\!|\Gamma[p_i \mapsto \tilde{e}_i]|\!\}\rho) = [\![z]\!](\{\!|\Delta|\!\}\rho)$. Luego solo nos resta probar

$$[\![\mathbf{let}\ x_i{=}e_i\ \mathbf{in}\ e]\!](\{\!|\Gamma|\!\}\rho) = [\![z]\!](\{\!|\Delta|\!\}\rho)$$

Por definición $[\![\textbf{let } x_i{=}e_i \textbf{ in } e]\!](\{\!|\Gamma|\!\}\rho) = [\![e]\!](\{\!|x_i \mapsto e_i|\!\}(\{\!|\Gamma|\!\}\rho))$. Notar que $\Gamma[\,p_i \mapsto \tilde{e}_i\,]$ es equivalente a la unión $((p_i \mapsto \tilde{e}_i) \cup \Gamma)$, luego podemos reescribir la hipótesis inductiva utilizando el lema 18

$$\begin{aligned}
[\![z]\!](\{\!|\Delta|\!\}\rho) &= [\![\tilde{e}]\!](\{\!|\Gamma[\,p_i \mapsto \tilde{e}_i\,]|\!\}\rho) && \text{(HI)}\\
&= [\![\tilde{e}]\!](\{\!|\,((p_i \mapsto \tilde{e}_i) \cup \Gamma)\,|\!\}\rho)\\
&= [\![\tilde{e}]\!](\{\!|p_i \mapsto \tilde{e}_i|\!\}(\{\!|\Gamma|\!\}\rho)) && \text{(Lema 18)}
\end{aligned}$$

es fácil ver que las condiciones de este lema se cumplen: es decir $\{p_i\} \cap \mathrm{dom}(\Gamma) = \emptyset$ y $\{p_i\} \cap \{FV(e) \mid e \in \mathrm{img}(\Gamma)\} = \emptyset$, ya que los punteros p_i se toman frescos con respecto a todas las variables involucradas en la derivación.

Si probamos $[\![e]\!](\{\!|x_i \mapsto e_i|\!\}(\{\!|\Gamma|\!\}\rho)) = [\![\tilde{e}]\!](\{\!|p_i \mapsto \tilde{e}_i|\!\}(\{\!|\Gamma|\!\}\rho))$ entonces hemos completado la prueba, recordemos que $\tilde{e} = e/[x_i : p_i]$ y $\tilde{e_i} = e_i/[x_i : p_i]$. Utilizaremos el lema 19 (de Substitución), donde la substitución será $[x_i : p_i]$ y la condición del lema que probaremos es que para toda variable $v \in FV(e)$ vale

$$(\{\!|x_i \mapsto e_i|\!\}(\{\!|\Gamma|\!\}\rho))(v) = (\{\!|p_i \mapsto \tilde{e}_i|\!\}(\{\!|\Gamma|\!\}\rho))([x_i : p_i]v)$$

Probaremos entonces que las cadenas asociadas a estos supremos son iguales punto a punto y por lo tanto el supremo es el mismo. Denotamos con ρ_Γ al entorno actualizado $\{\!|\Gamma|\!\}\rho$ luego probamos por inducción sobre $n \in \mathbb{N}$ que para toda variable $v \in (FV(e) \cup FV(e_i))$ vale

$$(\!(x_i \mapsto e_i)\!)^n_{\rho_\Gamma} \bot_{Env})(v) = (\!(p_i \mapsto \tilde{e}_i)\!)^n_{\rho_\Gamma} \bot_{Env})([x_i : p_i]v)$$

El caso base cuando $n = 0$ es directo, luego solo nos resta probar el caso inductivo

$$(\!(x_i \mapsto e_i)\!)^{n+1}_{\rho_\Gamma} \bot_{Env})(v) = (\!(p_i \mapsto \tilde{e}_i)\!)^{n+1}_{\rho_\Gamma} \bot_{Env})([x_i : p_i]v)$$

Si $v \notin \{x_i\}$, entonces la prueba es directa por definición

$$\begin{aligned}
(\!(x_i \mapsto e_i)\!)^{n+1}_{\rho_\Gamma} \bot_{Env})(v) &= \rho_\Gamma(v)\\
&= (\!(p_i \mapsto \tilde{e}_i)\!)^{n+1}_{\rho_\Gamma} \bot_{Env})(v)\\
&= (\!(p_i \mapsto \tilde{e}_i)\!)^{n+1}_{\rho_\Gamma} \bot_{Env})([x_i : p_i]v)
\end{aligned}$$

Si $v = x_i$ para algún i, por definición de la semántica de heaps queremos probar que los siguientes lados derechos son iguales

$$\begin{aligned}
(\!(x_i \mapsto e_i)\!)^{n+1}_{\rho_\Gamma} \bot_{Env})(v) &= [\![e_i]\!](\!(x_i \mapsto e_i)\!)^n_{\rho_\Gamma} \bot_{Env})\\
(\!(p_i \mapsto \tilde{e}_i)\!)^{n+1}_{\rho_\Gamma} \bot_{Env})([x_i : p_i]v) &= [\![\tilde{e}_i]\!](\!(p_i \mapsto \tilde{e}_i)\!)^n_{\rho_\Gamma} \bot_{Env})
\end{aligned}$$

pero esto es directo de utilizar el lema 19 (de Substitución) con substitución $[x_i : p_i]$ y la hipótesis inductiva sobre n.

Por lo tanto, probamos que para toda variable $v \in FV(e)$ vale

$$(\{\!|x_i \mapsto e_i|\!\}(\{\!|\Gamma|\!\}\rho))(v) = (\{\!|p_i \mapsto \tilde{e}_i|\!\}(\{\!|\Gamma|\!\}\rho))([x_i : p_i]v) \ .$$

Luego utilizando el lema de Substitución probamos

$$[\![e]\!](\{\!|x_i \mapsto e_i|\!\}(\{\!|\Gamma|\!\}\rho)) = [\![\tilde{e}]\!](\{\!|p_i \mapsto \tilde{e}_i|\!\}(\{\!|\Gamma|\!\}\rho))$$

y por lo tanto completamos la prueba.

■

4

COHERENCIA Y SUBTIPADO

Como mencionamos en la introducción los dos enfoques más utilizados para especificar la semántica de un lenguaje de programación son el operacional y el denotacional, el primero describe mejor la estrategia de evaluación y el segundo es mucho más abstracto. Si un lenguaje de programación tiene definida su semántica tanto operacionalmente como denotacionalmente entonces uno espera que la semántica operacional sea *adecuada* con respecto a la semántica denotacional; esto nos asegura que si la semántica de un programa es algún objeto matemático entonces el programa computa a la representación de ese objeto. Además, existen dos estilos de semántica denotacional para lenguajes tipados: la *extrínseca* que asigna significado para toda expresión (incluso mal tipada) y la *intrínseca* que solo da semántica a expresiones bien tipadas. El modelo necesario para la semántica intrínseca es relativamente simple (no hay necesidad de resolver ecuaciones recursivas de dominio), sin embargo uno necesita probar la *coherencia* de la semántica, esto es que diferentes pruebas del mismo juicio de tipado de una expresión tengan el mismo significado. Por otro lado, la semántica extrínseca necesita resolver ecuaciones recursivas de dominio [47]. Reynolds demostró que se puede obtener una prueba de la coherencia de la semántica intrínseca a través de la semántica extrínseca utilizando relaciones lógicas.

El capítulo está dividido en dos etapas, comenzamos definiendo un lenguaje call-by-value con un sistema de tipos simple y especificamos su semántica operacional y semánticas denotacionales extrínseca e intrínseca. Probamos que las semánticas denotacionales son equivalentes; esto es interesante porque nos conduce a una prueba directa de la *coherencia* de la semántica intrínseca. Además, probamos la adecuación de la semántica operacional con respecto a la denotacional utilizando la técnica de *biortogonalidad* [35]; esta estrategia modular nos permite probar adecuación para programas convergentes y utilizando step-indexing [36] complementamos biortogonalidad para probar la adecuación de programas divergentes. Una vez exploradas las técnicas para obtener la coherencia y adecuación para el lenguaje simple, lo extendemos de manera que el sistema de tipos soporte subtipado y extendemos la prueba de coherencia y adecuación.

La novedad principal de este capítulo consiste en el uso de biortogonalidad con una semántica extrínseca, lo que según nuestro conocimiento no ha sido suficientemente explorado (solo hemos encontrado el trabajo de Berger [4]). Por otro lado, también constituye una novedad importante la mecanización del teorema de *teorema de bracketing* enunciado por John Reynolds, como parte de la formalización de todo el capítulo.

4.1 LENGUAJE

En esta sección introducimos el lenguaje simple dando su sintaxis, sistema de tipos y las semánticas operacional y denotacional. El lenguaje es una extensión del utilizado por Pitts en su tutorial [36].

SINTAXIS La definición de nuestro lenguaje está dada en términos de ANF (por applicative normal form) y por lo tanto distinguimos entre valores V y expresiones Λ; la conversión a una sintaxis menos restringida se puede realizar mediante un pre-procesamiento y la expresión *let* como ocurría en el capítulo anterior; supongamos tenemos la aplicación $\mathsf{e}\,\mathsf{e}'$, luego la estrategia general es utilizar la expresión let para introducir dos variables, digamos x y x', referenciando a cada sub-expresión y en el cuerpo colocar la expresión $\mathsf{x}\,\mathsf{x}'$. Asumimos las siguientes convenciones, sea $\mathbb{V}$ un conjunto numerable de variables, x y f serán las meta-variables sobre $\mathbb{V}$ y n pertenecerá a $\mathbb{N}$; $\oplus$ es cualquier operador aritmético.

Definición 14 (Sintaxis)**.**

Valores

$$\mathsf{v} \in V ::= \mathsf{x} \mid \lfloor \mathsf{n} \rfloor \mid \textbf{\textit{fun}}\ \mathsf{f}\ \mathsf{x} = \mathsf{e}$$

Expresiones

$$\mathsf{e} \in \Lambda ::= \mathsf{v} \mid \mathsf{v}\,\mathsf{v}' \mid \mathsf{e} \oplus \mathsf{e}' \mid \textbf{\textit{let}}\ \mathsf{x}{=}\mathsf{e}\ \textbf{\textit{in}}\ \mathsf{e}' \mid \textbf{\textit{if}}\ \mathsf{e}\ \textbf{\textit{then}}\ \mathsf{e}'\ \textbf{\textit{else}}\ \mathsf{e}''$$

En la declaración de la función **fun** $\mathsf{f}\ \mathsf{x} = \mathsf{e}$ tanto f como x están ligados en e, mientras que en **let** $\mathsf{x}{=}\mathsf{e}$ **in** e' sólo está ligada en e'; es decir, la declaración let no es recursiva. Para $X \subseteq \mathbb{V}$, definimos $V(X)$ y $\Lambda(X)$, respectivamente, como el conjunto de valores y expresiones tales que sus variables libres pertenecen a X. En la formalización, utilizamos valores y expresiones "well-scoped" (en el sentido que no existen términos sino términos bajo un cierto contexto) semejantes a la filosofía de representación de términos fuertemente tipada de Benton et al. [3].

SEMÁNTICA OPERACIONAL Dar una semántica operacional implica definir un conjunto de reglas de *reducción* que especifican como evaluar una expresión aplicando repetidamente las reglas hasta que alcanzamos un valor. En contraste con Pitts (que define una semántica big-step que permite construir pruebas de que la evaluación de una expresión termina) la semántica operacional que definimos consiste en una relación de reducción entre configuraciones, que son pares compuestos de un *frame stack* y una expresión. Diremos que la evaluación ha concluido cuando la expresión que forma parte de la configuración es un valor y el stack esta vacío; si el stack no esta vacío, el elemento superior del stack servirá como un contexto ("zippers" para expresiones, como menciona Danvy [11]) para combinarlo con el valor y producir una nueva expresión que es utilizada para continuar con la evaluación. Si la expresión de la configuración no es un valor, entonces colocamos en el tope del stack un nuevo contexto y evaluamos alguna sub-expresión.

Definición 15 (Frame Stack)**.**

$$\begin{aligned} \mathsf{E} \in \Lambda^\star ::= {} & \mathsf{Id} \mid \mathsf{E} \circ (\mathsf{x} \mapsto \mathsf{e}) \mid \mathsf{E} \circ (\Box \oplus \mathsf{e}) \\ & \mid \mathsf{E} \circ (\mathsf{e} \oplus \Box) \mid \mathsf{E} \circ (\mathit{if}\, \Box\, (\mathsf{e}, \mathsf{e}')) \end{aligned}$$

En $\mathsf{E} \circ (\mathsf{x} \mapsto \mathsf{e})$, x funciona como variable ligadora en e; tal como para valores y expresiones, definimos a $\Lambda^\star(X)$ como el conjunto de frame stacks tales que sus variables libres se encuentran en X.

La relación de reducción está definida inductivamente por el siguiente conjunto de reglas de reescritura de la forma $(\mathsf{E}, \mathsf{e}) \longmapsto (\mathsf{E}', \mathsf{e}')$. Como es usual $\mathsf{e}[\mathsf{x}/\mathsf{v}]$ denota la substitución (que evita la captura de variables) de v por las ocurrencias libres de x en e.

Definición 16 (Reglas Operacionales)**.**

$$\begin{array}{lcll}
\mathsf{E}, \mathit{let}\ \mathsf{x}{=}\mathsf{e}\ \mathit{in}\ \mathsf{e}' & \longmapsto & \mathsf{E} \circ (\mathsf{x} \mapsto \mathsf{e}'), \mathsf{e} & (\textsc{let}) \\
\mathsf{E} \circ (\mathsf{x} \mapsto \mathsf{e}), \mathsf{v} & \longmapsto & \mathsf{E}, \mathsf{e}[\mathsf{x}/\mathsf{v}] & (\textsc{subst}) \\
\mathsf{E}, (\mathit{fun}\ \mathsf{f}\ \mathsf{x} = \mathsf{e})\, \mathsf{v} & \longmapsto & \mathsf{E}, \mathsf{e}[\mathsf{x}/\mathsf{v}, \mathsf{f}/\mathit{fun}\ \mathsf{f}\ \mathsf{x} = \mathsf{e}] & (\textsc{ap}) \\
\mathsf{E}, \mathsf{e} \oplus \mathsf{e}' & \longmapsto & \mathsf{E} \circ (\Box \oplus \mathsf{e}'), \mathsf{e} & (\textsc{op}) \\
\mathsf{E} \circ (\Box \oplus \mathsf{e}'), \lfloor \mathsf{n} \rfloor & \longmapsto & \mathsf{E} \circ (\lfloor \mathsf{n} \rfloor \oplus \Box), \mathsf{e}' & (\textsc{op-l}) \\
\mathsf{E} \circ (\lfloor \mathsf{n} \rfloor \oplus \Box), \lfloor \mathsf{m} \rfloor & \longmapsto & \mathsf{E}, \lfloor \mathsf{n} \oplus \mathsf{m} \rfloor & (\textsc{op-r}) \\
\mathsf{E}, \mathit{if}\ \mathsf{e}\ \mathit{then}\ \mathsf{e}'\ \mathit{else}\ \mathsf{e}'' & \longmapsto & \mathsf{E} \circ (\mathit{if}\, \Box\, (\mathsf{e}', \mathsf{e}'')), \mathsf{e} & (\textsc{if}) \\
\mathsf{E} \circ (\mathit{if}\, \Box\, (\mathsf{e}, \mathsf{e}')), \lfloor 0 \rfloor & \longmapsto & \mathsf{E}, \mathsf{e} & (\textsc{ifz}) \\
\mathsf{E} \circ (\mathit{if}\, \Box\, (\mathsf{e}, \mathsf{e}')), \lfloor \mathsf{n} + 1 \rfloor & \longmapsto & \mathsf{E}, \mathsf{e}' & (\textsc{ifs})
\end{array}$$

Notar que la regla (AP) expone explícitamente que las funciones son potencialmente recursivas: si f no ocurre en e entonces **fun** f x = e se comporta como la abstracción λx. e. Notar además que la estrategia de evaluación call-by-value es implementada primero evaluando las expresiones introducidas por el let.

La evaluación se define tomando la clausura reflexiva y transitiva de las reglas de reducción; diremos que una expresión cerrada e *evalúa* a v si $\mathsf{Id}, \mathsf{e} \longmapsto^* \mathsf{Id}, \mathsf{v}$. Además, diremos que e *diverge* con el frame stack E, denotado $(\mathsf{E}, e) \longmapsto^* \infty$, si la secuencia de reescritura $(\mathsf{E}, \mathsf{e}) \longmapsto (\mathsf{E}_1, \mathsf{e}_1) \longmapsto (\mathsf{E}_2, \mathsf{e}_2) \longmapsto \ldots$ es infinita.

SEMÁNTICA DENOTACIONAL EXTRÍNSECA En la semántica denotacional extrínseca uno da significado a las expresiones y valores independientemente si están o no bien tipadas, lo cual resulta en una situación similar a la del calculo lambda sin tipos donde uno necesita modelar la auto-aplicación como mencionamos en la sección 2.1 del capítulo 2; recordemos que la ecuación recursiva de dominios se puede generalizar definiendo un endofunctor. En esta sección vamos a elegir $D = \mathcal{V}_\perp$, donde $\perp$ es la denotación de las expresiones divergentes y $\mathcal{V}$ es el predominio para interpretar los valores. El predominio de valores se obtiene resolviendo la ecuación recursiva de dominios que involucra la interpretación de los valores numéricos y funcionales. Interpretamos las constantes numéricas como elementos del predominio discreto $\mathbb{N}$. Bajo la estrategia de evaluación call-by-value, las funciones son aplicadas a valores y no a expresiones posiblemente divergentes, por lo tanto los valores funcionales deben ser interpretados como funciones continuas en $\mathcal{V} \to \mathcal{V}_\perp$. En resumen, diremos que $\mathcal{V}$ es la menor solución para $X \cong \mathsf{F}\,X$ con $\mathsf{F}\,X = \mathbb{N} \oplus (X \to X_\perp)$. Asumimos que el isomorfismo esta dado por $\mathcal{V} \underset{\psi}{\overset{\varphi}{\rightleftarrows}} \mathbb{N} \oplus (\mathcal{V} \to \mathcal{V}_\perp)$ e introducimos las siguientes funciones:

$$\iota_{\mathbf{nat}}(\_) : \mathbb{N} \to \mathcal{V} \qquad \iota_{\mathbf{fun}}(\_) : (\mathcal{V} \to \mathcal{V}_\perp) \to \mathcal{V}$$
$$\iota_{\mathbf{nat}}(\_) = \psi \circ \iota_0 \qquad \iota_{\mathbf{fun}}(\_) = \psi \circ \iota_1$$

Escribimos $\iota_\uparrow(v)$ para el valor promovido $\mathit{Val}\, v$. Notar que el elemento $\perp \in \mathcal{V}_\perp$ es utilizado para representar tanto la no terminación como los errores que son resultados de expresiones mal tipadas, como por ejemplo en $(\mathbf{fun}\ \mathsf{f}\ \mathsf{x} = \mathsf{x}) \oplus \lfloor 42 \rfloor$. Este ejemplo muestra que la interpretación de la ope-

ración binaria $\oplus\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ tiene que ser promovida a una operación binaria $\oplus_{\mathbf{nat}}\colon \mathcal{V} \times \mathcal{V} \to \mathcal{V}_\perp$ que esta definida como

$$v \oplus_{\mathbf{nat}} v' = \begin{cases} \iota_\uparrow(\iota_{\mathbf{nat}}(m \oplus n)) & \text{si } v = \iota_{\mathbf{nat}}(m) \text{ y } v' = \iota_{\mathbf{nat}}(n) \\ \bot & \text{si } v = \iota_{\mathbf{fun}}(f) \text{ o } v' = \iota_{\mathbf{fun}}(g) \end{cases}$$

Además, como permitimos expresiones como operandos, usamos la extensión doblemente estricta de $\oplus_{\mathbf{nat}}$, denotada como $\oplus_{\mathbf{nat}_\perp}\colon \mathcal{V}_\perp \times \mathcal{V}_\perp \to \mathcal{V}_\perp$.

La semántica del lenguaje se define con dos funciones mutuamente recursivas dando significado a los valores y expresiones; para $X \subseteq \mathbb{V}$, los entornos son funciones en $\mathrm{Env}_X = X \to \mathcal{V}$ dando semántica a las variables. La interpretación de cada valor y expresión debe ser una función continua, en la mecanización la definición es mucho más abstracta combinando morfismos en la categoría de dominios, de esta manera nos ahorramos las pruebas de continuidad para cada definición.

Definición 17 (Semántica Extrínseca)**.**

$$\begin{aligned}
[\![_]\!]^{ev}\colon \mathrm{V}(X) \to \mathrm{Env}_X \to \mathcal{V} \qquad & [\![_]\!]^{e\Lambda}\colon \Lambda(X) \to \mathrm{Env}_X \to \mathcal{V}_\perp \\
[\![x]\!]^{ev}\eta = \eta(x) \qquad & [\![v]\!]^{e\Lambda}\,\eta = \iota_\uparrow([\![v]\!]^{ev}\eta) \\
[\![\lfloor n \rfloor]\!]^{ev}\eta = \iota_{\mathit{nat}}(n) \qquad & [\![v\,v']\!]^{e\Lambda}\,\eta = (\lambda f.\, f([\![v']\!]^{ev}\eta))_{\mathit{fun}}([\![v]\!]^{ev}\eta) \\
[\![\mathit{fun}\ f\ x = e]\!]^{ev}\eta = \iota_{\mathit{fun}}(Y_{\mathcal{V}\to\mathcal{V}_\perp} F) \qquad & [\![e \oplus e']\!]^{e\Lambda}\,\eta = ([\![e]\!]^{e\Lambda}\eta) \oplus_{\mathit{nat}_\perp} ([\![e']\!]^{e\Lambda}\eta) \\
\multicolumn{2}{c}{\textit{donde}} \\
\multicolumn{2}{c}{F\colon (\mathcal{V} \to \mathcal{V}_\perp) \to \mathcal{V} \to \mathcal{V}_\perp} \\
\multicolumn{2}{c}{F\ g\ v = [\![e]\!]^{e\Lambda}[\eta \mid f : \iota_{\mathit{fun}}(g) \mid x : v]}
\end{aligned}$$

$$[\![\mathit{let}\ x{=}e\ \mathit{in}\ e']\!]^{e\Lambda}\,\eta = (\lambda v.\, [\![e']\!]^{e\Lambda}[\,\eta \mid x : v\,])_\perp([\![e]\!]^{e\Lambda}\eta)$$

$$[\![\mathit{if}\ e\ \mathit{then}\ e'\ \mathit{else}\ e'']\!]^{e\Lambda}\,\eta = \begin{cases} [\![e']\!]^{e\Lambda}\eta & \mathit{if}\ [\![e]\!]^{e\Lambda}\eta = \iota_\uparrow(\iota_{\mathit{nat}}(0)) \\ [\![e'']\!]^{e\Lambda}\eta & \mathit{if}\ [\![e]\!]^{e\Lambda}\eta = \iota_\uparrow(\iota_{\mathit{nat}}(n+1)) \\ \bot & \mathit{if}\ [\![e]\!]^{e\Lambda}\eta = \bot \end{cases}$$

SEMÁNTICA DENOTACIONAL INTRÍNSECA Se puede argumentar que utilizar al modelo del cálculo lambda sin tipos es demasiado complicado si existe un sistema de tipos que fuerza que solo las expresiones bien tipadas son las evaluadas. A continuación, introducimos un sistema de tipos

para el lenguaje y definimos una semántica solo para expresiones bien tipadas. Este tipo de semántica denotacional es llamado *semántica intrínseca* por Reynolds y evita tener que resolver ecuaciones recursivas de dominio.

Por simplicidad consideremos un solo tipo atómico (**nat**) y tipos funcionales (más adelante, en la sección 4.4 veremos una extensión interesante de este sistema de tipos). Los contextos, como es usual, asignan tipos a variables libres.

Definición 18 (Tipos y Contextos)**.**

$$\theta \in \mathsf{Type} ::= \mathit{nat} \mid \theta \to \theta' \qquad \pi \in \mathsf{Ctx} ::= \emptyset \mid \pi, x : \theta$$

Un juicio de tipado $\pi \vdash^{\wedge} e : \theta$ dice que la expresión e tiene tipo θ bajo el contexto π (analogamente para valores v). Un juicio es válido si uno puede construir una prueba usando las siguiente reglas de tipado.

Definición 19 (Reglas de tipado)**.**

$$\frac{x : \theta \in \pi}{\pi \vdash^{V} x : \theta} \qquad \frac{}{\pi \vdash^{V} \lfloor n \rfloor : \mathit{nat}} \qquad \frac{\pi \vdash^{\wedge} e : \mathit{nat} \quad \pi \vdash^{\wedge} e' : \mathit{nat}}{\pi \vdash^{\wedge} e \oplus e' : \mathit{nat}}$$

$$\frac{\pi \vdash^{\wedge} e : \theta' \quad \pi, x : \theta' \vdash^{\wedge} e' : \theta}{\pi \vdash^{\wedge} \mathit{let}\ x = e\ \mathit{in}\ e' : \theta} \qquad \frac{\pi, x : \theta', f : \theta' \to \theta \vdash^{\wedge} e : \theta}{\pi \vdash^{V} \mathit{fun}\ f\ x = e : \theta' \to \theta}$$

$$\frac{\pi \vdash^{V} v : \theta' \to \theta \quad \pi \vdash^{V} v' : \theta'}{\pi \vdash^{\wedge} v v' : \theta}$$

$$\frac{\pi \vdash^{\wedge} e : \mathit{nat} \quad \pi \vdash^{\wedge} e' : \theta \quad \pi \vdash^{\wedge} e'' : \theta}{\pi \vdash^{\wedge} \mathit{if}\ e\ \mathit{then}\ e'\ \mathit{else}\ e'' : \theta}$$

Es importante remarcar que a priori uno puede tener más de una derivación, o prueba, para el mismo juicio de tipado $\pi \vdash^{\wedge} e : \theta$.

Recursivamente definimos la semántica de los tipos, que determina la semántica de los contextos, definidos como el producto indexado de la semántica de los tipos sobre el dominio del contexto.

Definición 20 (Semántica intrínseca de tipos y contextos)**.**

$$[\![\mathit{nat}]\!]^i = \mathbb{N} \qquad [\![\theta' \to \theta]\!]^i = [\![\theta']\!]^i \to [\![\theta]\!]^i_{\perp}$$

$$[\![\pi]\!] = \prod_{x \in \mathrm{dom}(\pi)} [\![\pi x]\!]^i$$

La semántica intrínseca interpreta solamente las expresiones y valores bien tipados, por lo tanto el significado se asigna a los juicios recursión en su prueba. Escribimos $[\![\pi \vdash^{\Lambda} e : \theta]\!]^{i\Lambda}$ haciendo abuso de notación para $[\![\mathcal{D}]\!]^{i\Lambda}$ donde $\mathcal{D}$ es la derivación cuya conclusión es $\pi \vdash^{\Lambda} e : \theta$. En consecuencia, un enunciado más preciso de coherencia es: $[\![\mathcal{D}]\!]^{i\Lambda} = [\![\mathcal{D}']\!]^{i\Lambda}$ para cualesquiera $\mathcal{D}$ y $\mathcal{D}'$ tales que son pruebas del mismo juicio de tipado $\pi \vdash^{\Lambda} e : \theta$.

Definición 21 (Semántica Intrínseca)**.**

$$
\begin{aligned}
[\![\pi \vdash^{V} v : \theta]\!]^{iv} &: [\![\pi]\!] \to [\![\theta]\!]^{i} \\
[\![\pi \vdash^{V} x : \theta]\!]^{iv}\, \xi &= \xi(x) \\
[\![\pi \vdash^{V} \lfloor n \rfloor : \mathit{nat}]\!]^{iv}\, \xi &= n \\
[\![\pi \vdash^{V} \mathit{fun}\ f\ x = e : \theta' \to \theta]\!]^{iv}\, \xi &= Y_{[\![\theta']\!] \to [\![\theta]\!]_{\perp}} F \quad \textit{donde} \\
&\qquad F : ([\![\theta']\!] \to [\![\theta]\!]_{\perp}) \to [\![\theta']\!] \to [\![\theta]\!]_{\perp} \\
&\qquad F\ g\ v = [\![\pi \vdash^{\Lambda} e : \theta]\!]^{i\Lambda} [\xi \mid f : g \mid x : v]
\end{aligned}
$$

$$
\begin{aligned}
[\![\pi \vdash^{\Lambda} e : \theta]\!]^{i\Lambda} &: [\![\pi]\!] \to [\![\theta]\!]^{i}_{\perp} \\
[\![\pi \vdash^{\Lambda} v : \theta]\!]^{i\Lambda}\ \xi &= \iota_{\uparrow}([\![\pi \vdash^{V} v : \theta]\!]^{iv} \xi) \\
[\![\pi \vdash^{\Lambda} v v' : \theta]\!]^{i\Lambda}\ \xi &= ([\![\pi \vdash^{V} v : \theta' \to \theta]\!]^{iv} \xi)([\![\pi \vdash^{V} v' : \theta']\!]^{iv} \xi) \\
[\![\pi \vdash^{\Lambda} e \oplus e' : \mathit{nat}]\!]^{i\Lambda}\ \xi &= ([\![\pi \vdash^{\Lambda} e : \mathit{nat}]\!]^{i\Lambda} \xi) \oplus_{\perp} ([\![\pi \vdash^{\Lambda} e' : \mathit{nat}]\!]^{i\Lambda} \xi) \\
[\![\pi \vdash^{\Lambda} \mathit{let}\ x{=}e\ \mathit{in}\ e' : \theta]\!]^{i\Lambda}\ \xi &= (\lambda v.\ [\![\pi, x : \theta' \vdash^{\Lambda} e' : \theta]\!]^{i\Lambda} [\ \xi \mid x : v\ ])_{\perp} \\
&\qquad ([\![\pi \vdash^{\Lambda} e : \theta']\!]^{i\Lambda} \xi) \\
[\![\pi \vdash^{\Lambda} \mathit{if}\ e\ \mathit{then}\ e'\ \mathit{else}\ e'' : \theta]\!]^{i\Lambda} \xi &= \begin{cases} [\![\pi \vdash^{\Lambda} e' : \theta]\!]^{i\Lambda} \xi & \textit{si } [\![\pi \vdash^{\Lambda} e : \mathit{nat}]\!]^{i\Lambda} \xi = \iota_{\uparrow}(0) \\ [\![\pi \vdash^{\Lambda} e'' : \theta]\!]^{i\Lambda} \xi & \textit{si } [\![\pi \vdash^{\Lambda} e : \mathit{nat}]\!]^{i\Lambda} \xi = \iota_{\uparrow}(n+1) \\ \perp & \textit{si } [\![\pi \vdash^{\Lambda} e : \mathit{nat}]\!]^{i\Lambda} \xi = \perp \end{cases}
\end{aligned}
$$

4.2 EL SIGNIFICADO DE LOS TIPOS

Es razonable esperar que ambas semánticas denotacionales *coincidan* de alguna manera; además, uno bien puede cuestionar la utilidad de tener ambas semánticas. En esta sección aplicamos la técnica de Reynolds para probar la equivalencia de ambas semánticas; un corolario del método de prueba es el resultado de coherencia para la semántica intrínseca. Creemos

que esto da motivo suficiente para definir la semántica extrínseca de un lenguaje para el cual ya existe una semántica intrínseca.

Podemos construir un modelo del sistema de tipos encima de la semántica extrínseca definiendo subconjuntos de $\mathcal{V}$ y $\mathcal{V}_\perp$ para cada tipo y probar que la denotación de valores y expresiones bien tipadas pertenecen a estos subconjuntos. Esta idea está ya presente en el trabajo de Scott [43], donde la denotación de los tipos son retracciones del dominio $\mathcal{V}_\perp$. Definimos por inducción en los tipos los subconjuntos $[\![\theta]\!]^e$:

$$\begin{aligned}[\![\mathbf{nat}]\!]^e &= \{v \in \mathcal{V} \mid v = \iota_{\mathbf{nat}}(n) \text{ para algún } n\} \\ [\![\theta' \to \theta]\!]^e &= \{v \in \mathcal{V} \mid v = \iota_{\mathbf{fun}}(f) \text{ para alguna } f, \\ &\qquad \text{tal que para toda } w \in [\![\theta']\!]^e \text{ vale } f\, w \in [\![\theta]\!]^e_\perp\}\end{aligned}$$

Uno puede extender esta definición para contextos y probar por inducción en la derivación del juicio de tipado la corrección del sistema de tipos: si $\pi \vdash^\wedge e : \theta$ y $\eta \in [\![\pi]\!]^e$, entonces $[\![e]\!]^{e\wedge}\eta \in [\![\theta]\!]^e_\perp$.

Reynolds [38, 39] definió "embedding-projection pairs" entre la semántica intrínseca de los tipos y el dominio universal $\mathcal{V}$ por recursión en los tipos.

Definición 22 (Embedding-projections pairs)**.** *Definimos simultáneamente las funciones continuas* $\downarrow_\theta : [\![\theta]\!]^i \to \mathcal{V}$ *y* $\uparrow_\theta : \mathcal{V} \to [\![\theta]\!]^i_\perp$.

$$\begin{aligned}\downarrow_{nat} n &= \iota_{nat}(n) & \uparrow_{nat} &= [\iota_\uparrow, f \mapsto \perp] \\ \downarrow_{\theta\to\theta'} f &= \iota_{fun}((\downarrow_{\theta'})_\perp \circ f_\perp \circ \uparrow_\theta) & \uparrow_{\theta\to\theta'} &= [n \mapsto \perp, g \mapsto (\uparrow_{\theta'})_\perp \circ g \circ \downarrow_\theta]\end{aligned}$$

Definimos a las proyecciones de $\mathcal{V}$ a $[\![\theta]\!]^i_\perp$ con el único morfismo mediador definido por co-productos. Notar que la proyección retorna $\perp$ en el caso de un "error" de tipos.

En lugar de probar directamente que estas funciones definen un "ep-pair", utilizamos la propuesta de Reynolds y obtenemos este resultado a partir de la definición de las relaciones lógicas entre la semántica intrínseca y la extrínseca. Es interesante remarcar que la corrección del sistema de tipos con respecto a la semántica extrínseca es también un corolario de estas relaciones.

EXTENSIÓN ESTRICTA DE LAS RELACIONES Reynolds desarrolla su técnica considerando un lenguaje call-by-name y utilizando lógica clásica para razonar acerca de la promoción de dominios. Cuando consideramos la estrategia de evaluación call-by-value necesitamos promover una relación

$R \subseteq P \times P'$ entre predominios a la de los predominios promovidos, es decir $R_\perp \subseteq P_\perp \times P'_\perp$ sea la menor relación bi-estricta que contiene a R; clásicamente es simple definir tal extensión:

$$\perp R_\perp \perp \quad \text{y} \quad \iota_\uparrow(v)\, R_\perp\, \iota_\uparrow(v') \text{ sii } v\, R\, v'$$

Esta definición no puede ser formalizada; recordemos que la promoción de un predominio en la librería está construida utilizando listas infinitas, luego es imposible decidir si un elemento es $\perp$ o no. Sin embargo, es posible dar una definición co-inductiva para $R_\perp$ y probar luego que captura la noción de extensión estricta de R; utilizamos doble línea guionada para indicar que las reglas son co-inductivas.

$$\frac{d\, R_\perp\, d'}{\mathsf{Eps}\ d\, R_\perp\, \mathsf{Eps}\ d'} \qquad \frac{\exists\, n, m \quad d \upharpoonleft n = \mathsf{Val}\, v \quad d' \upharpoonleft m = \mathsf{Val}\, v' \quad v R v'}{d\, R_\perp\, d'}$$

Los siguientes lemas prueban que $R_\perp$ es, efectivamente, la menor extensión bi-estricta de R.

Lema 20.

1. $\perp R_\perp \perp$
2. *Si* $v R v'$, *entonces* $(\mathsf{Val}\, v)\, R_\perp\, (\mathsf{Val}\, v')$.

La primera parte de este lema es equivalente a probar $(\mathsf{Eps}\ \perp)\, R_\perp\, (\mathsf{Eps}\ \perp)$, lo cual es directo por co-inducción, la segunda parte es directa también utilizando la definición $\upharpoonleft$. Además, enunciamos y probamos un lema de inversión para los elementos relacionados bajo la extensión estricta de R.

Lema 21 (Lema de Inversión para $R_\perp$)**.**

1. *Si* $(\mathsf{Val}\, v)\, R_\perp\, d'$, *entonces existe* v', *tal que* $d' = \mathsf{Val}\, v'$ *y* $v R v'$.
2. *Si* $d\, R_\perp\, (\mathsf{Val}\, v')$, *entonces existe* v, *tal que* $d = \mathsf{Val}\, v$ *y* $v R v'$.
3. *Si* $d\, R_\perp \perp$, *entonces* $d = \perp$
4. *Si* $\perp\, R_\perp\, d'$, *entonces* $d' = \perp$

Un corolario inmediato de este lema es que no existen pares de la forma $(\perp, \mathsf{Val}\, v')$ o $(\mathsf{Val}\, v, \perp)$ en $R_\perp$. En particular si $R = P \times Q$, entonces $R_\perp$ es el producto aplastado de los predominios promovidos: $R_\perp = P_\perp \otimes Q_\perp$; esta ultima afirmación depende de demostrar que la extensión estricta de R es cerrada por supremo de cadenas para cualquier R que también lo sea.

Lema 22.

1. $\mathsf{R}_\perp$ *es estricta.*
2. *Si* R *es cerrada por cadenas, entonces* $\mathsf{R}_\perp$ *es también cerrada por supremos de cadenas.*

Teorema de Bracketing

Las relaciones lógicas están definidas por inducción en los tipos y permiten probar propiedades de las expresiones bien tipadas por inducción en la derivación del juicio de tipado, obteniendo de esta manera una hipótesis más fuerte que si solo procediéramos por inducción en la estructura de las expresiones. La relación lógica para valores, $\delta[\theta] \subseteq [\![\theta]\!]^i \times \mathcal{V}$ captura la noción de que elementos del dominio $\mathcal{V}$ pertenecen a la retracción correspondiente del dominio; en el caso del tipo funcional, la función correspondiente preserva esta relación. La relación lógica para las expresiones $\rho[\theta] \subseteq [\![\theta]\!]^i_\perp \times \mathcal{V}_\perp$, se define como la extensión estricta de $\delta[\theta]$; $\rho[\theta] = \delta[\theta]_\perp$. Siendo más precisos, probaremos que los significados dados a una expresión bien tipada por cada semántica están lógicamente relacionados.

Definición 23 (Relación Lógica entre la semántica intrínseca y extrínseca)**.**

$$\frac{}{(n, \iota_{\mathit{nat}}(n)) \in \delta[\mathit{nat}]} \qquad \frac{(f\ x, g\ y) \in \rho[\theta], \textit{ para todo } (x, y) \in \delta[\theta']}{(f, \iota_{\mathit{fun}}(g)) \in \delta[\theta' \to \theta]}$$

Probamos simultáneamente por inducción en θ y utilizando el lema 22 que $\delta[\theta]$ y $\rho[\theta]$ son cerradas por supremos de cadenas.

Lema 23. *Para todo* $\theta \in \mathsf{Type}$*,* $\delta[\theta]$ *y* $\rho[\theta]$ *son cerradas por supremos de cadenas.*

El siguiente teorema relaciona los "embedding-projection pairs" con las relaciones lógicas y expone dos resultados importantes: (I) cualquier valor denotacional intrínseco está relacionado con uno extrínseco que es el resultado del "embedding"; y (II) cualquier valor denotacional extrínseco está relacionado con uno intrínseco que se encuentra en la imagen de la proyección.

Teorema 2 (Bracketing)**.**

1. *Si* $v \in [\![\theta]\!]^i$, *entonces* $(v, \downarrow_\theta v) \in \delta[\theta]$.
2. *Si* $(v, v') \in \delta[\theta]$, *entonces* $\iota_\uparrow(v) = \uparrow_\theta v'$.
3. *Si* $d \in [\![\theta]\!]^i_\perp$, *entonces* $(d, (\iota_\uparrow \circ \downarrow_\theta)_\perp d) \in \rho[\theta]$.
4. *Si* $(d, d') \in \rho[\theta]$, *entonces* $d = (\uparrow_\theta)_\perp d'$.

Un corolario inmediato de esto es que $\uparrow$ y $\downarrow$ satisfacen la condición de "embedding-projection".

Corolario 1 (Embedding-projection pairs)**.** $\uparrow_\theta \circ \downarrow_\theta = \iota_\uparrow$.

Demostración. Supongamos $v \in [\![\theta]\!]^i$, queremos probar que $\uparrow_\theta (\downarrow_\theta v) = \iota_\uparrow(v)$. Utilizando la primera parte de bracketing 2 tenemos que $(v, \downarrow_\theta v) \in \delta[\theta]$, entonces por la segunda parte de bracketing 2 podemos concluir $\uparrow_\theta (\downarrow_\theta v) = \iota_\uparrow(v)$. ■

La otra mitad de la condición de ep-pairs es $(\iota_\uparrow \circ \downarrow_\theta)_\perp \circ \uparrow_\theta \leqslant \iota_\uparrow$ que la probamos por inducción sobre θ. Para expresar la propiedad fundamental de las relaciones lógicas, que expresa que el significado de expresiones bien tipadas está relacionado, necesitamos extender las relaciones lógicas para entornos tipados y no tipados; luego introducimos la relación lógica para valores y expresiones no cerradas.

Definición 24 (Relaciones lógicas para entornos y relaciones lógicas abiertas)**.** *Sea* π *un contexto tal que* $\mathrm{dom}(\pi) = X$. *Definimos* $\gamma[\pi] \subseteq [\![\pi]\!] \times \mathrm{Env}_X$, $\Delta[\pi, \theta] \subseteq ([\![\pi]\!] \to [\![\theta]\!]^i) \times (\mathrm{Env}_X \to \mathcal{V})$, *y* $P[\pi, \theta] \subseteq ([\![\pi]\!] \to [\![\theta]\!]^i_\perp) \times (\mathrm{Env}_X \to \mathcal{V}_\perp)$.

1. $(\xi, \eta) \in \gamma[\pi]$ *sii* $(\xi\, x, \eta\, x) \in \delta[\pi x], \forall x \in \mathrm{dom}(\pi)$.
2. $(f, g) \in \Delta[\pi, \theta]$ *sii* $(f\ \xi, g\ \eta) \in \delta[\theta], \forall (\xi, \eta) \in \gamma[\pi]$.
3. $(f, g) \in P[\pi, \theta]$ *sii* $(f\ \xi, g\ \eta) \in \rho[\theta], \forall (\xi, \eta) \in \gamma[\pi]$.

Finalmente podemos enunciar el teorema fundamental de las relaciones lógicas que relaciona el significado intrínseca y extrínseco de valores y expresiones bien tipadas. La prueba procede por inducción mutua en la derivación del juicio de tipado para valores y expresiones.

Teorema 3 (Propiedad Fundamental de las Relaciones Lógicas)**.**

1. *Si* $\pi \vdash^{V} v : \theta$, *entonces* $([\![\pi \vdash^{V} v : \theta]\!]^{iv}, [\![v]\!]^{ev}) \in \Delta[\pi, \theta]$
2. *Si* $\pi \vdash^{\Lambda} e : \theta$, *entonces* $([\![\pi \vdash^{\Lambda} e : \theta]\!]^{i\Lambda}, [\![e]\!]^{e\Lambda}) \in P[\pi, \theta]$

Claramente podemos extender los embedding y las proyecciones punto a punto. Definimos $\downarrow$ como el embedding de entornos tipados en entornos no tipados. Es directo extender el teorema 2 para entornos.

Corolario 2. *Si* $\xi \in [\![\pi]\!]$, *entonces* $(\xi, \downarrow_{\pi} \xi) \in \gamma[\pi]$.

Finalmente, podemos combinar todos los resultados previos y probar que la semántica denotacional intrínseca coincide con la semántica denotacional extrínseca.

Teorema 4. $[\![\pi \vdash^{\Lambda} e : \theta]\!]^{i\Lambda} = \uparrow_{\theta\perp} \circ [\![e]\!]^{e\Lambda} \circ \downarrow_{\pi}$.

Demostración. Supongamos $\xi \in [\![\pi]\!]$, entonces tenemos $(\xi, (\downarrow_{\pi} \xi)) \in \gamma[\pi]$, por el corolario 2 y el teorema 3 deducimos

$$([\![\pi \vdash^{\Lambda} e : \theta]\!]^{i\Lambda} \xi, [\![e]\!]^{e\Lambda} (\downarrow_{\pi} \xi)) \in \rho[\theta]$$

luego concluimos, utilizando el ultimo punto del teorema 2,

$$[\![\pi \vdash^{\Lambda} e : \theta]\!]^{i\Lambda} \xi = (\uparrow_{\theta})_{\perp} ([\![e]\!]^{e\Lambda} (\downarrow_{\pi} \xi)) \qquad \blacksquare$$

El teorema previo nos conduce a una prueba directa de la coherencia de la semántica denotacional intrínseca.

Corolario 3 (Coherencia)**.** *Sean* $\mathcal{D}$ *y* $\mathcal{D}'$ *diferentes derivaciones del juicio de tipado* $\pi \vdash^{\Lambda} e : \theta$, *entonces* $[\![\mathcal{D}]\!]^{i\Lambda} = [\![\mathcal{D}']\!]^{i\Lambda}$.

4.3 ADECUACIÓN

En esta sección probamos la adecuación de la semántica operacional con respecto a la semántica denotacional extrínseca, que hasta donde sabemos no ha sido considerado antes. Como mencionamos en la introducción, adecuación esta compuesta de dos partes: (I) si la semántica denotacional de una expresión bien tipada e es igual a algún valor numérico n, entonces e evalúa a $\lfloor n \rfloor$ – recordemos que esto significa $(\mathsf{Id}, e) \longmapsto^{*} (\mathsf{Id}, \lfloor n \rfloor)$; además (II) si la semántica denotacional de e es $\perp$, entonces la reducción que comienza con (Id, e) diverge – $(\mathsf{Id}, e) \longmapsto^{*} \infty$.

Uno podría ingenuamente pensar que la prueba de adecuación puede proceder por inducción en la estructura de las expresiones, pero el caso de la aplicación demuestra lo contrario. Utilizando relaciones lógicas podemos obtener hipótesis inductivas lo suficientemente fuertes para probar por inducción en la derivación del juicio de tipado un resultado más general del cual obtenemos adecuación como un corolario.

Adecuación para expresiones divergentes

Formalmente probaremos que, dada una expresión cerrada $e \in \Lambda$ tal que para el contexto vacío existe una derivación del juicio de tipado $\vdash^{\Lambda} e : \theta$, si $[\![e]\!]^{e\Lambda}() = \bot$, entonces $(\mathsf{Id}, e) \longmapsto^{*} \infty$. La estrategia de prueba consiste en definir dos relaciones lógicas indexadas, que cumplan con la propiedad de step-indexed, de manera que relacionen valores y expresiones sintácticas con elementos de los modelos semánticos $\mathcal{V}$ y $\mathcal{V}_{\bot}$, respectivamente. Esta última captura la noción de que una expresión sintáctica puede realizar alguna cantidad de transiciones para ser una mejor aproximación de un elemento semántico; probamos que si e aproxima a $\bot$ para cualquier índice, entonces $(\mathsf{Id}, e) \longmapsto^{*} \infty$. La propiedad fundamental de las relaciones lógicas dirá que cada expresión bien tipada aproxima a su semántica para todo índice.

Como mencionamos en la sección 2.2, la prueba de adecuación para expresiones divergentes utiliza una familia indexada de relaciones; en nuestro caso esa familia surgirá de una familia de observaciones $\perp\!\!\!\perp_i$. De hecho, exigiremos una condición más sobre el conjunto $\perp\!\!\!\perp_i$ para considerarla una observación: debe ser cerrada por anti-ejecución:

$$(E, e) \in \perp\!\!\!\perp_i \text{ y } (E', e') \longmapsto (E, e) \text{ implica } (E', e') \in \perp\!\!\!\perp_{i+1}$$

Concretamente nosotros definiremos la relación $\perp\!\!\!\perp_i \subseteq \Lambda^{\star} \times \Lambda$ si y solo si la configuración puede realizar al menos i pasos de reducción y probamos que si la expresión bien tipada e aproxima a su semántica, entonces $(\mathsf{Id}, e) \in \longmapsto_n$ para cualquier n. Es fácil ver que $\perp\!\!\!\perp_i$ satisface esta condición.

Dada una observación definimos una relación step-indexed sobre los conjuntos indexados de valores, expresiones y frame stacks utilizando la definición 3 del capítulo 2; denotados $iV = \mathbb{N} \times V$, $i\Lambda = \mathbb{N} \times \Lambda$ e $i\Lambda^{\star} = \mathbb{N} \times \Lambda^{\star}$, respectivamente. Luego cualquier observación $\perp\!\!\!\perp_i$ nos induce una relación step-indexed $\hat{\perp\!\!\!\perp} \subseteq i\Lambda^{\star} \times i\Lambda$

$$((i, E), (j, e)) \in \hat{\perp\!\!\!\perp} \text{ sii } (E, e) \in \perp\!\!\!\perp_{\min(j,i)}$$

Como vimos en la sección 2.2, podemos usar biortogonalidad sobre esta observación $\hat{\mathbb{1}}$ sobre configuraciones indexadas; los operadores ortogonales asociados, que nos inducen el operador de clausura $\_^{\perp\top} : \mathcal{P}(\mathfrak{i}V) \to \mathcal{P}(\mathfrak{i}\Lambda)$, están dados por:

Definición 25.

$$\begin{aligned}
&_^{\perp} : \mathcal{P}(\mathfrak{i}V) \to \mathcal{P}(\mathfrak{i}\Lambda^{\star}) \\
&X^{\perp} = \{\, (\mathfrak{j}, E) \mid \textit{para todo } (\mathfrak{i}, \mathfrak{v}) \in X,\ (\mathfrak{j}, E)\hat{R}(\mathfrak{i}, \mathfrak{v})\} \\
&_^{\top} : \mathcal{P}(\mathfrak{i}\Lambda^{\star}) \to \mathcal{P}(\mathfrak{i}\Lambda) \\
&Y^{\top} = \{\, (\mathfrak{j}, e) \mid \textit{para todo } (\mathfrak{i}, E) \in X,\ (\mathfrak{j}, E)\hat{R}(\mathfrak{i}, e)\}\ .
\end{aligned}$$

Para definir las relaciones lógicas $\lhd^{\theta}_{\mathfrak{i}}$ y $\blacktriangleleft^{\theta}_{\mathfrak{i}}$ que relacionen valores y expresiones con elementos de $\mathcal{V}$ y $\mathcal{V}_{\perp}$ respectivamente, donde esta ultima se define en términos de la primera introducimos la extensión mediante la promoción de cualquier relación $\bowtie \subseteq V \times \mathcal{V}$. Denotamos a esta extensión como $\Omega^{\bowtie} \subseteq V \times \mathcal{V}_{\perp}$.

Definición 26 (Extensión mediante promover)**.**

$$\mathfrak{v}\ \Omega^{\bowtie}\ d \textit{ si existe } v \in \mathcal{V} \textit{ tal que } d = \iota_{\uparrow}(v) \textit{ y } \mathfrak{v} \bowtie v$$

La relación lógica para expresiones se obtiene como resultado de utilizar el operador relacional sobre una familia indexada de relaciones sobre los tipos.

Definición 27 (Aproximación Operacional)**.** *Sea* $\lhd^{\theta}_{\mathfrak{i}} \subseteq V \times \mathcal{V}$*, definimos* $\blacktriangleleft^{\theta}_{\mathfrak{i}} \subseteq \Lambda \times \mathcal{V}_{\perp}$ *como*

$$e \blacktriangleleft^{\theta}_{0} \perp \quad y \quad e \blacktriangleleft^{\theta}_{n} d \textit{ sii } (n, e) \in\ (\Omega^{\lhd^{\theta}_{n}} d)^{\perp\top}$$

Ahora definimos la relación lógica $\lhd^{\theta}_{\mathfrak{i}} \subseteq V \times \mathcal{V}$ por inducción en los tipos. Para el tipo básico **nat** cada constante $\lfloor m \rfloor$ aproxima al natural m para cualquier índice; en el caso del tipo funcional está definida inductivamente, esperando que la aplicación de argumentos relacionados (en un índice menor) preserva la relación en el tipo de llegada (esta es la hipótesis inductiva más fuerte que mencionábamos antes):

Definición 28 (Aproximación Operacional para valores)**.**

$$\frac{}{\lfloor m \rfloor \lhd^{\textit{nat}}_{n} \iota_{\textit{nat}}(m)}$$

$$\frac{e[x/\mathfrak{v}, f/(\textbf{\textit{fun}}\ f\ x = e)] \blacktriangleleft^{\theta}_{m} f\mathfrak{v}, \quad \textit{para todo } m < n \textit{ y } \mathfrak{v} \lhd^{\theta'}_{m} v}{\textbf{\textit{fun}}\ f\ x = e \lhd^{\theta' \to \theta}_{n} \iota_{\textit{fun}}(f)}$$

Estas relaciones lógicas $\_ \lhd^{\theta}_{i} \_$ y $\_ \blacktriangleleft^{\theta}_{i} \_$ son step-indexed.

Lema 24 (Aproximaciones operacionales son Step-Indexed)**.**

1. *Si* $v \lhd^{\theta}_{i} v$ *y* $j \leqslant i$, *entonces* $v \lhd^{\theta}_{j} v$
2. *Si* $e \blacktriangleleft^{\theta}_{i} d$ *y* $j \leqslant i$, *entonces* $e \blacktriangleleft^{\theta}_{j} d$

Al igual que en la sección anterior, extendemos esta relación punto a punto para obtener una relación entre substituciones y entornos no tipados, la cual claramente es step-indexed.

Definición 29 (Aproximación opercional para contextos)**.**

$$\sigma \lhd^{\pi}_{i} \eta \quad \textit{sii} \quad \forall x \in \mathrm{dom}(\pi),\ \sigma\, x \lhd^{\pi x}_{i} \eta\, x$$

Utilizando esta relación definimos la versión no cerrada de las relaciones lógicas previas:

Definición 30 (Aproximación para expresiones no cerradas)**.**

$$\pi \vdash v \lhd^{\theta} v \quad \textit{sii} \quad v\,\sigma \lhd^{\theta}_{i} v\,\eta,\ \textit{para todo}\ \sigma \lhd^{\pi}_{i} \eta$$

$$\pi \vdash e \blacktriangleleft^{\theta} d \quad \textit{sii} \quad e\,\sigma \blacktriangleleft^{\theta}_{i} d\,\eta,\ \textit{para todo}\ \sigma \lhd^{\pi}_{i} \eta$$

La propiedad fundamental de las relaciones lógicas expresa que cualquier expresión bien tipada está relacionada con su semántica extrínseca. Este resultado lo probamos por inducción en la derivación del juicio de tipado; es interesante remarcar que el caso problemático de la aplicación se prueba directamente gracias a la hipótesis inductiva más fuerte dada por las relaciones lógicas.

Teorema 5 (Propiedad Fundamental de las Relaciones Lógicas)**.**

1. *Si* $\pi \vdash^{V} v : \theta$, *entonces* $\pi \vdash v \lhd^{\theta} [\![v]\!]^{ev}$
2. *Si* $\pi \vdash^{\Lambda} e : \theta$, *entonces* $\pi \vdash e \blacktriangleleft^{\theta} [\![e]\!]^{e\Lambda}$

Para completar la prueba de adecuación para expresiones divergentes notemos que cualquier frame stack indexado (n, E) es un test del conjunto vacío de valores que aproxima a $\bot$; además, si cualquier expresión e aproxima a $\bot$ en cualquier índice, entonces (Id, e) diverge.

Lema 25.

1. $(n, E) \in (\Omega^{\triangleleft^{\theta}_{n}}(\bot))^{\bot}$,
2. *Si* $e \blacktriangleleft^{\theta}_{n} \bot$ *para todo* n, *entonces* $(Id, e) \longmapsto^{*} \infty$.

Obtenemos adecuación como un corolario de la propiedad fundamental y del lema previo.

Teorema 6 (Adecuación para expresiones divergentes)**.** *Si* $\vdash^{\Lambda} e : \theta$ *y* $[\![e]\!]^{e\Lambda}() = \bot$, *entonces* $(Id, e) \longmapsto^{*} \infty$.

Demostración. Tenemos $[\![e]\!]^{e\Lambda}() = \bot$, y trivialmente $[] \triangleleft^{\emptyset}_{n} ()$ para todo n. Luego por el teorema 5 tenemos que $e \blacktriangleleft^{\theta}_{n} [\![e]\!]^{e\Lambda}()$, es decir $e \blacktriangleleft^{\theta}_{n} \bot$. Por lo tanto, podemos concluir que $(Id, e) \longmapsto^{*} \infty$, utilizando el lema 25. ■

Adecuación para expresiones convergentes

En esta parte probaremos que para aquellas expresiones cuya denotación intrínseca es un número natural, entonces evaluamos operacionalmente a la representación de ese natural. Además, utilizando los resultados de la sección 4.2 obtenemos la prueba de adecuación con respecto a la semántica extrínseca. Notar que este resultado solo es valido para expresiones cuyo tipo atómico sea **nat**, ya que solo podemos observar el comportamiento de valores de alto-orden aplicándolos. En contraste con la parte divergente, no hay necesidad de utilizar step-indexing.

Asumimos que nuestro conjunto de *observaciones*, es decir un subconjunto de configuraciones, es cerrado por anti-ejecución:

$$\text{si } (E, e) \in \perp\!\!\!\perp \text{ y } (E', e') \longmapsto^{*} (E, e) \text{ entonces } (E', e') \in \perp\!\!\!\perp \ .$$

los operadores quedan definidos como:

Definición 31.

$$\begin{aligned}
&_^{\bot} : \mathcal{P}(V) \to \mathcal{P}(\Lambda^{\star}) \\
&X^{\bot} = \{ E \mid \textit{para todo } v \in X, (E, v) \in \perp\!\!\!\perp \} \\
&_^{\top} : \mathcal{P}(\Lambda^{\star}) \to \mathcal{P}(\Lambda) \\
&Y^{\top} = \{ e \mid \textit{para todo } E \in X, (E, v) \in \perp\!\!\!\perp \} \ .
\end{aligned}$$

La definición de las relaciones lógicas está parametrizada en el conjunto de observaciones. Como antes, utilizamos biortogonalidad para extender la relación entre valores sintácticos y denotacionales, $\_ \rhd^{\theta} \_ \subseteq V \times [\![\theta]\!]^i$, a una relación $\_ \blacktriangleright^{\theta} \_ \subseteq \Lambda \times [\![\theta]\!]^i_{\perp}$.

Definición 32 (Aproximación denotacional)**.**

$$e \blacktriangleright^{\theta} d \textit{ sii } e \in (\rhd^{\theta} v)^{\perp\top}, \textit{ para todo } v \in [\![\theta]\!]^i \textit{ tal que } d = \iota_{\uparrow}(v) \ .$$

La relación lógica para valores $\_ \rhd^{\theta} \_ \subseteq V \times [\![\theta]\!]^i$ se define por inducción en los tipos.

Definición 33 (Aproximación denotacional para valores)**.**

$$\frac{}{\lfloor m \rfloor \rhd^{\textit{nat}} m}$$

$$\frac{e[x/v, f/(\textbf{\textit{fun}}\ f\ x = e)] \blacktriangleright^{\theta} fv, \quad \textit{para todo } v \rhd^{\theta'} v}{\textbf{\textit{fun}}\ f\ x = e \rhd^{\theta' \to \theta} f}$$

Definimos a $\_ \rhd^{\pi} \_$ como la extensión análoga para el caso de la aproximación operacional, donde relacionamos substituciones con entornos, esta vez, tipados. La definición de las relaciones lógicas no-cerradas se obtiene cerrando las expresiones utilizando la substitución.

Definición 34 (Relaciones lógicas no-cerradas)**.**

$$\pi \vdash v \rhd^{\theta} v \quad \textit{sii} \quad v\sigma \rhd^{\theta} v\, \xi, \textit{ para todo } \sigma \rhd^{\pi} \xi$$

$$\pi \vdash e \blacktriangleright^{\theta} d \quad \textit{sii} \quad e\sigma \blacktriangleright^{\theta} d\, \xi, \textit{ para todo } \sigma \rhd^{\pi} \xi$$

La propiedad fundamental de las relaciones lógicas dirá que la denotación intrínseca de un valor aproxima a su valor sintáctico. Si nos enfocamos en el caso del valor funcional, **fun** $f\ x = e$, tendremos que probar que ese valor es aproximado por el supremo de alguna cadena; sin embargo, las relaciones lógicas $(v \rhd^{\theta} \_)$ y $(e \blacktriangleright^{\theta} \_)$ que hemos definido no son necesariamente cerradas por supremos de cadenas. La solución es tomar la clausura de Scott para estas relaciones. Utilizamos esta nuevas relaciones para expresar la propiedad fundamental.

Definición 35 (Aproximación denotacional clausurada)**.**

$$\pi \vdash v \overline{\rhd}^{\theta} v \quad \textit{sii} \quad v \in \mathrm{IC}(\pi \vdash v \rhd^{\theta} \_)$$

$$\pi \vdash e \overline{\blacktriangleright}^{\theta} d \quad \textit{sii} \quad e \in \mathrm{IC}(\pi \vdash e \blacktriangleright^{\theta} \_)$$

Teorema 7 (Propiedad Fundamental de las Relaciones Lógicas)**.**

1. *Si* $\pi \vdash^{V} v : \theta$, *entonces* $\pi \vdash v \overline{\rhd}^{\theta} [\![\pi \vdash^{V} v : \theta]\!]^{iv}$.
2. *Si* $\pi \vdash^{\wedge} v : \theta$, *entonces* $\pi \vdash e \overline{\blacktriangleright}^{\theta} [\![\pi \vdash^{\wedge} e : \theta]\!]^{i\wedge}$.

Notar que podemos probar de manera directa que la substitución vacía es aproximada por el entorno vacío.

Lema 26. $[] \rhd^{\emptyset} ()$

Para deducir el resultado final de adecuación necesitamos fijar una observación considerando un $n \in \mathbb{N}$; definimos el conjunto de configuraciones que evalúan a la configuración con frame stack vacío y valor $\lfloor n \rfloor$, $\perp\!\!\!\perp = \{(E, e) \mid (E, e) \longmapsto^{*} (\mathsf{Id}, \lfloor n \rfloor)\}$ y probamos, fácilmente, que es cerrado por anti-ejecución. Luego cualquier expresión e que es aproximada por n, evalúa a $\lfloor n \rfloor$. Notar que no tenemos una observación que nos permita demostrar el resultado final de adecuación para cualquier n, sino que para cada n definimos una observación distinta y obtenemos el resultado de adecuación correspondiente.

Lema 27. *Si* $e \blacktriangleright^{\theta} \iota_{\uparrow}(n)$, *entonces* $(\mathsf{Id}, e) \longmapsto^{*} (\mathsf{Id}, \lfloor n \rfloor)$

La prueba de este lema es inmediatamente ya que Id es un test para $\{\lfloor n \rfloor\}$ y por la definición del conjunto de observaciones tenemos que las configuraciones convergen exactamente a $(\mathsf{Id}, \lfloor n \rfloor)$.

Teorema 8 (Adecuación para expresiones convergentes)**.**
Si $[\![\vdash^{\wedge} e : \textbf{\textit{int}}]\!]^{i\wedge}[] = \iota_{\uparrow}(n)$, *entonces* $(\mathsf{Id}, e) \longmapsto^{*} (\mathsf{Id}, \lfloor n \rfloor)$.

Demostración. Sea $[\![\vdash^{\wedge} e : \mathbf{int}]\!]^{i\wedge}[] = \iota_{\uparrow}(n)$, luego por el teorema 7 tenemos $[] \vdash e \overline{\blacktriangleright}^{\mathbf{int}} \iota_{\uparrow}(n)$; esto significa que $\iota_{\uparrow}(n) \in \mathrm{IC}(e \blacktriangleright^{\mathbf{int}} \_)$. Ahora como $\mathbb{N}_{\perp}$ es un dominio con orden discreto podemos obtener $e \blacktriangleright^{\mathbf{int}} \iota_{\uparrow}(n)$. Por lo tanto, concluimos $(\mathsf{Id}, e) \longmapsto^{*} (\mathsf{Id}, \lfloor n \rfloor)$, utilizando el lema 27. ■

Además, podemos obtener el resultado de adecuación con respecto a la semántica extrínseca.

Teorema 9 (Adecuación Extrínseca)**.**
Si $\vdash^{\Lambda} e : \textbf{\textit{int}}$ *y* $[\![e]\!]^{e\Lambda}() = \iota_{\textbf{\textit{nat}}}(n)$*, entonces* $(\mathsf{Id}, e) \longmapsto^{*} (\mathsf{Id}, \lfloor n \rfloor)$.

Demostración. Si $\vdash^{\Lambda} e : \textbf{int}$ y $[\![e]\!]^{e\Lambda}() = \iota_{\textbf{nat}}(n)$, entonces por el teorema. 4 obtenemos que $[\![\vdash^{\Lambda} e : \textbf{int}]\!]^{i\Lambda}[] = n$; y podemos concluir utilizando el teorema 8. ■

La adecuación para tipos de alto-orden se reduce a la adecuación para tipos básicos: una expresión representa adecuadamente una función si cuando aplicamos todos sus argumentos computamos el valor de la función en esos argumentos; es decir la aplicación captura la noción de aproximación para el tipo funcional.

4.4 COHERENCIA PARA SUBTIPOS

En la sección anterior explicamos las metodologías involucradas en las pruebas de coherencia y adecuación para un lenguaje con un sistema de tipos simples. En esta sección introducimos una versión extendida del lenguaje que agrega booleanos, pares y subtipado. En este lenguaje extendido coherencia es más demandante ya que las reglas de tipado dejan de ser dirigidas por sintaxis. Es importante remarcar que las pruebas de bracketing y adecuación son modulares, lo cual nos permite reusar de forma directa todo lo realizado para el lenguaje simple y por lo tanto solo necesitamos completar las pruebas para las nuevas construcciones. Para evitar repetición omitimos las construcciones comunes entre los lenguajes.

Definición 36 (Sintaxis)**.**

Valores

$$v \in V ::= \ldots \mid \lceil b \rceil \mid (v, v')$$

Expresiones

$$e \in \Lambda ::= \ldots \mid e \otimes e' \mid e \oslash e' \mid \textbf{\textit{fst}}\ v \mid \textbf{\textit{snd}}\ v$$

Aun distinguimos entre valores V y expresiones Λ. Los nuevos valores son booleanos, con b sobre el conjunto de booleanos $\mathbb{B}$, y pares escritos en ANF; es decir, solo tenemos pares de valores y por lo tanto cuando una expresión computa a un par sus dos componentes están también computadas; también agregamos a las expresiones las proyecciones para operar sobre los pares. Además, tenemos operadores relacionales para números y

operaciones sobre booleanos; gracias a la introducción de booleanos la expresión condicional ahora usará una guarda booleana en lugar de chequear por cero.

Semántica Operacional

Recordemos que la evaluación se basa en definir una relación small-step que depende de los frame stacks. Las proyecciones son similares a la aplicación y por lo tanto no hay necesidad de agregar nuevas construcciones a los frame stacks. Por otro lado, claramente sí necesitamos nuevos frame stacks para las nuevas operaciones, ya que los dos operandos tiene que ser computados antes de computar la operación. Utilizamos $\odot$ como una meta-variable común sobre operaciones booleanos y relacionales.

Definición 37 (Frame Stack)**.** $\mathsf{E} \in \Lambda^\star ::= \ldots \mid \mathsf{E} \circ (\Box \odot \mathsf{e}) \mid \mathsf{E} \circ (\mathsf{e} \odot \Box)$

Introducimos nuevas reglas de reescritura para manipular la evaluación de estas nuevas construcciones. Como ya mencionamos, la expresión condicional **if** ahora tiene una guarda booleana y actualizar su regla es directo; si en el tope del frame stack se encuentra $(\mathbf{if}\,\Box\,(\mathsf{e},\mathsf{e}'))$ y el valor es $\lceil \mathtt{true} \rceil$ entonces evaluamos la expresión e, en caso contrario que el valor sea $\lceil \mathtt{false} \rceil$ evaluamos e', resaltamos que la regla original (IF) no cambia, simplemente extendemos el frame stack para computar la guarda. Notar que si la guarda no computa a un valor booleano entonces la evaluación se detiene (lo mismo ocurre si una proyección no es aplicada a un par).

Definición 38 (Reglas Operacionales)**.**

$$\begin{array}{lll}
\mathsf{E} \circ (\mathit{if}\,\Box\,(\mathsf{e},\mathsf{e}')), \lceil \mathtt{true} \rceil & \longmapsto & \mathsf{E},\mathsf{e} \qquad \text{(IFT)} \\
\mathsf{E} \circ (\mathit{if}\,\Box\,(\mathsf{e},\mathsf{e}')), \lceil \mathtt{false} \rceil & \longmapsto & \mathsf{E},\mathsf{e}' \qquad \text{(IFF)} \\
\mathsf{E}, \mathsf{e} \oslash \mathsf{e}' & \longmapsto & \mathsf{E} \circ (\Box \oslash \mathsf{e}'), \mathsf{e} \qquad \text{(OOP)} \\
\mathsf{E} \circ (\Box \oslash \mathsf{e}'), \lfloor \mathfrak{n} \rfloor & \longmapsto & \mathsf{E} \circ (\lfloor \mathfrak{n} \rfloor \oslash \Box), \mathsf{e}' \qquad \text{(OOP-L)} \\
\mathsf{E} \circ (\lfloor \mathfrak{n} \rfloor \oslash \Box), \lfloor \mathfrak{m} \rfloor & \longmapsto & \mathsf{E}, \lceil \mathfrak{n} \oslash \mathfrak{m} \rceil \qquad \text{(OOP-R)} \\
\mathsf{E}, \mathsf{e} \oslash \mathsf{e}' & \longmapsto & \mathsf{E} \circ (\Box \oslash \mathsf{e}'), \mathsf{e} \qquad \text{(BOP)} \\
\mathsf{E} \circ (\Box \oslash \mathsf{e}'), \lceil \mathsf{b} \rceil & \longmapsto & \mathsf{E} \circ (\lceil \mathsf{b} \rceil \oslash \Box), \mathsf{e}' \qquad \text{(BOP-L)} \\
\mathsf{E} \circ (\lceil \mathsf{b} \rceil \oslash \Box), \lceil \mathsf{b}' \rceil & \longmapsto & \mathsf{E}, \lceil \mathsf{b} \oslash \mathsf{b}' \rceil \qquad \text{(BOP-R)} \\
\mathsf{E}, \mathit{fst}\,(\mathsf{v},\mathsf{v}') & \longmapsto & \mathsf{E},\mathsf{v} \qquad \text{(FST-OP)} \\
\mathsf{E}, \mathit{snd}\,(\mathsf{v},\mathsf{v}') & \longmapsto & \mathsf{E},\mathsf{v}' \qquad \text{(SND-OP)} \\
\mathsf{E}, \lceil \mathsf{b} \rceil & \longmapsto & \mathsf{E}, \lfloor \underline{\mathsf{b}} \rfloor \qquad \text{(CAST)}
\end{array}$$

Las operaciones binarias no introducen ninguna novedad, excepto por el hecho de que tenemos que tener cuidado de chequear que tengamos valores booleanos cuando computamos operaciones booleanas. Sin embargo, en el caso de las operaciones sobre naturales, tal vez aceptemos booleanos, ya que pretendemos agregar la relación de subtipado **bool** $\leqslant$ **nat**. Por lo tanto, agregamos una nueva transición para castear booleanos a naturales, mapeando $\lfloor \mathtt{true} \rfloor$ a $\lfloor 1 \rfloor$ y $\lfloor \mathtt{false} \rfloor$ a $\lfloor 0 \rfloor$. Por ejemplo la expresión $\lceil \mathtt{true} \rceil \oplus \lceil \mathtt{false} \rceil$ evalúa a $\lfloor 1 \rfloor$.

	Frame Stack	Expresión	Regla
1	Id	$\lceil \mathtt{true} \rceil \oplus \lceil \mathtt{false} \rceil$	OP
2	$\mathsf{Id} \circ (\square \oplus \lceil \mathtt{false} \rceil)$	$\lceil \mathtt{true} \rceil$	CAST
3	$\mathsf{Id} \circ (\square \oplus \lceil \mathtt{false} \rceil)$	$\lfloor \mathtt{true} \rfloor$	OP-L
4	$\mathsf{Id} \circ (\lfloor \mathtt{true} \rfloor \oplus \square)$	$\lceil \mathtt{false} \rceil$	CAST
5	$\mathsf{Id} \circ (\lfloor \mathtt{true} \rfloor \oplus \square)$	$\lfloor \mathtt{false} \rfloor$	OP-R
6	Id	$\lfloor 1 \rfloor$	

Semántica Denotacional Extrínseca

La semántica extrínseca que dimos para el lenguaje simple utilizaba la menor solución $\mathcal{V}$ para la ecuación $\mathsf{F}\,X = \mathbb{N} \oplus X \to X$ para dar significado a valores y $\mathcal{V}_\perp$ para expresiones. Claramente, necesitamos añadir productos, aunque decidimos no añadir booleanos. Por lo tanto, ahora $\mathcal{V}$ será la menor solución de $X \cong \mathsf{F}X$ con $\mathsf{F}\,X = \mathbb{N} \oplus (X \to X_\perp) \oplus (X \times X)$. Asumimos que este isomorfismo nos induce

$$\mathcal{V} \underset{\psi}{\overset{\varphi}{\rightleftarrows}} \mathbb{N} \oplus (\mathcal{V} \to \mathcal{V}_\perp) \oplus (\mathcal{V} \times \mathcal{V})$$

e introducimos las inyecciones:

Definición 39.

$$\begin{aligned} \iota_{nat}(\_) &: \mathbb{N} \to \mathcal{V} & \iota_{nat}(\_) &= \psi \circ \iota_0 \circ \iota_0 \\ \iota_{fun}(\_) &: (\mathcal{V} \to \mathcal{V}_\perp) \to \mathcal{V} & \iota_{fun}(\_) &= \psi \circ \iota_0 \circ \iota_1 \\ \iota_{pair}(\_) &: (\mathcal{V} \times \mathcal{V}_\perp) \to \mathcal{V} & \iota_{pair}(\_) &= \psi \circ \iota_1 \end{aligned}$$

La ausencia de booleanos en el dominio semántico se debe a que vamos a interpretar a los valores booleanos con su correspondiente número natural resultado de su coerción.

$$\overline{-} : \mathbb{N} \to \mathbb{B}_\perp \qquad\qquad \underline{-} : \mathbb{B} \to \mathbb{N}$$

$$\overline{x} = \begin{cases} \iota_\uparrow \mathit{false} & \text{if } x = 0 \\ \iota_\uparrow \mathit{true} & \text{if } x = 1 \\ \perp & \text{if } x > 1 \end{cases} \qquad \underline{b} = \begin{cases} 0 & \text{if } \neg\, b \\ 1 & \text{if } b \end{cases}$$

Tal como hicimos con la operación binaria sobre naturales, promovemos y coercionamos las otras operaciones binarias. La relación binaria $\circledcirc : \mathbb{N} \times \mathbb{N} \to \mathbb{B}$ tiene que ser promovida y casteada a $\circledcirc_{\mathbf{nat}} : \mathcal{V} \times \mathcal{V} \to \mathcal{V}_\perp$ donde su definición es

$$v \circledcirc_{\mathbf{nat}} v' = \begin{cases} \iota_\uparrow(\iota_{\mathbf{nat}}(\underline{m \circledcirc n})) & \text{if } v = \iota_{\mathbf{nat}}(m) \text{ and } v' = \iota_{\mathbf{nat}}(n) \\ \perp & \text{caso contrario} \end{cases}$$

El operador booleano $\oslash : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$, requiere un poco más de trabajo, ya que estamos codificando los booleanos como naturales. Definimos la promoción $\oslash_{\mathbf{bool}} : \mathcal{V} \times \mathcal{V} \to \mathcal{V}_\perp$ validando si los operandos son resultado de castear a booleano.

$$v \oslash_{\mathbf{bool}} v' = \begin{cases} \iota_\uparrow(\iota_{\mathbf{nat}}(\underline{b \oslash b'})) & \text{if } v = \iota_{\mathbf{nat}}(\underline{b}) \text{ and } v' = \iota_{\mathbf{nat}}(\underline{b'}) \\ \perp & \text{caso contrario} \end{cases}$$

Ya que los operadores aceptan expresiones en su construcción sintáctica, tenemos que lidiar con no-terminación (y con expresiones mal tipadas); en consecuencia utilizamos la extensión bi-estricta de $\circledcirc_{\mathbf{nat}}$ y $\oslash_{\mathbf{bool}}$, denotadas respectivamente $\circledcirc_{\mathbf{nat}_\perp} : \mathcal{V}_\perp \times \mathcal{V}_\perp \to \mathcal{V}_\perp$ y $\oslash_{\mathbf{bool}_\perp} : \mathcal{V}_\perp \times \mathcal{V}_\perp \to \mathcal{V}_\perp$.

En la siguiente definición, como ocurría cuando presentamos la sintaxis, omitimos las ecuaciones para valores y expresiones que ya están presentes en el lenguaje simple; notar que las ecuaciones de la definición 17 son válidas en este contexto cuando interpretamos $\iota_{\mathbf{nat}}(-)$ y $\iota_{\mathbf{fun}}(-)$ utilizando la definición 39.

Definición 40 (Semántica Extrínseca)**.**

$$\begin{array}{ll} [\![\_]\!]^{ev} : V(X) \to En v_X \to \mathcal{V} & [\![\_]\!]^{e\Lambda} : \Lambda(X) \to En v_X \to \mathcal{V}_\perp \\ [\![\lceil b \rceil]\!]^{ev}\eta = \iota_{\boldsymbol{nat}}(\underline{b}) & [\![e \otimes e']\!]^{e\Lambda}\ \eta = ([\![e]\!]^{e\Lambda}\eta) \otimes_{\boldsymbol{nat}_\perp} ([\![e']\!]^{e\Lambda}\eta) \\ [\![(v, v')]\!]^{ev}\eta = \iota_{\boldsymbol{pair}}(([\![v]\!]^{ev}\eta, [\![v']\!]^{ev}\eta)) & [\![e \oslash e']\!]^{e\Lambda}\ \eta = ([\![e]\!]^{e\Lambda}\eta) \oslash_{\boldsymbol{bool}_\perp} ([\![e']\!]^{e\Lambda}\eta) \end{array}$$

$$[\![\boldsymbol{fst}\ v]\!]^{e\Lambda}\ \eta = (\iota_\uparrow \circ \pi_1)_{\boldsymbol{pair}}([\![v]\!]^{ev}\eta)$$
$$[\![\boldsymbol{snd}\ v]\!]^{e\Lambda}\ \eta = (\iota_\uparrow \circ \pi_2)_{\boldsymbol{pair}}([\![v]\!]^{ev}\eta)$$
$$[\![\boldsymbol{if}\ e\ \boldsymbol{then}\ e'\ \boldsymbol{else}\ e'']\!]^{e\Lambda}\ \eta = \begin{cases} [\![e']\!]^{e\Lambda}\eta & si\ [\![e]\!]^{e\Lambda}\eta = \iota_\uparrow(\iota_{\boldsymbol{nat}}(1)) \\ [\![e'']\!]^{e\Lambda}\eta & si\ [\![e]\!]^{e\Lambda}\eta = \iota_\uparrow(\iota_{\boldsymbol{nat}}(0)) \\ \perp & caso\ contrario \end{cases}$$

Semántica Denotacional Intrínseca

La actualización en el sistema de tipos es significante ya que añadir subtipos transforma a coherencia en una pregunta más interesante. Agregamos dos nuevas construcciones de tipos para los booleanos y los pares.

Definición 41 (Tipos)**.**

$$\theta \in Type ::= \boldsymbol{bool} \mid \boldsymbol{nat} \mid \theta \to \theta' \mid \theta \times \theta'$$

Añadimos además un nueva forma de juicio de tipado $\theta \leqslant \theta'$, el cual se lee como θ es un *subtipo* de θ'. Conceptualmente, esto significa que en cualquier contexto donde esperamos una expresión de tipo θ' podemos utilizar una expresión de tipo θ.

Definición 42 (Reglas de Subtipado)**.**

$$\frac{}{\boldsymbol{bool} \leqslant \boldsymbol{nat}} \qquad \frac{}{\theta \leqslant \theta} \qquad \frac{\theta \leqslant \theta' \qquad \theta' \leqslant \theta''}{\theta \leqslant \theta''}$$

$$\frac{\theta_0' \leqslant \theta_0 \qquad \theta_1 \leqslant \theta_1'}{\theta_0 \to \theta_1 \leqslant \theta_0' \to \theta_1'} \qquad \frac{\theta_0 \leqslant \theta_0' \qquad \theta_1 \leqslant \theta_1'}{\theta_0 \times \theta_1 \leqslant \theta_0' \times \theta_1'}$$

Los únicos axiomas son la coerción de booleanos a naturales y la reflexividad del subtipado. Las otras tres reglas son las usuales: transitividad,

congruencia de pares y la regla contra-variante del subtipado para el tipo funcional.

Respecto de las reglas de tipado que expresan el tipo que tienen las expresiones y valores, agregamos las reglas para las nuevas construcciones y la regla de *subsunción* que nos permite cambiar el tipo de una expresión por un super-tipo; es decir, si tenemos que $\pi \vdash e : \theta$ y $\theta \leqslant \theta'$ entonces podemos deducir $\pi \vdash e : \theta'$.

Definición 43 (Reglas de Tipado)**.**

$$\frac{}{\pi \vdash^{V} \lceil b \rceil : \textbf{\textit{bool}}} \qquad \frac{\pi \vdash^{\wedge} e : \textbf{\textit{nat}} \qquad \pi \vdash^{\wedge} e' : \textbf{\textit{nat}}}{\pi \vdash^{\wedge} e \oplus e' : \textbf{\textit{nat}}}$$

$$\frac{\pi \vdash^{\wedge} e : \textbf{\textit{nat}} \qquad \pi \vdash^{\wedge} e' : \textbf{\textit{nat}}}{\pi \vdash^{\wedge} e \otimes e' : \textbf{\textit{bool}}} \qquad \frac{\pi \vdash^{\wedge} e : \textbf{\textit{bool}} \qquad \pi \vdash^{\wedge} e' : \textbf{\textit{bool}}}{\pi \vdash^{\wedge} e \oslash e' : \textbf{\textit{bool}}}$$

$$\frac{\pi \vdash^{V} v : \theta \qquad \pi \vdash^{V} v' : \theta'}{\pi \vdash^{V} (v, v') : \theta \times \theta'} \qquad \frac{\pi \vdash^{V} v : \theta \times \theta'}{\pi \vdash^{\wedge} \textbf{\textit{fst}}\ v : \theta} \qquad \frac{\pi \vdash^{V} v : \theta \times \theta'}{\pi \vdash^{\wedge} \textbf{\textit{snd}}\ v : \theta'}$$

$$\frac{\pi \vdash^{V} v : \theta \qquad \theta \leqslant \theta'}{\pi \vdash^{V} v : \theta'} \qquad \frac{\pi \vdash^{\wedge} e : \theta \qquad \theta \leqslant \theta'}{\pi \vdash^{\wedge} e : \theta'}$$

$$\frac{\pi \vdash^{\wedge} e : \textbf{\textit{bool}} \qquad \pi \vdash^{\wedge} e' : \theta \qquad \pi \vdash^{\wedge} e'' : \theta}{\pi \vdash^{\wedge} \textbf{\textit{if}}\ e\ \textbf{\textit{then}}\ e'\ \textbf{\textit{else}}\ e'' : \theta}$$

La semántica intrínseca considerando los nuevos tipos (booleanos y producto) es directa; notar que intrínsecamente denotamos a los booleanos con el conjunto $\mathbb{B}$, en contraste con lo que ocurría extrínsecamente.

Definición 44 (Semántica Intrínseca de Tipos)**.**

$$[\![\textbf{\textit{bool}}]\!]^i = \mathbb{B} \qquad [\![\theta \times \theta']\!]^i = [\![\theta]\!]^i \times [\![\theta']\!]^i$$

Antes de extender la semántica de los nuevos juicios de tipado tenemos que dar semántica a los juicios de subtipado $\theta \leqslant \theta'$; definida por recursión en las derivaciones.

Definición 45 (Semántica Intrínseca para las Reglas de Subtipado)**.**

$$\begin{aligned}
[\![\theta \leqslant \theta']\!]^{is} &: [\![\theta]\!]^i \to [\![\theta']\!]^i \\
[\![\textbf{\textit{bool}} \leqslant \textbf{\textit{nat}}]\!]^{is} &= b \mapsto \underline{b} \\
[\![\theta \leqslant \theta]\!]^{is} &= \mathrm{id} \\
[\![\theta \leqslant \theta'']\!]^{is} &= [\![\theta' \leqslant \theta'']\!]^{is} \circ [\![\theta \leqslant \theta']\!]^{is} \\
[\![\theta_0 \times \theta_1 \leqslant \theta_0' \times \theta_1']\!]^{is} &= [\![\theta_0 \leqslant \theta_0']\!]^{is} \times [\![\theta_1 \leqslant \theta_1']\!]^{is} \\
[\![\theta_0 \to \theta_1 \leqslant \theta_0' \to \theta_1']\!]^{is} &= f \mapsto (\iota_\uparrow \circ [\![\theta_1 \leqslant \theta_1']\!]^{is})_\perp \circ f \circ [\![\theta_0' \leqslant \theta_0]\!]^{is}
\end{aligned}$$

En el caso base de **bool** $\leqslant$ **nat** necesitamos definir una función continua de $\mathbb{B}$ en $\mathbb{N}$, es decir, convertir un booleano en un número natural, para esto utilizamos la función ya definida $\underline{}$. El caso de la reflexividad y transitividad son directos; es importante recordar que estamos haciendo abuso de notación, por lo tanto en el caso de la transitividad lo que realmente estamos expresando es $[\![\mathcal{D}]\!]^{is} = [\![\mathcal{D}'']\!]^{is} \circ [\![\mathcal{D}']\!]^{is}$ donde:

$$\begin{array}{c}
\mathcal{D}' : \dfrac{\vdots}{\theta \leqslant \theta'} \qquad \mathcal{D}'' : \dfrac{\vdots}{\theta' \leqslant \theta''} \\
\mathcal{D} : \dfrac{\qquad\qquad}{\theta \leqslant \theta''}
\end{array}$$

El significado de subtipado de pares esta dado por el producto de las funciones de subtipado para cada componente del par. Finalmente, considerando la derivación cuya conclusión es $\theta_0 \to \theta_1 \leqslant \theta_0' \to \theta_1'$, su semántica estará definida utilizando las sub-derivaciones para tener funciones $[\![\theta_0' \leqslant \theta_0]\!]^{is} : [\![\theta_0']\!]^i \to [\![\theta_0]\!]^i$ y $[\![\theta_1 \leqslant \theta_1']\!]^{is} : [\![\theta_1]\!]^i \to [\![\theta_1']\!]^i$. Luego, tenemos que construir una función continua de $[\![\theta_0]\!]^i \to [\![\theta_1]\!]^i_\perp$ a $[\![\theta_0']\!]^i \to [\![\theta_1']\!]^i_\perp$; primero podemos pre-componer el argumento, digamos f, para obtener $f \circ [\![\theta_0' \leqslant \theta_0]\!]^{is} : [\![\theta_0']\!]^i \to [\![\theta_1]\!]^i_\perp$; ahora, para poder post-componer esta función con la función $[\![\theta_1 \leqslant \theta_1']\!]^{is}$ tenemos que promoverla.

La semántica de las nuevas construcciones es directa; notar que la regla de subsunción es interpretada como la composición en la categoría apropiada. Además, para definir la semántica de la expresión condicional utilizamos la construcción condicional del meta-lenguaje definida como:

$$\textsc{if}\ cond\ \textsc{then}\ a\ \textsc{else}\ b = \begin{cases} a & \text{si } cond = \lceil true \rceil \\ b & \text{si } cond = \lceil false \rceil \\ \perp & \text{si } cond = \perp \end{cases}$$

Definición 46 (Semántica Intrínseca)**.**

$$\begin{aligned}
[\![\pi \vdash^{V} v : \theta]\!]^{iv} &: [\![\pi]\!] \to [\![\theta]\!]^{i} \\
[\![\pi \vdash^{V} \lceil b \rceil : \textbf{\textit{bool}}]\!]^{iv}\xi &= b \\
[\![\pi \vdash^{V} (v, v') : \theta \times \theta']\!]^{iv}\xi &= ([\![\pi \vdash^{V} v : \theta]\!]^{iv}\xi\,,\, [\![\pi \vdash^{V} v' : \theta']\!]^{iv}\xi) \\
[\![\pi \vdash^{V} v : \theta']\!]^{iv}\xi &= [\![\theta \leqslant \theta']\!]^{is}\,([\![\pi \vdash^{V} v : \theta]\!]^{iv}\xi)
\end{aligned}$$

$$\begin{aligned}
[\![\pi \vdash^{\wedge} e : \theta]\!]^{i\wedge} &: [\![\pi]\!] \to [\![\theta]\!]^{i}_{\perp} \\
[\![\pi \vdash^{\wedge} \textbf{\textit{fst}}\ v : \theta]\!]^{i\wedge}\xi &= (\iota_{\uparrow} \circ \pi_1)\,([\![\pi \vdash^{V} v : \theta \times \theta']\!]^{iv}\xi) \\
[\![\pi \vdash^{\wedge} \textbf{\textit{snd}}\ v : \theta']\!]^{i\wedge}\xi &= (\iota_{\uparrow} \circ \pi_2)\,([\![\pi \vdash^{V} v : \theta \times \theta']\!]^{iv}\xi) \\
[\![\pi \vdash^{\wedge} e \oslash e' : \textbf{\textit{bool}}]\!]^{i\wedge}\,\xi &= ([\![\pi \vdash^{\wedge} e : \textbf{\textit{nat}}]\!]^{i\wedge}\xi) \oslash_{\perp} ([\![\pi \vdash^{\wedge} e' : \textbf{\textit{nat}}]\!]^{i\wedge}\xi) \\
[\![\pi \vdash^{\wedge} e \oslash e' : \textbf{\textit{bool}}]\!]^{i\wedge}\,\xi &= ([\![\pi \vdash^{\wedge} e : \textbf{\textit{bool}}]\!]^{i\wedge}\xi) \oslash_{\perp} ([\![\pi \vdash^{\wedge} e' : \textbf{\textit{bool}}]\!]^{i\wedge}\xi) \\
[\![\pi \vdash^{\wedge} \textbf{\textit{if}}\ e\ \textbf{\textit{then}}\ e'\ \textbf{\textit{else}}\ e'' : \theta]\!]^{i\wedge}\xi &= \textsc{if}\ [\![\pi \vdash^{\wedge} e : \textbf{\textit{bool}}]\!]^{i\wedge}\xi \\
&\quad\ \textsc{then}\ [\![\pi \vdash^{\wedge} e' : \theta]\!]^{i\wedge}\xi \\
&\quad\ \textsc{else}\ [\![\pi \vdash^{\wedge} e'' : \theta]\!]^{i\wedge}\xi \\
[\![\pi \vdash^{\wedge} e : \theta']\!]^{i\wedge}\xi &= (\iota_{\uparrow} \circ [\![\theta \leqslant \theta']\!]^{is})_{\perp}\,([\![\pi \vdash^{\wedge} e : \theta]\!]^{i\wedge}\xi)
\end{aligned}$$

El Significado de los (Sub)Tipos

En el lenguaje extendido coherencia es mucho más interesante porque existen diferentes derivaciones de un mismo juicio de tipado. Un pequeño ejemplo puede ser el de la suma de booleanos $\lceil \mathtt{true} \rceil \oplus \lceil \mathtt{false} \rceil$, que se traduce a una expresión del lenguaje como:

let oplus = (**fun** _ x = (**fun** _ y = x ⊕ y)) **in**
  let true_oplus = oplus ⌈true⌉ **in** true_oplus ⌈false⌉

Es evidente que oplus puede tener el tipo **nat** → **nat** → **nat**; además podemos dar otra derivación que prueba que tiene tipo **bool** → **bool** → **nat**. Evitemos oscurecer la derivación con demasiado detalle y concedámosnos la libertad de no tipar la variable correspondiente con el argumento funcional.

$$\dfrac{\dfrac{\dfrac{x : \mathbf{nat} \in \{x : \mathbf{nat}, y : \mathbf{nat}\}}{x : \mathbf{nat}, y : \mathbf{nat} \vdash x : \mathbf{nat}} \quad \dfrac{y : \mathbf{nat} \in \{x : \mathbf{nat}, y : \mathbf{nat}\}}{x : \mathbf{nat}, y : \mathbf{nat} \vdash y : \mathbf{nat}}}{x : \mathbf{nat}, y : \mathbf{nat} \vdash x \oplus y : \mathbf{nat}}}{x : \mathbf{nat} \vdash \mathbf{fun}\ \_\ y = x \oplus y : \mathbf{nat} \to \mathbf{nat}}}{\vdash \mathbf{fun}\ \_\ x = (\mathbf{fun}\ \_\ y = x \oplus y) : \mathbf{nat} \to \mathbf{nat} \to \mathbf{nat}}$$

A partir de esta derivación podemos utilizar subsunción para deducir que oplus tiene tipo **bool** $\to$ **bool** $\to$ **nat**; otra posibilidad, es utilizar subsunción en cada argumento. Sea π el contexto x: **bool**, y: **bool**.

$$\dfrac{\dfrac{\dfrac{\dfrac{x : \mathbf{bool} \in \pi}{\pi \vdash x : \mathbf{bool}} \quad \dfrac{}{\mathbf{bool} \leqslant \mathbf{nat}}}{\pi \vdash x : \mathbf{nat}} \quad \dfrac{\dfrac{\dfrac{y : \mathbf{bool} \in \pi}{\pi \vdash y : \mathbf{bool}} \quad \dfrac{}{\mathbf{bool} \leqslant \mathbf{nat}}}{\pi \vdash y : \mathbf{nat}}}{\pi \vdash x \oplus y : \mathbf{nat}}}{x : \mathbf{bool} \vdash \mathbf{fun}\ \_\ y = x \oplus y : \mathbf{bool} \to \mathbf{nat}}}{\vdash \mathbf{fun}\ \_\ x = (\mathbf{fun}\ \_\ y = x \oplus y) : \mathbf{bool} \to \mathbf{bool} \to \mathbf{nat}}$$

Es fácil completar las dos derivaciones para tipar la expresión completa con los operadores aplicados; en el primer caso tenemos que coercionar ambos valores booleanos a naturales. Las derivaciones completas, junto con la aplicación de coherencia a este ejemplo, puede encontrarse en la formalización.

Recordemos que la prueba de coherencia de Reynolds define una relación lógica entre las semánticas intrínseca y extrínseca. En algún sentido, esta relación lógica describe las coerciones $\downarrow_\theta$ y $\uparrow_\theta$ entre la interpretación intrínseca de los tipos y el dominio utilizado en la semántica extrínseca. Actualizamos la definición de los embedding-projection pairs entre la semántica intrínseca de los tipos y el dominio $\mathcal{V}$ por recursión en los tipos.

Definición 47 (Embedding-projections pairs)**.** *Definimos simultáneamente las funciones continuas* $\downarrow_\theta : [\![\theta]\!]^i \to \mathcal{V}$ *y* $\uparrow_\theta : \mathcal{V} \to [\![\theta]\!]^i_\perp$.

Embeddings

$$\begin{aligned}
\downarrow_{\mathit{bool}} b &= \iota_{\mathit{nat}}(\underline{b}) \\
\downarrow_{\mathit{nat}} n &= \iota_{\mathit{nat}}(n) \\
\downarrow_{\theta \times \theta'} (p_1, p_2) &= \iota_{\mathit{pair}}((\downarrow_\theta p_1, \downarrow_{\theta'} p_2)) \\
\downarrow_{\theta \to \theta'} f &= \iota_{\mathit{fun}}((\downarrow_{\theta'})_\perp \circ f_\perp \circ \uparrow_\theta)
\end{aligned}$$

Projections

$$
\begin{aligned}
\uparrow_{\textbf{\textit{bool}}} &= [n \mapsto \overline{n}, f \mapsto \bot, p \mapsto \bot] \\
\uparrow_{\textbf{\textit{nat}}} &= [\iota_{\uparrow}, f \mapsto \bot, p \mapsto \bot] \\
\uparrow_{\theta \times \theta'} &= [n \mapsto \bot, f \mapsto \bot, \uparrow_{\theta} \otimes \uparrow_{\theta'}] \\
\uparrow_{\theta \to \theta'} &= [n \mapsto \bot, g \mapsto (\uparrow_{\theta'})_{\bot} \circ g \circ \downarrow_{\theta}, p \mapsto \bot]
\end{aligned}
$$

Las proyecciones de $\mathcal{V}$ a $[\![\theta]\!]^i_{\bot}$ están definidas utilizando el morfismo mediador del co-producto y el isomorfismo entre $\mathcal{V}$ y el co-producto $\mathbb{N} \oplus (\mathcal{V} \to \mathcal{V}_{\bot}) \oplus (\mathcal{V} \times \mathcal{V})$. Notar que como resultado de interpretar los booleanos como naturales, los coercionamos cuando vamos de la interpretación intrínseca de los tipos a la versión extrínseca; en el otro sentido, solo mapeamos 0 y 1 como booleanos. Al igual que antes, utilizamos $\bot$ para representar un error de tipos.

Teorema de Bracketing

El contenido conceptual de una relación lógica está dado por la propiedad fundamental la cual expresa que las semánticas de una expresión en los dos modelos involucrados en la relación efectivamente están relacionadas. En nuestro caso, la propiedad fundamental expresa que la semántica intrínseca de una expresión bien tipada está relacionada con su semántica extrínseca bajo la familia de relaciones $\rho[\theta] \subseteq [\![\theta]\!]^i_{\bot} \times \mathcal{V}_{\bot}$, la cual es resultado de la extensión estricta de la relación para valores $\delta[\theta] \subseteq [\![\theta]\!]^i \times \mathcal{V}$.

La definición para los tipos natural y funcional no cambia; en el caso de los booleanos y los pares es directa: un booleano "intrínseco" está relacionado con un natural "extrínseco" si este último es el resultado de coercionar el primero. Además, los pares están relacionados si sus correspondientes componentes lo están.

Definición 48 (Relación Lógica entre las semánticas intrínseca y extrínseca)**.**

$$
\frac{}{(b, \iota_{\textbf{\textit{nat}}}(\underline{b})) \in \delta[\textbf{\textit{bool}}]} \qquad \frac{}{(n, \iota_{\textbf{\textit{nat}}}(n)) \in \delta[\textbf{\textit{nat}}]}
$$

$$
\frac{(p_1, q_1) \in \delta[\theta] \qquad (p_2, q_2) \in \delta[\theta']}{((p_1, p_2), \iota_{\textbf{\textit{pair}}}((q_1, q_2))) \in \delta[\theta \times \theta']} \qquad \frac{(f\ x, g\ y) \in \rho[\theta], \textit{for all } (x, y) \in \delta[\theta']}{(f, \iota_{\textbf{\textit{fun}}}(g)) \in \delta[\theta' \to \theta]}
$$

Utilizando la misma estrategia que en la sección 4.2 probamos el siguiente lema, que nos asegura que las relaciones son cerradas por supremos de cadenas; este resultado es importante para probar el teorema de bracketing.

Lema 28. *Para todo* $\theta \in \mathtt{Type}$, $\delta[\theta]$ *y* $\rho[\theta]$ *son cerradas por supremos de cadenas. Más aun,* $\rho[\theta]$ *es estricta.*

Es importante notar que las pruebas realizadas para el lenguaje simple para el tipo de los naturales y funcionales son las mismas. En este punto podemos apreciar la modularidad del método de prueba, esto se debe principalmente a que la prueba procede por inducción en los tipos.

Teorema 10 (Bracketing)**.**

1. *Si* $v \in [\![\theta]\!]^i$, *entonces* $(v, \downarrow_\theta v) \in \delta[\theta]$.
2. *Si* $(v, v') \in \delta[\theta]$, *entonces* $\iota_\uparrow(v) = \uparrow_\theta v'$.
3. *Si* $d \in [\![\theta]\!]^i_\perp$, *entonces* $(d, (\iota_\uparrow \circ \downarrow_\theta)_\perp d) \in \rho[\theta]$.
4. *Si* $(d, d') \in \rho[\theta]$, *entonces* $d = (\uparrow_\theta)_\perp d'$.

Corolario 4 (Embedding-projection pairs)**.** $\uparrow_\theta \circ \downarrow_\theta = \iota_\uparrow$.

Omitimos las definiciones de las relaciones lógicas no-cerradas (es decir, las relaciones lógicas para términos con variables libres), ya que son exactamente iguales a las de la definición 24. El siguiente lema demuestra que el subtipado conserva las relaciones lógicas; notar que la coerción por el subtipado solo se realiza en la parte intrínseca.

Lema 29. *Sea* $\theta \leqslant \theta'$.

1. *Si* $(f, g) \in \Delta[\pi, \theta]$, *entonces* $([\![\theta \leqslant \theta']\!]^{is} \circ f, g) \in \Delta[\pi, \theta']$
2. *Si* $(h, k) \in P[\pi, \theta]$, *entonces* $((\iota_\uparrow \circ [\![\theta \leqslant \theta']\!]^{is})_\perp \circ h, k) \in P[\pi, \theta']$

La estrategia de prueba de la propiedad fundamental es la misma, es decir procedemos por inducción mutua en la derivación del juicio de tipado para valores y expresiones; el caso de subsunción se desprende directamente gracias a este lema.

Teorema 11 (Propiedad Fundamental de las Relaciones Lógicas)**.**

1. *Si* $\pi \vdash^V v : \theta$, *entonces* $([\![\pi \vdash^V v : \theta]\!]^{iv}, [\![v]\!]^{ev}) \in \Delta[\pi, \theta]$
2. *Si* $\pi \vdash^\Lambda e : \theta$, *entonces* $([\![\pi \vdash^\Lambda e : \theta]\!]^{i\Lambda}, [\![e]\!]^{e\Lambda}) \in P[\pi, \theta]$

Utilizando el resultado previo obtenemos una caracterización de la semántica intrínseca de las expresiones (y valores) en términos de la semántica extrínseca.

Teorema 12. $[\![\pi \vdash^{\wedge} e : \theta]\!]^{i\wedge} = \uparrow_{\theta\perp} \circ [\![e]\!]^{e\wedge} \circ \downarrow_{\pi}$.

Finalmente, obtenemos el resultado de coherencia como un corolario directo de esta caracterización.

Corolario 5 (Coherencia). *Sean $\mathcal{D}$ y $\mathcal{D}'$ diferentes derivaciones del juicio de tipado $\pi \vdash^{\wedge} e : \theta$, entonces $[\![\mathcal{D}]\!]^{i\wedge} = [\![\mathcal{D}']\!]^{i\wedge}$.*

4.5 ADECUACIÓN PARA SUBTIPOS

En esta sección extendemos las relaciones lógicas entre los valores sintácticos y sus denotaciones, las cuales utilizaremos para probar la adecuación operacional con respecto a la denotacional para el lenguaje extendido. Recordemos que la prueba de adecuación involucra probar dos aspectos por separado; considerando expresiones divergentes y convergentes. La adecuación para programas divergentes la probamos con respecto a la semántica extrínseca, mientras que utilizamos la semántica intrínseca para la prueba con respecto a los programas convergentes.

Adecuación para expresiones divergentes

Dada una expresión bien tipada $\vdash e : \theta$, queremos probar que $[\![e]\!]^{e\wedge}() = \perp$ implica que la evaluación de e diverge. Siguiendo la misma estrategia que utilizamos en la sección 4.3 para el lenguaje simple, definimos dos relaciones lógicas con la propiedad de step-indexed tal que relacionen valores y expresiones sintácticas con dominios semánticos $\mathcal{V}$ y $\mathcal{V}_{\perp}$, respectivamente. Haciendo uso de la modularidad de biortogonalidad solamente necesitamos agregar a las relaciones lógicas las nuevas definiciones con respecto a los nuevos tipos y probar los mismos lemas y teoremas. Primero definimos la relación para valores sobre tipos básicos y luego la relación sobre expresiones queda definida utilizando el operador biortogonal y la extensión dada por la definición 26.

Definición 49 (Aproximación Operacional). *Sea $\lhd_i^{\theta} \subseteq V \times \mathcal{V}$, definimos $\blacktriangleleft_i^{\theta} \subseteq \Lambda \times \mathcal{V}_{\perp}$ como*

$$e \blacktriangleleft_0^{\theta} \perp \quad y \quad e \blacktriangleleft_n^{\theta} d \text{ sii } (n, e) \in (\Omega^{\lhd_n^{\theta}} d)^{\perp\top}$$

Definición 50 (Aproximación Operacional para Valores)**.**

$$\frac{}{\lceil b \rceil \lhd_n^{\mathit{bool}} \iota_{\mathit{nat}}(\underline{b})} \qquad \frac{}{\lceil b \rceil \lhd_n^{\mathit{nat}} \iota_{\mathit{nat}}(\underline{b})} \qquad \frac{}{\lfloor m \rfloor \lhd_n^{\mathit{nat}} \iota_{\mathit{nat}}(m)}$$

$$\frac{\forall m < n, \quad v \lhd_m^{\theta} p_1 \quad y \quad v' \lhd_m^{\theta'} p_2}{(v, v') \lhd_n^{\theta \times \theta'} \iota_{\mathit{pair}}((p_1, p_2))}$$

$$\frac{\forall m < n \quad y \quad v \lhd_m^{\theta'} v, \quad e[x/v, f/(\mathbf{\mathit{fun}}\ f\ x = e)] \blacktriangleleft_m^{\theta} f v}{\mathbf{\mathit{fun}}\ f\ x = e \lhd_n^{\theta' \to \theta} \iota_{\mathit{fun}}(f)}$$

La definición de la relación para valores para el caso del producto es directa: un par sintáctico aproxima a un par denotacional si cada componente sintáctico aproxima a su contra-parte denotacional en un índice menor. Notar que cualquier booleano sintáctico aproxima a su denotación tanto booleana como natural. Es directo probar que estas relaciones son cerradas bajo subtipado.

Lema 30 (Cerradas bajo Subtipado)**.** *Sea* $\theta \leqslant \theta'$,

1. *Para todo* n, $\_ \lhd_n^{\theta} \_ \subseteq \_ \lhd_n^{\theta'} \_$.
2. *Para todo* n, $\_ \blacktriangleleft_n^{\theta} \_ \subseteq \_ \blacktriangleleft_n^{\theta'} \_$.

Estas relaciones son step-indexed (cf. Lema. 24). La extensión de estas relaciones para términos no-cerrados la realizamos como en la definición 30 utilizando la misma aproximación de substituciones y entornos (cf. Definición. 29). La prueba de la propiedad fundamental de las relaciones lógicas (cf. Teorema. 5) es directa: el caso de subsunción lo probamos utilizando el lema 30. Finalmente, completamos la prueba de adecuación para expresiones divergentes notando que si una expresión aproxima a $\bot$ para cualquier natural, entonces la configuración (ld, e) puede realizar alguna cantidad arbitraria de transiciones.

Adecuación para Expresiones Convergentes

La pieza que resta probar para tener adecuación completa es demostrar que si la denotación de una expresión para cualquier tipo básico es una constante, entonces la evaluación operacional evalúa a ese valor. Recordemos que una observación, en este contexto, será un subconjunto de configuraciones cerrado por anti-ejecución; esta observación la utilizamos para definir las relaciones lógicas para expresiones utilizando biortogonalidad y la relación lógica correspondiente a los valores.

Definición 51 (Aproximación Denotacional)**.**

$$e \blacktriangleright^{\theta} d \textit{ sii } e \in (\triangleright^{\theta} \nu)^{\bot\top}, \textit{ para todo } \nu \in [\![\theta]\!]^{i} \textit{ tal que } d = \iota_{\uparrow}(\nu) \ .$$

Definición 52 (Aproximación Denotacional para Valores)**.**

$$\frac{}{\lceil b \rceil \triangleright^{\textit{bool}} b} \qquad \frac{}{\lceil b \rceil \triangleright^{\textit{nat}} \underline{b}} \qquad \frac{}{\lfloor m \rfloor \triangleright^{\textit{int}} m} \qquad \frac{v \triangleright^{\theta} \nu \qquad v' \triangleright^{\theta'} \nu'}{(v, v') \triangleright^{\theta \times \theta'} (\nu, \nu')}$$

$$\frac{e[x/v, f/(\textit{fun } f\ x = e)] \blacktriangleright^{\theta} f\nu, \quad \textit{para todo } v \triangleright^{\theta'} \nu}{\textit{fun } f\ x = e \triangleright^{\theta' \to \theta} f}$$

Similar a lo que ocurre con la aproximación operacional, los valores sintácticos booleanos son aproximados por su representación tanto natural como booleana. Notar que estas aproximaciones tiene una relación cercana con la regla operacional de CAST; es decir, podemos probar utilizando esta regla que la expresión $\lceil \mathit{true} \rceil$ evalúa a $\lfloor 1 \rfloor$.

Estas aproximaciones denotacionales son cerradas por subtipado, aunque como estamos en presencia de un modelo intrínseco tenemos que utilizar las funciones de coerción dadas por la interpretación de los juicios de subtipado.

Lema 31. *Sea* $\theta \leqslant \theta'$.

1. *Si* $v \triangleright^{\theta} \nu$, *entonces* $v \triangleright^{\theta'} [\![\theta \leqslant \theta']\!]^{is} \circ \nu$.
2. *Si* $e \blacktriangleright^{\theta} e$, *entonces* $e \blacktriangleright^{\theta'} (\iota_{\uparrow} \circ [\![\theta \leqslant \theta']\!]^{is})_{\bot} \circ e$.

Al igual que para el lenguaje simple (cf. Teorema. 7) expresamos la propiedad fundamental de las relaciones lógicas en términos de las relaciones Scott clausuradas. Para obtener la adecuación para cualquier expresión bien tipada con tipo **nat**, fijamos la observación como el conjunto de configuraciones $\{(E, e) \mid (E, e) \longmapsto^{*} (\mathsf{Id}, \lfloor n \rfloor)\}$.

Corolario 6 (Adecuación para Expresiones Convergentes)**.**
Si $[\![\vdash^{\Lambda} e : \textit{int}]\!]^{i\Lambda}[] = \iota_{\uparrow}(n)$, *entonces* $(\mathsf{Id}, e) \longmapsto^{*} (\mathsf{Id}, \lfloor n \rfloor)$.

Además, podemos utilizar este corolario y obtener la adecuación con respecto a la semántica extrínseca.

Corolario 7. *Si* $\vdash^{\Lambda} e : \textit{int}$ *y* $[\![e]\!]^{e\Lambda}() = \iota_{\textit{nat}}(n)$, *entonces* $(\mathsf{Id}, e) \longmapsto^{*} (\mathsf{Id}, \lfloor n \rfloor)$.

Finalmente, es interesante remarcar que los resultados, Corolario 6 y Teorema 7 también son validos para booleanos; cambiando la observación por el conjunto de configuraciones $\{(E, e) \mid (E, e) \longmapsto^{*} (\mathsf{Id}, \lceil b \rceil)\}$.

5

BIORTOGONALIDAD PARA UN LENGUAJE LAZY

En este capítulo extendemos el uso de biortogonalidad para probar la corrección de un compilador para un lenguaje lazy básico con recursión hacia una variante de la máquina abstracta descripta por Sestoft [46]. Particularmente queremos probar la corrección del compilador con respecto a una semántica denotacional, es decir queremos probar que un código o programa de bajo nivel generado por el compilador "implementa" el objeto matemático que es la denotación de la expresión de alto nivel que se compiló. El desarrollo toma como base los estudios preliminares realizador por Rodríguez [41], que explora biortogonalidad para un lenguaje lazy simple sin recursión.

En capítulos pasados vimos que la utilización de biortogonalidad trae asociada alguna noción de observación sobre las configuración de la maquina abstracta. En lenguaje con estrategia de evaluación call-by-value (por ejemplo [1, 21, 22]) las configuraciones pueden separarse en el realizador y el test; esta clara separación entre realizadores y tests también ocurre para el caso de lenguajes con estrategia de evaluación call-by-name [26]. Una de las dificultades técnicas que aparece en el caso de evaluación lazy (call-by-need) es la imposibilidad de una separación clara entre los realizadores y los tests; uno de los avances preliminares de Rodríguez [41] es resolver parcialmente este problema para un lenguaje simple sin recursión el cual nosotros extendemos con recursión.

5.1 LENGUAJE DE ALTO NIVEL

Nuestro lenguaje fuente es un cálculo lambda extendido con constantes, operadores binarios, productos y expresión condicional (chequeando por cero). Este lenguaje está inspirado principalmente en los lenguajes presentados en Launchbury [25] y Sestoft [46], en nuestro caso hacemos uso de índices de De Bruijn para las variables y la expresión *let* recursiva solamente permite introducir una variable. Para referirnos a la variable n utilizamos $\overline{n}$ y a la constante m la denotaremos como $\lfloor m \rfloor$. Si bien el lenguaje restringe la aplicación; el operando es siempre una variable, esto no es mayor proble-

ma ya que es fácil transformar una aplicación no restringida usando una definición: $t\ t'$ puede ser escrito como *let* t' *in* $t\ \bar{0}$.

Definición 53 (Expresiones)**.**

$$\begin{aligned} t, t' \in \mathit{Expr} \ ::= \ & \bar{n} \mid \lambda t \mid t\ \bar{n} \mid \diamond \mid \mathit{let}\ t\ \mathit{in}\ t' \\ & \mid (t, t') \mid \mathit{fst}\ \bar{n} \mid \mathit{snd}\ \bar{n} \\ & \mid \lfloor n \rfloor \mid t \odot t' \mid \mathit{ifz}\ t\ \mathit{then}\ t'\ \mathit{else}\ t'' \end{aligned}$$

Es importante mencionar que la restricción en la expresión **let** la cual solo permite introducir una variable, no es una limitación sobre el poder expresivo; por ejemplo podemos definir la función que determina si el valor de la variable n es par (e impar cambiando *fst* por *snd* en el cuerpo del ultimo *let*) donde interpretamos a la constante $\underline{0}$ como verdadero y $\underline{1}$ como falso en el valor final devuelto.

```
1  let λ λ (ifz 0̄ then 0
2           else let 1̄ − 1 in
3           (snd 2̄) 0̄
4   ) in
5  let λ λ (ifz 0̄ then 1
6           else let 1̄ − 1 in
7           (fst 2̄) 0̄
8   ) in
9  let λ
10     (let 2̄ 1̄ in
11          ((fst 1̄) 0̄, (snd 1̄) 0̄)
12  ) in
13 let n in
14 let (4̄, 3̄) in
15 let (3̄ 1̄)
16 in (fst 0̄) 2̄
```

Usando índices De Bruijn

```
1  let even = λeo. λm. (ifz m then 0
2           else let m' = m − 1 in
3           (snd eo) m'
4   ) in
5  let odd = λeo. λm. (ifz m then 1
6           else let m' = m − 1 in
7           (fst eo) m'
8   ) in
9  let eoRec = λeo.
10      (let eor = eoRec eo in

11
        ((fst eo) eor, (snd eo) eor)
12  ) in
13  let n = n in
14  let eo = (even, odd) in
15  let evenodd = evenoddRec eo
16  in (fst evenodd) n
```

Usando variables con nombre

A diferencia de Launchbury y Sestoft quienes definen un lenguaje sin tipos, nosotros consideramos solo expresiones bien tipadas.

Definición 54 (Tipos)**.**

$$\theta, \theta' \in \mathit{Type} \ ::= \ \mathbf{unit} \mid \mathbf{int} \mid \theta \to \theta' \mid \theta \times \theta'$$

El tipo de las variables se define mediante los contextos, los cuales son listas de tipos ya que utilizamos índices de De Bruijn. Para agregar un tipo nuevo al contexto utilizamos $\_ :: \_$, los contextos siempre crecen a izquierda, de esta manera podemos mantener una conexión entre cualquier índice y la posición de su tipo en el contexto.

Definición 55 (Contextos)**.**

$$\pi \in \mathrm{Ctx} \ ::= \ [] \mid \theta :: \pi$$

Para definir las reglas de tipado definimos la función de búsqueda sobre cualquier contexto dada una posición, denotada por $\_ \cdot \_$, la cual devuelve el conjunto vacío si el índice no esta definido y un singleton cuando lo esta. Un juicio de tipado $\pi \vdash t : \theta$ denotará que t tiene tipo θ bajo el contexto π. Un juicio es válido si uno puede construir una derivación utilizando las siguientes reglas.

Definición 56 (Reglas de tipado)**.**

$$\textsc{Abs}\ \frac{\theta :: \pi \vdash t : \theta'}{\pi \vdash \lambda t : \theta \to \theta'} \qquad \textsc{App}\ \frac{\pi \vdash t : \theta \to \theta'}{\pi \vdash t\ \overline{n} : \theta'}\ \pi \cdot n = \{\theta\}$$

$$\textsc{Var}\ \frac{}{\pi \vdash \overline{n} : \theta}\ \pi \cdot n = \{\theta\} \qquad \textsc{Unit}\ \frac{}{\pi \vdash \diamond : \textbf{\textit{unit}}}$$

$$\textsc{Let}\ \frac{\theta :: \pi \vdash t : \theta \qquad \theta :: \pi \vdash t' : \theta'}{\pi \vdash \mathit{let}\ t\ \mathit{in}\ t' : \theta'}$$

$$\textsc{Pair}\ \frac{\pi \vdash t : \theta \qquad \pi \vdash t' : \theta'}{\pi \vdash (t, t') : \theta \times \theta'}$$

$$\textsc{Fst}\ \frac{\pi \vdash \overline{n} : \theta \times \theta'}{\pi \vdash \mathit{fst}\ \overline{n} : \theta} \qquad \textsc{Snd}\ \frac{\pi \vdash \overline{n} : \theta \times \theta'}{\pi \vdash \mathit{snd}\ \overline{n} : \theta'}$$

$$\textsc{Int}\ \frac{}{\pi \vdash \lfloor m \rfloor : \textbf{\textit{int}}} \qquad \textsc{BinOp}\ \frac{\pi \vdash t : \textbf{\textit{int}} \qquad \pi \vdash t' : \textbf{\textit{int}}}{\pi \vdash t \odot t' : \textbf{\textit{int}}}$$

$$\textsc{Ifz}\ \frac{\pi \vdash t : \textbf{\textit{int}} \qquad \pi \vdash t' : \theta \qquad \pi \vdash t'' : \theta}{\pi \vdash \mathit{ifz}\ t\ \mathit{then}\ t'\ \mathit{else}\ t'' : \theta}$$

Semántica denotacional

Nuestro lenguaje posee recursión por lo tanto el modelo matemático deberá ser lo suficientemente rico como para poder abordar esta característica, por lo tanto utilizaremos los dominios que introdujimos en el capítulo 2; recordemos que los dominios son conjuntos parcialmente ordenados que nos garantizan la existencia de un menor punto fijo para funciones continuas, exactamente esta característica es la que nos permitirá dar semántica a nuestra expresión *let* recursiva. La semántica de cada tipo estará dada por un dominio; notar que alcanza con fijar la denotación de enteros y unit, pues los otros tipos obtienen su semántica utilizando la estructura cartesiana de la categoría. Además caracterizaremos como *tipos básicos* a aquellos cuya semántica esté determinada por un dominio con orden discreto.

Definición 57 (Semántica de tipos)**.**

$$[\![\, \textbf{\textit{unit}} \,]\!]^{\nu} = \{\diamond\} \qquad [\![\, \theta \to \theta' \,]\!]^{\nu} = [\![\, \theta \,]\!] \to [\![\, \theta' \,]\!]$$
$$[\![\, \textbf{\textit{int}} \,]\!]^{\nu} = \mathbb{Z} \qquad [\![\, \theta \times \theta' \,]\!]^{\nu} = [\![\, \theta \,]\!] \times [\![\, \theta' \,]\!]$$
$$[\![\, \theta \,]\!] = [\![\, \theta \,]\!]^{\nu}_{\perp}$$

La semántica de los contextos está dada por productos; decimos que la tupla $\rho \in [\![\, \pi \,]\!]$ es un *entorno*.

Definición 58 (Semántica de contextos)**.**

$$[\![\, [] \,]\!] = \{\diamond\} \qquad [\![\, \theta :: \pi \,]\!] = [\![\, \theta \,]\!] \times [\![\, \pi \,]\!]$$

Damos significado solamente a expresiones bien tipadas, lo que se conoce según Reynolds [39] como *semántica intrínseca*. Por recursión en las derivaciones asignamos a cada derivación de un juicio de tipado con conclusión $\pi \vdash t : \theta$ una función continua de $[\![\, \pi \,]\!]$ a $[\![\, \theta \,]\!]$. En la formalización utilizamos una definición puramente categórica de las ecuaciones semánticas que nos asegura la continuidad de manera directa; usamos combinadores categóricos como la composición en la categoría base, productos, exponenciales, etc. Además, hacemos uso de la categoría de Kleisli asociada a la mónada de promoción; los objetos son los mismos que los de la categoría base, pero $\mathsf{Hom}_K(A, B) = \mathsf{Hom}(A, FB)$ donde el functor F es en este caso la promoción, luego la identidad $\mathsf{id}_A : A \to A_\perp$ es exactamente $\iota_\uparrow$ y la composición se define como $g \circ_k f = g_{\perp\!\!\perp} \circ f$. Sin embargo, en este contexto si bien

preferimos una definición más categórica que la de los capítulos anteriores, nos permitimos explicitar los argumentos a fines de mejorar la legibilidad. Por lo tanto, para la definición de cada ecuación semántica utilizamos algunas de las funciones continuas asociadas a la estructura cartesiana de la categoría y definimos nuevas funciones continuas: sean A, B y C predominios,

- Respecto al producto tenemos las proyecciones $\pi_1 : A \times B \to A$, $\pi_2 : A \times B \to B$ y el producto de funciones continuas $\langle f , g \rangle : A \to B \times C$ con $f : A \to B$ y $g : A \to C$.
- Como ya mencionamos, dadas dos funciones continuas $f : A \to B$ y $g : B \to C$, denotamos a la composición como $g \circ f : A \to C$ y además si tenemos $f : A \to B_\perp$ y $g : B \to C_\perp$ podemos utilizar la promoción de funciones para definir la composición $g \circ_k f : A \to C_\perp$ como $g \circ_k f = g_{\perp\!\!\perp} \circ f$.
- Para dar significado a la abstracción definimos la siguiente función para cualquier $f : A \times B \to C$,

$$\begin{aligned} &\langle f \rangle \quad : B \to (A \to C)_\perp \\ &\langle f \rangle \;\; b = \iota_\uparrow(a \mapsto f\,(a, b)) \;\; . \end{aligned}$$

- Para el caso de la aplicación definimos una función continua con alguna similitud a la composición para cualquieras funciones continuas $f : A \to B$ y $g : A \to (B \to C_\perp)_\perp$,

$$\begin{aligned} &g \circ_{app} f \quad : A \to C_\perp \\ &g \circ_{app} f \;\; a = (h \mapsto h \circ (f\,a)) \circ_k (g\,a) \;\; . \end{aligned}$$

- Dada una operación binaria $\odot : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$ y dos funciones continuas $f : A \to \mathbb{Z}_\perp$ y $g : A \to \mathbb{Z}_\perp$ definimos

$$\begin{aligned} &f \circ_\odot g \quad : A \to \mathbb{Z}_\perp \\ &f \circ_\odot g \;\; a = (i \mapsto (i' \mapsto \iota_\uparrow(i \odot i'))_{\perp\!\!\perp}(f\,a))_{\perp\!\!\perp}(g\,a) \;\; . \end{aligned}$$

- Dadas funciones continuas $f : A \to \mathbb{Z}_\perp$, $g : A \to B_\perp$ y $h : A \to B_\perp$ podemos definir además un operador condicional $\mathbf{ifz}\,(f)\,(g)\,(h) : A \to B_\perp$ tal que

$$(\mathbf{ifz}\,(f)\,(g)\,(h))\,a = \begin{cases} g\,a & \text{si } f\,a = \iota_\uparrow 0 \\ h\,a & \text{si } f\,a \neq \iota_\uparrow 0 \\ \perp & \text{si } f\,a = \perp \end{cases} \;\; .$$

- Definir la función continua para dar semántica a la expresión *let* implica tomar un punto fijo y aplicarlo, para esto definimos dos funciones continuas. La primera es el operador de punto fijo de manera que dada una función continua $f : A_\perp \times B \to A_\perp$ donde denotamos a $\langle f \rangle_n : B \to A_\perp$ como el n-ésimo elemento de la cadena formada por f,

$$\begin{aligned} &\mathbf{Fix}(f) : B \to A_\perp && \langle f \rangle_0 \quad b = \perp \\ &\mathbf{Fix}(f) = \bigsqcup_{n=0}^{\infty} \langle f \rangle_n && \langle f \rangle_{n+1} \quad b = f\,(\langle f \rangle_n b\,,\, b)\ . \end{aligned}$$

  La segunda definición toma dos funciones continuas $f : B \to A_\perp$ y $g : A_\perp \times B \to C_\perp$ y define

$$\begin{aligned} &\mathbf{let}\,(f)\,(g) \quad : B \to C_\perp \\ &\mathbf{let}\,(f)\,(g) \quad b = g\,(f\,b, b)\ . \end{aligned}$$

- Finalmente, dado algún elemento c denotaremos a la función constante como $\_ \mapsto c$.

Por conveniencia omitimos la derivación y escribimos el contexto y el tipo como sub-índices. Además, extendemos las tuplas hacia la izquierda asumiendo que (d, ρ) es igual que su versión "aplastada" y escribimos $\rho \swarrow n$ para denotar la n-ésima proyección desde la tupla ρ; asumiendo que $n < |\rho|$.

Definición 59 (Semántica de Expresiones)**.**

$$\begin{aligned} [\![\diamond]\!]_{\pi, \mathit{unit}} &= \_ \mapsto \iota_\uparrow \diamond \\ [\![\lfloor m \rfloor]\!]_{\pi, \mathit{int}} &= \_ \mapsto \iota_\uparrow m \\ [\![\lambda\, t]\!]_{\pi, \theta \to \theta'} &= \langle [\![t]\!]_{\theta :: \pi, \theta'} \rangle \\ [\![\overline{n}]\!]_{\pi, \theta} &= \rho \mapsto \rho \swarrow n \\ [\![t\ \overline{n}]\!]_{\pi, \theta'} &= [\![t]\!]_{\pi, \theta \to \theta'} \circ_{\mathrm{app}} [\![\overline{n}]\!]_{\pi, \theta} \\ [\![(t, t')]\!]_{\pi, \theta \times \theta'} &= \iota_\uparrow \circ \langle [\![t]\!]_{\pi, \theta}\,,\, [\![t']\!]_{\pi, \theta'} \rangle \\ [\![\mathit{fst}\ n]\!]_{\pi, \theta} &= \pi_1 \circ_k [\![\overline{n}]\!]_{\pi, \theta \times \theta'} \\ [\![\mathit{snd}\ n]\!]_{\pi, \theta'} &= \pi_2 \circ_k [\![\overline{n}]\!]_{\pi, \theta \times \theta'} \\ [\![\mathit{let}\ t\ \mathit{in}\ t']\!]_{\pi, \theta'} &= \boldsymbol{let}\,(\boldsymbol{Fix}([\![t]\!]_{\theta :: \pi, \theta}))\,([\![t']\!]_{\theta :: \pi, \theta'}) \\ [\![t \odot t']\!]_{\pi, \mathit{int}} &= [\![t]\!]_{\pi, \mathit{int}} \circ_\odot [\![t']\!]_{\pi, \mathit{int}} \\ [\![\mathit{ifz}\ t\ \mathit{then}\ t'\ \mathit{else}\ t'']\!]_{\pi, \theta} &= \boldsymbol{ifz}\,([\![t]\!]_{\pi, \mathit{int}})\,([\![t']\!]_{\pi, \theta})\,([\![t'']\!]_{\pi, \theta}) \end{aligned}$$

5.2 LENGUAJE DE BAJO NIVEL

En esta sección presentamos el lenguaje objeto en el cual estarán representados los programas resultantes de compilar los programas escritos en el lenguaje de alto nivel. La representación del lenguaje la realizamos mediante la introducción de una máquina abstracta; tomamos la máquina básica definida por Sestoft [46] y presentada por Rodríguez [41, Capítulo 7] en su tesis doctoral y la extendemos con productos, operadores binarios aritméticos y operador de recursión. La máquina abstracta original definida por Sestoft es producto de la derivación de la semántica operacional descripta por Launchbury [25] (presentada en el capítulo 3, definición 6) y resulta ser una variante de la máquina abstracta de Krivine [1]; la cual implementa la estrategia de evaluación call-by-name.

Una configuración en la máquina de Sestoft es una cuádrupla (Γ, i, η, s) compuesta por el *heap* Γ, la *instrucción* i, el *entorno* η, y el *stack* s; una configuración en la semántica operacional de Launchbury está compuesta por el heap y la expresión, con la finalidad de transformar el estilo arbóreo de evaluación en una secuencia de ejecución "paso a paso", agregamos el stack s. Además, para capturar de forma operacional la substitución realizada durante la aplicación introducimos el entorno η. En relación con la máquina de Krivine[2] los entornos y stacks en la máquina abstracta definida por Sestoft serán listas de *punteros*, además el stack podrá contener *marcadores* que permiten realizar transiciones particulares una vez que computamos a un valor. Un heap será un mapa de punteros a pares de instrucción y entorno que llamaremos *clausuras*.

1 descripta en una publicación por [24] más de veinticinco años después de su definición.
2 donde una configuración es una terna compuesta por una instrucción, un entorno y un stack que contienen pares de instrucción y entorno

Definición 60 (Componentes)**.**

$$
\begin{array}{lcl}
iv \in \mathit{Val} & ::= & \mathtt{Grab} \triangleright \mathtt{i} \mid \mathtt{Unit} \mid \mathtt{Pair}\,(\mathtt{i},\mathtt{i}') \mid \underline{n} \\
\mathtt{i},\mathtt{i}' \in \mathit{Code} & ::= & iv \mid \mathtt{Access}\,n \mid \mathtt{BOp}_{\odot}\,\mathtt{i}\,\mathtt{i}' \\
& \mid & \mathtt{Push}\,n \triangleright \mathtt{i} \mid \mathtt{Let}\,\mathtt{i} \triangleright \mathtt{i}' \\
& \mid & \mathtt{Snd}\,n \mid \mathtt{Fst}\,n \mid \mathtt{IfZ}\,\mathtt{i}\,\mathtt{i}'\,\mathtt{i}'' \\
p, q \in \mathit{Pointer} & & \\
m \in \mathit{Marker} & ::= & \sharp p \mid \sharp_1 p \mid \sharp_2 p \mid ?p \mid \odot p \\
k \in \mathit{Cont} & ::= & \mathtt{IfZ}\,\square\,(\mathtt{i},\mathtt{i}') \mid \mathtt{BOp}_{\odot}\,\square\,\mathtt{i} \mid \mathtt{BOp}_{\odot}\,\underline{n}\,\square \\
\alpha \in \mathit{MClos} & ::= & (\mathtt{i}, \eta) \\
\eta \in \mathit{MEnv} & ::= & [] \mid p :: \eta \\
s \in \mathit{Stack} & ::= & [] \mid p :: s \mid m :: s \mid \pi_1 :: s \mid \pi_2 :: s \\
\Gamma, \Delta \in \mathit{Heap} & ::= & [] \mid (p, \alpha) :: \Gamma \mid (p, (k, \eta)) :: \Gamma \\
w \in \mathit{Conf} & ::= & (\Gamma, \mathtt{i}, \eta, s)
\end{array}
$$

Usaremos de manera indistinta el isomorfismo entre funciones con dominios finitos y mapas finitos pensados como lista de tuplas, por ejemplo denotaremos el heap $(p, \alpha) :: []$ como $\{p \mapsto \alpha\}$. Un puntero p estará en un entorno η si existe una posición n tal que $\eta \cdot n = p$. Los punteros de una clausura son aquellos que están en el entorno y los del heap serán los que se encuentran en su dominio sumados a los incluidos en la clausura que aparezca en su imagen.

Definición 61 (Punteros)**.**

$$
\begin{array}{llll}
ptr(\eta) & = \bigcup_{n \in \mathbb{N}} \eta \cdot n & ptr(s) & = \bigcup_{n \in \mathbb{N}} s \cdot n \\
ptr((\mathtt{i}, \eta)) & = ptr(\eta) & ptr((k, \eta)) & = ptr(\eta)
\end{array}
$$

$$
ptr(\Gamma) = dom(\Gamma) \cup \bigcup_{p \in dom(\Gamma)} ptr(\Gamma\, p)
$$

Denotaremos la actualización del puntero p con γ en el heap Γ como $\Gamma[p \mapsto \gamma]$, donde γ puede ser una clausura o un par (k, η):

$$
\Gamma[p \mapsto \gamma]\, q = \begin{cases} \gamma & \text{if } p = q \\ \Gamma\, q & \text{otherwise} \end{cases}
$$

Las reglas de transición que describen la semántica de la máquina están dadas por los siguientes cuadros: (ı) en 5.1 presentamos las reglas tomadas

directamente de la máquina de Sestoft y nuestra regla de recursión, (II) en 5.2 describimos las reglas de actualización del heap, (III) las reglas para evaluar operaciones binarias se describen en 5.3, (IV) en 5.4 damos las reglas sobre la ejecución de la instrucción condicional, (V) finalmente en 5.5 describimos las reglas respecto de los pares y las proyecciones. Claramente es posible agregar reglas de transición extra que nos permitan optimizar el heap.

Las primeras cuatro reglas presentadas en 5.1 están tomadas directamente de la máquina de Sestoft. Una instrucción `Grab` $\triangleright$ i es ejecutada moviendo el puntero desde el stack hacía el entorno y continuando la ejecución con i. El stack acumula los punteros hacía las instrucciones que forman los argumentos, luego pasar un puntero del stack hacía el entorno significa que el cuerpo de la abstracción puede acceder a sus argumentos. La instrucción `Access` n desreferencia el puntero $\eta \cdot n$ y actualiza la clausura actual por la desreferenciada; además la regla agrega un marcador al stack para forzar la actualización del heap una vez que se alcance un valor, es decir si alcanzamos una configuración donde la instrucción de la clausura es un *Val*.

La instrucción `Push` n $\triangleright$ i agrega al stack el n-ésimo puntero del entorno. Ese puntero fue creado y agregado al entorno por la instrucción `Let` i $\triangleright$ i', la cual extiende el heap agregando un nuevo puntero p y asociándolo con la clausura (i, η), donde η es el entorno actual. La garantía de que el puntero p es fresco está dada por la condición $p \notin \mathtt{ptr}(\Gamma) \cup \mathtt{ptr}(\eta) \cup \mathtt{ptr}(s)$; para evitar el no-determinismo en la creación del nuevo puntero podemos definir al conjunto de punteros como $\mathit{Pointer} = \mathbb{N}$ y la función de creación elige el menor natural no utilizado aun.

La ultima regla de transición es la relacionada con la recursión, en cada paso de recursión creamos un puntero fresco al que vamos a asociarle la clausura original, además colocamos para evaluar la clausura compuesta por la instrucción relacionada con la recursión y el entorno al que agregamos el nuevo puntero. Notar que por cada paso recursivo que dé la máquina durante la evaluación vamos a estar creando un nuevo puntero fresco, más aun cada vez que creemos este nuevo puntero, el puntero que nos condujo al nuevo paso recursivo queda inalcanzable.

Las reglas de 5.2 describen la actualización del heap. La primera regla se corresponde con la actualización del heap porque alcanzamos un valor y en el tope del stack tenemos un marcador que nos indica que puntero actualizar. Notar que podemos tener cuatro tipos distintos de valores, los

$$
\begin{array}{lll}
(\Gamma, \texttt{Grab} \triangleright i, \eta, p :: s) & \longmapsto (\Gamma, i, p :: \eta, s) & (\textsc{grab}) \\
(\Gamma, \texttt{Access}\ n, \eta, s) & \longmapsto (\Gamma, i', \eta', \sharp p :: s) & \\
 & \quad \text{cuando } n < |\eta|,\ p = \eta \cdot n \ y & \\
 & \quad (i', \eta') = \Gamma\, p & (\textsc{acc}) \\
(\Gamma, \texttt{Push}\ n \triangleright i, \eta, s) & \longmapsto (\Gamma, i, \eta, p :: s) & \\
 & \quad \text{cuando } p = \eta \cdot n & (\textsc{push}) \\
(\Gamma, \texttt{Let}\ i \triangleright i', \eta, s) & \longmapsto (\Gamma[p \mapsto (i, p :: \eta)], i', p :: \eta, s) & \\
 & \quad \text{cuando } p \notin \mathrm{ptr}(\Gamma) \cup \mathrm{ptr}(\eta) \cup \mathrm{ptr}(s) & (\textsc{let}) \\
(\Gamma, \texttt{Rec}\ i, \eta, s) & \longmapsto (\Gamma[p \mapsto (\texttt{Rec}\ i, \eta)], i, p :: \eta, s) & \\
 & \quad \text{cuando } p \notin \mathrm{ptr}(\Gamma) \cup \mathrm{ptr}(\eta) \cup \mathrm{ptr}(s) & (\textsc{rec})
\end{array}
$$

Cuadro 5.1: **Transiciones de Sestoft + Recursión**.

básicos `Unit` y $\underline{n}$ junto con `Pair` (i, i') y `Grab` $\triangleright i$. Las dos reglas siguientes son similares y actualizan una componente del par.

$$
\begin{array}{lll}
(\Gamma, iv, \eta, \sharp p :: s) & \longmapsto (\Gamma[p \mapsto (iv, \eta)], iv, \eta, s) & (\textsc{upd}) \\
(\Gamma, iv, \eta, \sharp_1 p :: s) & \longmapsto (\Gamma[p \mapsto (\texttt{Pair}\ (iv, i'), \eta')], iv, \eta, s) & \\
 & \quad \text{cuando } (\texttt{Pair}\ (i, i'), \eta') = \Gamma\, p & (\textsc{upd-1}) \\
(\Gamma, iv, \eta, \sharp_2 p :: s) & \longmapsto (\Gamma[p \mapsto (\texttt{Pair}\ (i, iv), \eta')], iv, \eta, s) & \\
 & \quad \text{cuando } (\texttt{Pair}\ (i, i'), \eta') = \Gamma\, p & (\textsc{upd-2})
\end{array}
$$

Cuadro 5.2: **Transiciones de actualización**.

Las reglas de transición introducidas en 5.3 y 5.4 se corresponden con la evaluación del operador binario (estricto) y la instrucción condicional, respectivamente. Las primeras transiciones de cada cuadro crean punteros, como en el caso de la instrucción let, pero son asociados a otra clase de clausura. La computación de un operador binario crea un *frame* (similar a la propuesta de Selinger [45]) que acumula los valores ya computados y señala con $\square$ el lugar del operando que esta siendo computado. La ejecución de la instrucción condicional asocia el nuevo puntero con el par de instrucciones que continuarán con la ejecución dependiendo de la evalua-

ción de la condición. Los marcadores $\odot p$ y $?p$ apuntan a la continuación que contiene las instrucciones para continuar la evaluación.

$$
\begin{array}{lll}
(\Gamma, \texttt{BOp}_{\odot}\ i\ i', \eta, s) \longmapsto (\Gamma[p \mapsto (\texttt{BOp}_{\odot} \square\, i', \eta)], i, \eta, \odot p :: s) & & \\
\qquad \text{cuando } p \notin \mathrm{ptr}(\Gamma) \cup \mathrm{ptr}(\eta) \cup \mathrm{ptr}(s) & & \text{(BOP)} \\
(\Gamma, \underline{n}, \eta', \odot p :: s) \longmapsto (\Gamma[p \mapsto (\texttt{BOp}_{\odot}\ \underline{n} \square, \eta)], i', \eta, \odot p :: s) & & \\
\qquad \text{cuando } \Gamma p = (\texttt{BOp}_{\odot} \square\, i', \eta) & & \text{(BOP-L)} \\
(\Gamma, \underline{m}, \eta', \odot p :: s) \longmapsto (\Gamma, \underline{n \odot m}, \eta, s) & & \\
\qquad \text{cuando } \Gamma p = (\texttt{BOp}_{\odot}\ \underline{n} \square, \eta) & & \text{(BOP-R)}
\end{array}
$$

Cuadro 5.3: **Transiciones de la operación aritmética**.

$$
\begin{array}{lll}
(\Gamma, \texttt{IfZ}\ i\ i'\ i'', \eta, s) \longmapsto (\Gamma[p \mapsto (\texttt{IfZ} \square\, (i', i'')), \eta)], i, \eta, ?p :: s) & & \\
\qquad \text{cuando } p \notin \mathrm{ptr}(\Gamma) \cup \mathrm{ptr}(\eta) \cup \mathrm{ptr}(s) & & \text{(IFZ)} \\
(\Gamma, \underline{0}, \eta, ?p :: s) \longmapsto (\Gamma, i', \eta', s) & & \\
\qquad \text{cuando } \Gamma p = (\texttt{IfZ} \square\, (i', i''), \eta') & & \text{(IFZ-Z)} \\
(\Gamma, \underline{n+1}, \eta, ?p :: s) \longmapsto (\Gamma, i'', \eta', s) & & \\
\qquad \text{cuando } \Gamma p = (\texttt{IfZ} \square\, (i', i''), \eta') & & \text{(IFZ-NZ)}
\end{array}
$$

Cuadro 5.4: **Transiciones del condicional**.

Las ultimas cuatro transiciones del cuadro5.5 involucran la ejecución de las proyecciones: para proyectar una componente, por ejemplo la que tiene como instrucción a `Fst` n, primero tenemos que evaluar el código asociado con el puntero dado por $\eta \cdot n$; además agregamos al stack los marcadores para actualizar el puntero, proyectar la primera componente (en el caso de que el código compute a un par) y finalmente actualizar la primera componente del par. Las transiciones que efectivamente realizan la proyección de la componente son aquellas que tiene el marcador π_i.

Denotamos a la clausura reflexiva y transitiva de las reglas de transición de la máquina como $\longmapsto^*$; escribimos $(\Gamma, i, \eta, s) \longmapsto^* (\Gamma', i', \eta', s')$ para describir que la máquina va de la configuración (Γ, i, η, s) a la configuración (Γ', i', η', s') en alguna cantidad finita de pasos. Además denotamos

$$
\begin{array}{lll}
(\Gamma,\ \mathtt{Fst}\ n,\ \eta,\ s) & \longmapsto (\Gamma,\ i,\ \eta',\ \sharp p :: \pi_1 :: \sharp_1 p :: s) & \\
& \quad \text{cuando } p = \eta \cdot n, \Gamma\, p = (i, \eta') & \text{(FST)} \\
(\Gamma,\ \mathtt{Snd}\ n,\ \eta,\ s) & \longmapsto (\Gamma,\ i,\ \eta',\ \sharp p :: \pi_2 :: \sharp_2 p :: s) & \\
& \quad \text{cuando } p = \eta \cdot n, \Gamma\, p = (i, \eta') & \text{(SND)} \\
(\Gamma,\ \mathtt{Pair}\ (i, i'),\ \eta,\ \pi_1 :: s) & \longmapsto (\Gamma,\ i,\ \eta,\ s) & \text{(PAIR-1)} \\
(\Gamma,\ \mathtt{Pair}\ (i, i'),\ \eta,\ \pi_2 :: s) & \longmapsto (\Gamma,\ i',\ \eta,\ s) & \text{(PAIR-2)}
\end{array}
$$

Cuadro 5.5: **Transiciones del producto**.

a la computación que puede dar una cantidad arbitraria de pasos como $(\Gamma,\ i,\ \eta,\ s) \longmapsto^{\infty}$.

COMPILACIÓN En contraste con la semántica denotacional, para la compilación no usamos ninguna información sobre el tipo de la expresión, por lo tanto podemos compilar cualquier expresión. Sin embargo la prueba de corrección solo será para expresiones bien tipadas y la prueba está basada en las reglas de derivación de los juicios de tipado.

Definición 62 (Compilador)**.**

$$
\begin{array}{llll}
(\!|\, \lambda\, t \,|\!) & = \mathtt{Grab} \triangleright (\!|\, t \,|\!) & (\!|\, \overline{n} \,|\!) & = \mathtt{Access}\ n \\
(\!|\, t\ \overline{n} \,|\!) & = \mathtt{Push}\ n \triangleright (\!|\, t \,|\!) & (\!|\, \diamond \,|\!) & = \mathtt{Unit} \\
(\!|\, \mathit{let}\ t\ \mathit{in}\ t' \,|\!) & = \mathtt{Let}\ (\mathtt{Rec}\ (\!|\, t \,|\!)) \triangleright (\!|\, t' \,|\!) & (\!|\, \lfloor m \rfloor \,|\!) & = \underline{m} \\
(\!|\, (t,\ t') \,|\!) & = \mathtt{Pair}\ ((\!|\, t \,|\!), (\!|\, t' \,|\!)) & (\!|\, \mathit{fst}\ \overline{n} \,|\!) & = \mathtt{Fst}\ n \\
(\!|\, t \odot t' \,|\!) & = \mathtt{BOp}_{\odot}\ (\!|\, t \,|\!)\ (\!|\, t' \,|\!) & (\!|\, \mathit{snd}\ \overline{n} \,|\!) & = \mathtt{Snd}\ n \\
\multicolumn{4}{l}{(\!|\, \mathit{ifz}\ t\ \mathit{then}\ t'\ \mathit{else}\ t'' \,|\!) = \mathtt{IfZ}\ (\!|\, t \,|\!)\ (\!|\, t' \,|\!)\ (\!|\, t'' \,|\!)}
\end{array}
$$

EJEMPLO Consideremos el siguiente ejemplo con el fin de analizar el comportamiento de la máquina abstracta enfocándonos especialmente en el "sharing". La expresión (a) está construida utilizando la sintaxis abstracta que presentamos, es decir utilizando índices de De Bruijn, la (b) es la representación utilizando nombres de variables y el código (c) es el resultado de la compilación de (a). Notar que estamos eligiendo $\odot$, la operación binaria, como $+$.

```
1  let ⌊1⌋ + ⌊2⌋
2  in 0̄ + 0̄
```
(a) De Bruijn

```
1  let x = ⌊1⌋ + ⌊2⌋
2  in x + x
```
(b) Nombres

```
1  Let (Rec (BOp+ 1 2))
2  ▷ (BOp+ (Access 0) (Access 0))
```
(c) Compilado

La ejecución del ejemplo comienza en el paso 1 con el heap, entorno y stack, todos vacíos y finaliza en el paso 13 cuando llegamos a la configuración donde el código de la clausura es un valor y el stack esta vacío. En particular para el ejemplo escribimos Γ_p para denotar al heap que contiene a p referenciando a alguna clausura, además cada vez que actualicemos un puntero ya existente lo escribimos como $\hat{p}$. Las transiciones de 1 a 3 (LET y BOP) generan los primeros punteros p y q, donde p será el puntero en el cual observaremos el "sharing" y q mantendrá el frame para evaluar la operación binaria (en este caso el $+$); lo mismo ocurre con r durante la transición de 5 a 6 (BOP). Entre las transiciones 1 y 6 extendemos el heap vacío progresivamente de la siguiente manera

$$
\begin{aligned}
\Gamma_p = \ & \{p \mapsto (\texttt{Rec}\,(\texttt{BOp}_+\ \underline{1}\ \underline{2}), [])\} \\
\Gamma_{p,q} = \ & \{p \mapsto (\texttt{Rec}\,(\texttt{BOp}_+\ \underline{1}\ \underline{2}), []) \\
& , q \mapsto (\texttt{BOp}_+\ \square\ (\texttt{Access}\ 0), [p])\} \\
\Gamma_{p,q,r} = \ & \{p \mapsto (\texttt{Rec}\,(\texttt{BOp}_+\ \underline{1}\ \underline{2}), []) \\
& , q \mapsto (\texttt{BOp}_+\ \square\ (\texttt{Access}\ 0), [p]) \\
& , r \mapsto (\texttt{BOp}_+\ \square\ \underline{2}, [])\}
\end{aligned}
$$

Por simplicidad en la transición de 4 a 5 (REC) obviamos la creación del puntero referido a la recursión; notar que deberíamos crear un puntero nuevo y asociarle la clausura del paso 4 para luego en el paso 5 extender el entorno con ese puntero. Claramente para computar en este ejemplo el código $\texttt{BOp}_+\ \underline{1}\ \underline{2}$ nos es irrelevante el entorno. Entre las transiciones 6 a 8 (BOP-L y BOP-R) se computa el valor de la operación binaria, en consecuencia actualizamos las componentes del frame referenciado por r

$$
\begin{aligned}
\Gamma_{p,q,\hat{r}} = \ & \{p \mapsto (\texttt{Rec}\,(\texttt{BOp}_+\ \underline{1}\ \underline{2}), []) \\
& , q \mapsto (\texttt{BOp}_+\ \square\ (\texttt{Access}\ 0), [p]) \\
& , r \mapsto (\texttt{BOp}_+\ \underline{1}\ \square, [])\}
\end{aligned}
$$

En la transición de 8 a 9 (UPD) podemos notar que la configuración tiene un valor como código y un marcador de actualización en el tope del stack, por lo tanto en esta transición se realiza la actualización del puntero

$$\begin{aligned}\Gamma_{\hat{p},q,\hat{r}} = \ & \{p \mapsto (\underline{3}, [])\\ & , q \mapsto (\mathtt{BOp}_+ \ \square \ (\mathtt{Access}\ 0), [p])\\ & , r \mapsto (\mathtt{BOp}_+ \ \underline{1} \ \square, [])\}\end{aligned}$$

A partir del paso 9 y hasta el 12 se da la evaluación del segundo operando del operador binario ubicado en el cuerpo del `Let`, en particular podemos notar que en la transición de 10 a 11 (ACC) al hacer el access obtenemos el valor actualizado en lugar de $\mathtt{BOp}_+ \ \underline{1} \ \underline{2}$. Además la ultima actualización del heap se realiza en la transición de 9 a 10

$$\begin{aligned}\Gamma_{\hat{p},\hat{q},\hat{r}} = \ & \{p \mapsto (\underline{3}, [])\\ & , q \mapsto (\mathtt{BOp}_+ \ \underline{3} \ \square, [p])\\ & , r \mapsto (\mathtt{BOp}_+ \ \underline{1} \ \square, [])\}\end{aligned}$$

Finalmente en la ultima transición de 12 a 13 (BOP-R) combinamos los dos valores, uno el valor ubicado en la clausura de la configuración y el segundo desde el puntero q; ya que en el tope del stack tenemos el marcador $+q$ que nos especifica donde ir a buscar el frame que completa la evaluación de la operación binaria.

5.3 CORRECCIÓN DEL COMPILADOR

En esta sección vamos a probar la corrección del compilador con respecto a una semántica denotacional. Dado que nuestro lenguaje posee recursión la corrección del compilador involucra dos aspectos. Por un lado la compilación de una expresión divergente debe producir un código tal que la máquina abstracta "se cuelgue", es decir la computación pueda dar una cantidad arbitraria de pasos. Por otro lado, si la interpretación denotacional de una expresión es una constante de algún tipo básico entonces el código generado por el compilador debe ser tal que la máquina termina y además la configuración final contiene a la representación de la constante como instrucción.

	Heap	*Code*	*MEnv*	*Stack*	*Regla*
1	$\{\,\}$	$\texttt{Let}(\texttt{Rec}\,(\texttt{BOp}_+\ \underline{1}\ \underline{2}))$	$[]$	$[]$	
		$\triangleright\ \texttt{BOp}_+\ (\texttt{Access}\ 0)\ (\texttt{Access}\ 0)$			LET
2	Γ_p	$\texttt{BOp}_+\ (\texttt{Access}\ 0)\ (\texttt{Access}\ 0)$	$[p]$	$[]$	BOP
3	$\Gamma_{p,q}$	$\texttt{Access}\ 0$	$[p]$	$[+q]$	ACC
4	$\Gamma_{p,q}$	$\texttt{Rec}\,(\texttt{BOp}_+\ \underline{1}\ \underline{2})$	$[]$	$[\sharp p, +q]$	REC
5	$\Gamma_{p,q}$	$\texttt{BOp}_+\ \underline{1}\ \underline{2}$	$[]$	$[\sharp p, +q]$	BOP
6	$\Gamma_{p,q,r}$	$\underline{1}$	$[]$	$[+r, \sharp p, +q]$	BOP-L
7	$\Gamma_{p,q,\hat{r}}$	$\underline{2}$	$[]$	$[+r, \sharp p, +q]$	BOP-R
8	$\Gamma_{p,q,\hat{r}}$	$\underline{3}$	$[]$	$[\sharp p, +q]$	UPD
9	$\Gamma_{\hat{p},q,\hat{r}}$	$\underline{3}$	$[]$	$[+q]$	BOP-L
10	$\Gamma_{\hat{p},\hat{q},\hat{r}}$	$\texttt{Access}\ 0$	$[p]$	$[+q]$	ACC
11	$\Gamma_{\hat{p},\hat{q},\hat{r}}$	$\underline{3}$	$[]$	$[\sharp p, +q]$	UPD
12	$\Gamma_{\hat{p},\hat{q},\hat{r}}$	$\underline{3}$	$[]$	$[+q]$	BOP-R
13	$\Gamma_{\hat{p},\hat{q},\hat{r}}$	$\underline{6}$	$[]$	$[]$	

Figura 5.1: Ejemplo de evaluación

Definición 63. *Un compilador,* $(\!|\_|\!) : \mathit{Expr} \to \mathit{Code}$*, es correcto si para todo* $t \in \mathit{Expr}$ *tal que* $[] \vdash t : \theta$ *con* θ *cualquier tipo básico:*

1. *Si* $[\![\, t \,]\!]_{[],\theta}\, \diamond = \iota_{\uparrow}(c)$ *entonces* $(\Gamma, (\!|\, t \,|\!), [], []) \longmapsto^* (\Delta, \underline{c}, \eta, [])$
2. *Si* $[\![\, t \,]\!]_{[],\theta}\, \diamond = \bot$ *entonces* $(\Gamma, (\!|\, t \,|\!), [], []) \longmapsto^{\infty}$

Observaciones

La idea para probar la corrección de nuestro compilador es utilizar biortogonalidad como ya aplicamos en el capítulo anterior, para esto necesitamos introducir algunas nociones que nos sean de utilidad para definir las relaciones de *satisfactibilidad* que induzca los operadores ortogonales para cada caso de corrección del compilador.

Diremos que un heap esta *bien formado* cuando toda posible desreferenciación en una clausura esta definida en el mismo heap; es decir, todos los punteros que ocurren en cualquier clausura dentro de heap están incluidos en su dominio. Por ejemplo, uno puede chequear fácilmente que el heap

$\Gamma = \{\, p \mapsto (\texttt{Unit}, [\, q \,])\,\}$ no cumple con la propiedad de *buena formación* ya que $q \notin \mathtt{dom}(\Gamma)$. El predicado $wf(\Gamma)$ establece que Γ esta *bien formado* y se extiende a $(\Gamma, \alpha) \in (\mathit{Heap} \times \mathit{MClos})$ y $(\Gamma, s) \in (\mathit{Heap} \times \mathit{Stack})$.

Definición 64 (Heap bien formado)**.**

$$
\begin{aligned}
&wf(\Gamma) \;\; \textit{si y solo si} \;\; \mathtt{ptr}(\Gamma) = \mathtt{dom}(\Gamma)\\
&wf(\Gamma, \alpha) \;\; \textit{si y solo si} \;\; wf(\Gamma)\ y\ \mathtt{ptr}(\alpha) \subseteq \mathtt{dom}(\Gamma)\,,\\
&wf(\Gamma, s) \;\; \textit{si y solo si} \;\; wf(\Gamma)\ y\ \mathtt{ptr}(s) \subseteq \mathtt{dom}(\Gamma)\ .
\end{aligned}
$$

El siguiente lema es de suma utilidad y su validez se desprende directamente de la definición.

Lema 32.

1. *Si* $wf(\Gamma)$*, entonces para todo* $p \in \mathtt{dom}(\Gamma)$ *vale* $wf(\Gamma, \Gamma\, p)$.
2. *Si* $wf(\Gamma)$ *y* $\mathtt{ptr}(\alpha) \subseteq \mathtt{dom}(\Gamma)$*, entonces* $wf(\Gamma[\, p \mapsto \alpha\,])$.

En el capítulo anterior ocurría que existía una clara separación entre los realizadores (las expresiones) y los tests (los frame-stacks) y esto nos permitía determinar una relación de satisfactibilidad de manera directa; combinábamos la expresión y el frame-stack para formar una configuración que debía pertenecer al conjunto de observaciones. En esa misma linea, incluso si consideráramos la máquina abstracta de Krivine con evaluación call-by-name, donde una configuración de la máquina será una clausura junto con un stack, también ocurre que existe esta separación directa entre realizadores (las clausuras) y tests (los stacks) que nos permite definir la relación de satisfactibilidad con una estrategia similar.

Una configuración de la máquina abstracta que estamos considerando es una cuádrupla que consiste de un heap, una clausura (instrucción y entorno) y un stack. El problema ahora es que no surge una clara separación entre realizadores y tests ya que tanto la clausura como el stack necesitan del heap para estar bien formados. Por esta razón la noción de satisfactibilidad será más complicada; un realizador será un elemento en $\mathit{Heap} \times \mathit{MClos}$ (un heap junto con una clausura) y un tests será un elemento en $\mathit{Heap} \times \mathit{Stack}$ (un heap junto con un stack).

Ahora, uno podría ingenuamente pensar que una estrategia similar a la utilizada en el capítulo previo podría ser adecuada; dado un conjunto de observaciones (por ejemplo el de configuraciones que evalúan a una constante) la relación de satisfactibilidad estará definida de manera que

la combinación del realizador (un heap junto con una clausura) y el test (un heap junto a un stack) formen una configuración que deba pertenecer al conjunto de observaciones, donde la configuración final resultado de la combinación tenga como heap a la unión de los heaps involucrados en el realizador y el test. Claramente este enfoque conlleva problemas como el que ocurre cuando existe un mismo puntero asociando distintas clausuras en cada heap. Por lo tanto, tenemos que restringir que uniones de heaps serán validas para luego determinar si una configuración pertenece o no a un determinado conjunto de observaciones; diremos entonces que dos heaps Γ y Δ son *compatibles* si coinciden en todos sus punteros en común.

Definición 65. *Los heaps Γ y Δ son compatibles, denotado como $\Gamma \bowtie \Delta$, si $\Gamma\, p = \Delta\, p$, para todo $p \in \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta)$.*

La siguiente definición nos asegura que la unión de heaps esta bien definida cuando estos son compatibles.

Definición 66. *Si $\Gamma \bowtie \Delta$, la* unión $\Gamma \cup \Delta$ *con dominio $\mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Delta)$ se define como*

$$(\Gamma \cup \Delta)\, p = \begin{cases} \Gamma\, p & \textit{si } p \in \mathrm{dom}(\Gamma) \\ \Delta\, p & \textit{si } p \in \mathrm{dom}(\Delta) \end{cases}$$

A continuación enunciamos algunas propiedades básicas acerca de la relación $\bowtie$ y la unión de heaps.

Lema 33. *La relación $\bowtie$ es reflexiva y simétrica. Además, asumiendo $\Delta \bowtie \Delta'$, entonces $\Gamma \bowtie \Delta$ y $\Gamma \bowtie \Delta'$ si y solo si $\Gamma \bowtie (\Delta \cup \Delta')$.*

La unión de heaps compatibles es asociativa y conmutativa; la propiedad de estar bien formado se preserva bajo unión.

Lema 34. *Si $wf(\Gamma)$, $wf(\Delta)$, y $\Gamma \bowtie \Delta$, entonces $wf(\Gamma \cup \Delta)$.*

Una *observación* $\perp\!\!\!\perp$, en el esquema de realizabilidad de Krivine es un conjunto de configuraciones cerrado por anti-ejecución, es decir si $w' \in \perp\!\!\!\perp$ y $w \longmapsto w'$, entonces $w \in \perp\!\!\!\perp$. En nuestro caso vamos a pedir restricciones adicionales.

Definición 67. *Un conjunto* $\perp\!\!\!\perp \subseteq \mathit{Conf}$ *es una* observación *si es cerrado por anti-ejecución y además satisface las siguientes propiedades:*

$$\mathrm{O}_1 \ \frac{(\Gamma, i, \eta, s) \in \perp\!\!\!\perp}{(\Gamma \cup \Delta, i, \eta, s) \in \perp\!\!\!\perp}\ \mathit{wf}(\Gamma, s), \mathit{wf}(\Gamma, (i, \eta)), \Gamma \bowtie \Delta$$

$$\mathrm{O}_2 \ \frac{(\Gamma, i, \eta, s) \in \perp\!\!\!\perp}{(\Gamma, i, \eta, \sharp p :: s) \in \perp\!\!\!\perp}\ \Gamma\, p = (i, \eta)$$

$$\mathrm{OP}_1 \ \frac{(\Gamma, i, \eta, \pi_1 :: s) \in \perp\!\!\!\perp}{(\Gamma, i, \eta, \sharp p :: \pi_1 :: \sharp_1 p :: s) \in \perp\!\!\!\perp}\ \Gamma\, p = (i, \eta)$$

$$\mathrm{OP}_2 \ \frac{(\Gamma, i, \eta, \pi_2 :: s) \in \perp\!\!\!\perp}{(\Gamma, i, \eta, \sharp p :: \pi_2 :: \sharp_2 p :: s) \in \perp\!\!\!\perp}\ \Gamma\, p = (i, \eta)$$

La primera regla O_1 asegura que el heap de una configuración perteneciente al conjunto de observaciones puede ser extendido con punteros que son "irrelevantes" para la ejecución. Decimos que estos punteros son irrelevantes ya que de las condiciones de la regla se desprende que $\mathtt{ptr}(\eta) \cup \mathtt{ptr}(s) \subseteq \mathtt{dom}(\Gamma)$ y $\mathit{wf}(\Gamma)$; todo puntero que se encuentra en la configuración (Γ, i, η, s) pertenece al dominio de Γ, por lo tanto cualquier puntero perteneciente a $\mathtt{dom}(\Delta) - \mathtt{dom}(\Gamma)$ no desreferencia a ninguna clausura durante la ejecución, a pesar de estar en el heap $\Gamma \cup \Delta$. Notar que la restricción $\mathit{wf}(\Gamma)$ es importante para evitar que la ejecución cambie al extender con un nuevo heap; supongamos el conjunto de configuraciones terminantes, digamos $\perp\!\!\!\perp$, además sea $\Gamma = \{\, p \mapsto (\mathtt{Access}\ 0, [\, q\,])\,\}$, el cual no está bien formado ya que $q \notin \mathtt{dom}(\Gamma)$ y queremos extenderlo con el heap $\Delta = \{\, q \mapsto (\mathtt{Access}\ 0, [\, q\,])\,\}$. Claramente la ejecución de la configuración $(\Gamma, \mathtt{Access}\ 0, [p], s)$ se detiene y por lo tanto pertenece a $\perp\!\!\!\perp$, luego ocurre que $(\Gamma \cup \Delta, \mathtt{Access}\ 0, [p], s) \notin \perp\!\!\!\perp$; ya que la introducción de q origina que la ejecución puede ahora dar una cantidad arbitraria de pasos.

Las otras tres reglas permiten añadir marcadores al stack para actualizar alguna componente del heap. Todas estas reglas implican que el mecanismo de "sharing" (que nos evita repetir la evaluación de mismos argumentos) es una optimización que no debería afectar la observación.

Corrección para Expresiones Convergentes

Lo primero que probamos es la corrección del compilador para expresiones convergentes, esto es probar que dada una derivación para el juicio de

tipado $[] \vdash t : \theta$ con θ un tipo básico, que establece que t tiene tipo θ bajo el contexto vacío, si $[\![t]\!]_{[],\theta} \diamond = \iota_{\uparrow}(c)$ entonces $([], (\!| t |\!), [], []) \longmapsto^* (\Delta, \underline{c}, \eta, [])$.

Definir la relación de satisfactibilidad que nos induce los operadores ortogonales no es directo como ya mencionamos; el heap juega un papel clave tanto en los realizadores como en los tests, luego tenemos que ser cuidadosos en como los combinamos para formar una configuración. Por lo tanto utilizamos las definiciones de heap *bien formado* y la *compatibilidad* entre heaps para definir que un realizador estará relacionado con un test cuando ambos sean bien formados y además compatibles.

Definición 68 (Satisfactibilidad)**.** *Sea* $\perp\!\!\!\perp \subseteq Conf$ *una observación, luego* $\models \subseteq (Heap \times MClos) \times (Heap \times Stack)$ *es la relación más chica que satisface*

$$\frac{wf(\Gamma, \alpha),\ wf(\Delta, s),\ \Gamma \bowtie \Delta \text{ implica } (\Gamma \cup \Delta, \alpha, s) \in \perp\!\!\!\perp}{(\Gamma, \alpha) \models (\Delta, s)}$$

Definida esta relación de satisfactibilidad podemos explicitar los operadores ortogonales asociados.

Definición 69. *Sea* $\perp\!\!\!\perp \subseteq Conf$ *una observación, que nos determina la relación* $\_ \models \_$.

$$\begin{aligned}
&_^{\perp} : \mathcal{P}(Heap \times MClos) \to \mathcal{P}(Heap \times Stack)\\
&X^{\perp} = \{\, (\Delta, s) \mid \textit{para todo } (\Gamma, \alpha) \in X,\ (\Gamma, \alpha) \models (\Delta, s) \,\}\\
&_^{\top} : \mathcal{P}(Heap \times Stack) \to \mathcal{P}(Heap \times MClos)\\
&Y^{\top} = \{\, (\Gamma, \alpha) \mid \textit{para todo } (\Delta, s) \in Y,\ (\Gamma, \alpha) \models (\Delta, s) \,\}
\end{aligned}$$

Recordemos que estos operadores ortogonales definen una conexión de Galois antítona entre los posets $\mathcal{P}(Heap \times MClos)$ y $\mathcal{P}(Heap \times Stack)$, de manera que esto nos induce el operador de clausura

$$\_^{\perp\top} : \mathcal{P}(Heap \times MClos) \to \mathcal{P}(Heap \times MClos)$$

Una interpretación para los realizadores que vamos a definir en esta etapa de la prueba de corrección es que estos son aproximaciones denotacionales, por lo tanto los realizadores primitivos serán aproximaciones directas; por ejemplo la clausura $(\underline{m}, \eta)$ junto con cualquier heap constituye un realizador primitivo del valor denotacional $m \in \mathbb{Z}$.

Definición 70 (Realizadores primitivos y Realizadores)**.** *Sea* $\perp\!\!\!\perp \subseteq \mathit{Conf}$ *una observación. La relación* $\mathcal{R}^{\theta}_{\mathsf{p}}(\_) \subseteq [\![\,\theta\,]\!]^{\nu} \times (\mathit{Heap} \times \mathit{MClos})$ *se define por inducción sobre* θ*.*

$$\frac{\mathtt{ptr}(\eta) \subseteq \mathtt{dom}(\Gamma)}{(\Gamma, \mathtt{Unit}, \eta) \in \mathcal{R}^{\mathit{unit}}_{\mathsf{p}}(\diamond)} \qquad \frac{\mathtt{ptr}(\eta) \subseteq \mathtt{dom}(\Gamma)}{(\Gamma, \underline{m}, \eta) \in \mathcal{R}^{\mathit{int}}_{\mathsf{p}}(m)}$$

$$\frac{\begin{array}{c}\textit{Para todo } \Gamma',\ \alpha \textit{ y } d \textit{ tal que} \\ \mathtt{ptr}(\eta) \subseteq \mathtt{dom}(\Gamma) \quad \mathit{wf}(\Gamma', \alpha) \\ \Gamma \bowtie \Gamma' \quad (\Gamma \cup \Gamma') \bowtie \{p \mapsto \alpha\} \\ (\Gamma', \alpha) \in \mathcal{R}^{\theta}(d) \quad (\Gamma \cup \Gamma' \cup \{p \mapsto \alpha\}, i, p :: \eta) \in \mathcal{R}^{\theta'}(f\,d)\end{array}}{(\Gamma, \mathtt{Grab} \triangleright i, \eta) \in \mathcal{R}^{\theta \to \theta'}_{\mathsf{p}}(f)}$$

$$\frac{\mathtt{ptr}(\eta) \subseteq \mathtt{dom}(\Gamma) \quad (\Gamma, i, \eta) \in \mathcal{R}^{\theta}(d_0) \quad (\Gamma, i', \eta) \in \mathcal{R}^{\theta'}(d_1)}{(\Gamma, \mathtt{Pair}\ (i, i'), \eta) \in \mathcal{R}^{\theta \times \theta'}_{\mathsf{p}}((d_0, d_1))}$$

$(\Gamma, \alpha) \in \mathcal{R}^{\theta}(d)$ *sii* $\forall d'$, *si* $d = \iota_{\uparrow}(d')$ *entonces* $(\Gamma, \alpha) \in \mathcal{R}^{\theta}_{\mathsf{p}}(d)^{\perp\top}$

El conjunto de tests dado un valor denotacional $d \in [\![\,\theta\,]\!]^{\nu}$ esta dado por el ortogonal de su realizador primitivo, $\mathcal{T}^{\theta}(d) = \mathcal{R}^{\theta}_{\mathsf{p}}(d)^{\perp}$.

Definición 71 (Aproximación denotacional)**.** *Let* $\perp\!\!\!\perp \subseteq \mathit{Conf}$*.*

$(\Gamma, \alpha) \rhd^{\theta} d$ *esta definida como* $(\Gamma, \alpha) \in \mathcal{R}^{\theta}(d)$

Claramente para cualquier tipo básico θ, $(\Gamma, (\underline{c}, \eta)) \rhd^{\theta} \iota_{\uparrow}(c)$ siempre y cuando $\mathtt{ptr}(\eta) \subseteq \mathtt{dom}(\Gamma)$, ya qué el operador biortogonal es extensivo (en este caso sobre los realizadores primitivos). Para probar que una función, digamos f, aproxima a $(\Gamma, (\mathtt{Grab} \triangleright i, \eta))$ bajo el tipo $\theta \to \theta'$ (es decir, $(\Gamma, (\mathtt{Grab} \triangleright i, \eta)) \rhd^{\theta \to \theta'} \iota_{\uparrow}(f)$) tenemos que mostrar que el valor denotacional resultado de aplicar la función a cualquier argumento aproxima al realizador que es fruto de extender el entorno de la clausura, cuya instrucción es el cuerpo del grab, con el puntero que referencia, en un heap extendido, al realizador aproximado por el argumento denotacional; tenemos que demostrar que $(\Gamma \cup \Gamma' \cup \{p \mapsto \alpha\}, (i, p :: \eta)) \rhd^{\theta'} f\ d$, suponiendo que $(\Gamma', \alpha) \rhd^{\theta} d$ donde los heaps y clausuras involucradas cumplen con la buena formación y compatibilidad.

Finalmente, notar que de la definición se desprende que $\perp$ aproxima a todos los realizadores; $(\Gamma, \alpha) \rhd^{\theta} \perp$.

Por otro lado, una estrategia general para concluir que algún valor denotacional $\iota_{\uparrow}(d)$ aproxima al realizador (Γ, α), es decir $(\Gamma, \alpha) \in \mathcal{R}^{\theta}(\iota_{\uparrow}(d)) = \mathcal{T}^{\theta}(d)^{\top}$, es probar que $(\Gamma, \alpha) \models (\Delta, s)$ para cualquier $(\Delta, s) \in \mathcal{T}^{\theta}(d)$. Luego de expandir la definición de satisfactibilidad, la estrategia se reduce a probar que $(\Gamma \cup \Delta, \alpha, s) \in \perp\!\!\!\perp$, asumiendo $wf(\Gamma, \alpha)$, $wf(\Delta, s)$ y $\Gamma \bowtie \Delta$.

Las restricciones adicionales que impusimos al conjunto de observaciones nos permiten probar algunos lemas útiles, como es el caso de la regla O_1, que nos permite extender el heap de un realizador uniéndolo con otro heap.

Lema 35. *Si* $wf(\Gamma, \alpha)$, $\Gamma \bowtie \Delta$ *y* $(\Gamma, \alpha) \rhd^{\theta} d$, *entonces* $(\Gamma \cup \Delta, \alpha) \rhd^{\theta} d$.

Demostración. Supongamos $wf(\Gamma \cup \Delta, \alpha)$, $d = \iota_{\uparrow}(d')$ y $(\Delta', s) \in \mathcal{T}^{\theta}(d')$ tal que $wf(\Delta', s)$ y $(\Gamma \cup \Delta) \bowtie \Delta'$ y probemos que $(\Gamma \cup \Delta \cup \Delta', \alpha, s) \in \perp\!\!\!\perp$.

Ya que $(\Gamma \cup \Delta) \bowtie \Delta'$, podemos concluir $\Gamma \bowtie \Delta'$ y $\Delta \bowtie \Delta'$. Además de $(\Gamma, \alpha) \rhd^{\theta} d$, utilizando el test tenemos que $(\Gamma \cup \Delta', \alpha, s) \in \perp\!\!\!\perp$. Utilizando el lema 34 sabemos que la unión esta bien formada $wf(\Gamma \cup \Delta')$. Además podemos probar fácilmente $(\Gamma \cup \Delta') \bowtie \Delta$, gracias a $\Gamma \bowtie \Delta$ y $\Delta \bowtie \Delta'$. Finalmente aplicando la regla O_1 concluimos $(\Gamma \cup \Delta' \cup \Delta, \alpha, s) \in \perp\!\!\!\perp$. ■

Introducimos además la relación de aproximación para contextos tipados que nos permite definir luego la aproximación no cerrada con respecto a un valor denotacional; cuando una función continua aproxima a una instrucción.

Definición 72 (Aproximación de Entornos)**.**

$$\frac{}{(\Gamma, []) \rhd^{[]} \diamond} \qquad \frac{p \in dom(\Gamma) \qquad (\Gamma, \Gamma\, p) \rhd^{\theta} d \qquad (\Gamma, \eta) \rhd^{\pi} \rho}{(\Gamma, p :: \eta) \rhd^{\theta :: \pi} (d, \rho)}$$

Las siguientes tres propiedades se desprenden directamente de la definición de aproximación de contextos.

Lema 36. *Sea* $(\Gamma, \eta) \rhd^{\pi} \rho$.

1. $|\eta| = |\pi|$ *y* $ptr(\eta) \subseteq dom(\Gamma)$.
2. *Para cualquier* $n < |\eta|$, $(\Gamma, \Gamma\,(\eta \cdot n)) \rhd^{\pi \cdot n} \rho \swarrow n$.

Tomando el lema 35 sobre la extensión del heap de un realizador, podemos enunciar y probar su equivalente para realizadores de entornos.

Lema 37. *Si* $wf(\Gamma)$, $\Gamma \bowtie \Delta$ *y* $(\Gamma, \eta) \rhd^{\pi} \rho$, *entonces* $(\Gamma \cup \Delta, \eta) \rhd^{\pi} \rho$.

Demostración. Procedemos por inducción en π.

CASO BASE: tenemos que $\pi = []$, luego es trivial ya que $(\Gamma \cup \Delta, []) \rhd^{[]} \diamond$.

CASO INDUCTIVO: en este caso consideramos el contexto como $\theta :: \pi$. Asumimos $(\Gamma, p :: \eta) \rhd^{\theta::\pi} (d, \rho)$ y probamos que $(\Gamma \cup \Delta, p :: \eta) \rhd^{\theta::\pi} (d, \rho)$. Por inversión en la hipótesis tenemos que vale para todo $p \in dom(\Gamma)$, $(\Gamma, \eta) \rhd^{\pi} \rho$ y $(\Gamma, \Gamma\, p) \rhd^{\theta} d$. La hipótesis inductiva nos permite concluir $(\Gamma \cup \Delta, \eta) \rhd^{\pi} \rho$. Por otro lado podemos probar que $wf(\Gamma, \Gamma\, p)$ puesto que sabemos $wf(\Gamma)$ y el lema 32. Finalmente, utilizando el lema 35 podemos deducir $(\Gamma \cup \Delta, \Gamma\, p) \rhd^{\theta} d$.

■

El siguiente resultado muestra que las relaciones de aproximación son modulares, esto significa que uno puede combinar tests y realizadores para formar nuevos realizadores de tipos más complejos. Por ejemplo, una aproximación para un entorno $(d, \rho) \in [\![\, \theta :: \pi \,]\!]$ se construye a partir de las aproximaciones para $d \in [\![\, \theta \,]\!]$ y $\rho \in [\![\, \pi \,]\!]$.

Lema 38. *Sea* $(\Gamma, \eta) \rhd^{\pi} \rho$ *con* $wf(\Gamma)$, *y* $(\Gamma', \alpha) \rhd^{\theta} d$ *con* $wf(\Gamma', \alpha)$, *y* $\Gamma \bowtie \Gamma'$. *Si* $(\Gamma \cup \Gamma') \bowtie \{p \mapsto \alpha\}$, *entonces* $(\Gamma \cup \Gamma' \cup \{p \mapsto \alpha\}, p :: \eta) \rhd^{\theta::\pi} (d, \rho)$.

Demostración. Renombremos $\Gamma \cup \Gamma' \cup \{p \mapsto \alpha\}$ como Γ''. Usando la definición 72 tenemos que probar que $(\Gamma'', \eta) \rhd^{\pi} \rho$ y $(\Gamma'', \Gamma''\, p) \rhd^{\theta} d$. La primera parte es fácil y se desprende de $(\Gamma, \eta) \rhd^{\pi} \rho$ y $wf(\Gamma)$, utilizando el lema 37. Para la segunda parte, ya que $\Gamma''\, p = \alpha$, tenemos que probar $(\Gamma'', \alpha) \rhd^{\theta} d$. Considerando que $\Gamma' \bowtie \Gamma$ y $\Gamma' \bowtie \{p \mapsto \alpha\}$, obtenemos $\Gamma' \bowtie (\Gamma \cup \{p \mapsto \alpha\})$. Luego podemos aplicar el lema 35 en $(\Gamma', \alpha) \rhd^{\theta} d$ y $wf(\Gamma', \alpha)$, para así concluir $(\Gamma' \cup \Gamma \cup \{p \mapsto \alpha\}, \alpha) \rhd^{\theta} d$. ■

La denotación de una expresión bien tipada es una función continua de la semántica del contexto en la semántica del tipo, luego una función continua aproximará a una instrucción siempre que al tomar un realizador del contexto tengamos un realizador del tipo; esta relación de aproximación nos define cuando una función continua $d : [\![\, \pi \,]\!] \to [\![\, \theta \,]\!]$ aproxima a una instrucción $i : Code$; suponiendo un heap Γ tal qué $wf(\Gamma)$, un entorno de la máquina η y un entorno semántico ρ, si ρ aproxima a (Γ, η), entonces la aplicación $d\, \rho$ aproxima a $(\Gamma, (i, \eta))$; es decir este es un realizador de $d\, \rho$.

Definición 73 (Aproximación no cerrada)**.**
La relación $\_ \rhd^{\pi,\theta} \_ \subseteq Code \times ([\![\, \pi \,]\!] \to [\![\, \theta \,]\!])$ *se define como*

$$i \rhd^{\pi,\theta} d \text{ si y solo si } wf(\Gamma) \text{ y } (\Gamma, \eta) \rhd^{\pi} \rho \text{ entonces } (\Gamma, (i, \eta)) \rhd^{\theta} d\rho$$

La corrección del compilador se desprende de la propiedad fundamental de las relaciones lógicas la cual expresará que dada una expresión bien tipada entonces su denotación aproxima a su compilación. El siguiente lema es el primero de una serie de lemas que nos permitirán enunciar y probar esta propiedad y por lo tanto el resultado final sobre la corrección del compilador para expresiones convergentes.

Lema 39 (Caracterización de realizadores para tipos básicos)**.**

1. $\texttt{Unit} \rhd^{\pi,\textit{unit}} \rho \mapsto \iota_{\uparrow}(\diamond)$
2. *Para todo* $z \in \mathbb{Z}$, $\underline{z} \rhd^{\pi,\textit{int}} \rho \mapsto \iota_{\uparrow}(z)$

Demostración. La prueba es análoga para ambos items. Tomemos $(\Gamma, \eta) \rhd^{\pi} \rho$, luego tenemos que probar que $(\Gamma, (\texttt{Unit}, \eta)) \rhd^{\textbf{unit}} \iota_{\uparrow}(\diamond)$. Lo cual es directo ya que, como mencionamos anteriormente, el operador de biortogonalidad es extensivo. ■

Los siguientes dos lemas son casi un resultado directo de la extensividad del operador biortogonal sobre los realizadores primitivos. El primero asegura que el producto de dos funciones continuas $\mathtt{d} : [\![\pi]\!] \to [\![\theta_1]\!]$ y $\mathtt{d}' : [\![\pi]\!] \to [\![\theta_2]\!]$, aproximan a la instrucción $\texttt{Pair}\ (\mathtt{i}, \mathtt{i}')$ siempre que $\mathtt{d}$ y $\mathtt{d}'$ aproximen a $\mathtt{i}$ e $\mathtt{i}'$ respectivamente. El segundo prueba que dada una función continua $\mathtt{d} : [\![\theta :: \pi]\!] \to [\![\theta']\!]$ que aproxima a una instrucción $\mathtt{i}$, entonces tenemos que la función continua $\langle \mathtt{d} \rangle : [\![\pi]\!] \to [\![\theta \to \theta']\!]^{\nu}$ aproxima a la instrucción $\texttt{Grab} \triangleright \mathtt{i}$.

Lema 40. *Si* $\mathtt{i} \rhd^{\pi,\theta_1} \mathtt{d}$ *y* $\mathtt{i}' \rhd^{\pi,\theta_2} \mathtt{d}'$ *entonces* $\texttt{Pair}\ (\mathtt{i}, \mathtt{i}') \rhd^{\pi,\theta_1 \times \theta_2} \iota_{\uparrow} \circ \langle\, \mathtt{d}\,,\, \mathtt{d}' \,\rangle$.

Demostración. Supongamos $(\Gamma, \eta) \rhd^{\pi} \rho$, luego tenemos que probar que $(\Gamma, (\texttt{Pair}\ (\mathtt{i}, \mathtt{i}'), \eta)) \rhd^{\theta_1 \times \theta_2} \iota_{\uparrow}((\mathtt{d}\rho, \mathtt{d}'\rho))$. Utilizando la extensividad del operador biortogonal tenemos que probar que

$$(\Gamma, (\texttt{Pair}\ (\mathtt{i}, \mathtt{i}'), \eta)) \in \mathcal{R}_{\mathsf{p}}^{\theta_1 \times \theta_2}((\mathtt{d}\rho, \mathtt{d}'\rho))$$

luego podemos completar la prueba utilizando el lema 36 y las hipótesis. ■

Lema 41. *Si* $\mathtt{i} \rhd^{\theta :: \pi, \theta'} \mathtt{d}$ *entonces* $\texttt{Grab} \triangleright \mathtt{i} \rhd^{\pi, \theta \to \theta'} \iota_{\uparrow} \circ \langle \mathtt{d} \rangle$.

Demostración. Sea $(\Gamma, \eta) \rhd^{\pi} \rho$, análogamente al lema pasado tenemos que probar $(\Gamma, (\texttt{Grab} \triangleright \mathtt{i}, \eta))$ es un realizador primitivo de $\iota_{\uparrow}(\langle \mathtt{d} \rangle \rho)$ bajo el tipo

$\theta \to \theta'$. Para esto tomamos $d' \in [\![\theta]\!]$, $p \in \mathit{Pointer}$, y $(\Gamma', \alpha) \in \mathcal{R}^{\theta}(d')$. Además asumimos que $\mathit{wf}(\Gamma', \alpha)$, $\Gamma \bowtie \Gamma'$, y $(\Gamma \cup \Gamma') \bowtie \{p \mapsto \alpha\}$. Sea $\Gamma'' = \Gamma \cup \Gamma' \cup \{p \mapsto \alpha\}$; aplicando el lema 38 obtenemos $(\Gamma'', p :: \eta) \rhd^{\theta :: \pi} (d', \rho)$. Ya que $\mathit{wf}(\Gamma'')$, por hipótesis obtenemos $(\Gamma'', (i, p :: \eta)) \rhd^{\theta'} d\,(d', \rho) = \langle d \rangle \rho\, d'$. Por lo tanto, podemos concluir $(\Gamma, \mathtt{Grab} \triangleright i, \eta) \in \mathcal{R}_p^{\theta \to \theta'}(\iota_\uparrow(\langle d \rangle \rho))$ como buscábamos. ■

El siguiente resultado nos permite construir el test para una función $f \in [\![\theta \to \theta']\!]$ combinando un realizador de $d \in [\![\theta]\!]$ y un test para el resultado de la aplicación que converge; $\iota_\uparrow(d') \in [\![\theta']\!]$ con $\iota_\uparrow(d')$ el resultado de la aplicación $f\,d$.

Lema 42. *Supongamos* $\Gamma \bowtie \Delta$, $(\Gamma, \alpha) \rhd^{\theta} d$ *tal que* $\mathit{wf}(\Gamma, \alpha)$, $f\,d = \iota_\uparrow(d')$, *y* $(\Delta, s) \in \mathcal{T}^{\theta'}(d')$ *con* $\mathit{wf}(\Delta, s)$. *Para todo* p *tal que* $(\Gamma \cup \Delta) \bowtie \{p \mapsto \alpha\}$, $(\Gamma \cup \Delta \cup \{p \mapsto \alpha\}, p :: s) \in \mathcal{T}^{\theta \to \theta'}(f)$.

Demostración. Sea $\Gamma' = \Gamma \cup \Delta \cup \{p \mapsto \alpha\}$; y probemos que $(\Gamma', p :: s) \in \mathcal{R}_p^{\theta \to \theta'}(f)^{\perp}$. Tomamos un realizador primitivo $(\Gamma'', \hat{\alpha}) \in \mathcal{R}_p^{\theta \to \theta'}(f)$ para la función f, tal que $\mathit{wf}(\Gamma'', \hat{\alpha})$ y $\Gamma' \bowtie \Gamma''$, y queremos probar $(\Gamma' \cup \Gamma'', \hat{\alpha}, p :: s) \in \perp\!\!\!\perp$. Por la definición de realizador primitivo funcional tenemos que $\hat{\alpha} = (\mathtt{Grab} \triangleright i, \eta)$ para algún $i \in \mathit{Code}$, $\eta \in \mathit{MEnv}$. Además, dado que $\Gamma'' \bowtie \Gamma$, $\Gamma'' \cup \Gamma \bowtie \{p \mapsto \alpha\}$ y que $(\Gamma'', \hat{\alpha})$ es realizador primitivo de f, sabemos que $(\Gamma'' \cup \Gamma \cup \{p \mapsto \alpha\}, i, p :: \eta) \in \mathcal{R}^{\theta'}(f\,d)$, más aun $(\Gamma'' \cup \Gamma \cup \{p \mapsto \alpha\}, i, p :: \eta) \in \mathcal{R}_p^{\theta'}(d')^{\perp\top}$. Notar que

$$(\Gamma' \cup \Gamma'', \hat{\alpha}, p :: s) \longmapsto (\Gamma' \cup \Gamma'', i, p :: \eta, s)$$

por lo tanto si probamos que esta ultima configuración está en $\perp\!\!\!\perp$, por antiejecución completamos la prueba. Sea $\Delta' = \Gamma'' \cup \Gamma \cup \{p \mapsto \alpha\}$, claramente vale $\Delta \bowtie \Delta'$ y $\mathit{wf}(\Delta')$, por lo tanto podemos concluir $(\Delta' \cup \Delta, i, p :: \eta, s) \in \perp\!\!\!\perp$ combinando el realizador y el test de d'. ■

Dada una aproximación de entornos y una aproximación funcional podemos construirnos la aproximación para la aplicación. En la prueba del siguiente lema en particular, pero además en general, tenemos que probar que algún valor denotacional aproxima a un realizador, este valor denotacional a priori puede ser $\perp$ o una inyección. El caso que sea $\perp$ es directo y por lo tanto vamos a omitirlo de las pruebas y solo nos enfocamos en suponer que el valor es una infección; comenzamos suponiendo $f\,(\rho \swarrow n) = \iota_\uparrow(d')$.

Lema 43. *Sea* $(\Gamma, \eta) \rhd^{\pi} \rho$, *con* $n < |\eta|$, *y* $\theta = \pi \cdot n$. *Si* $(\Gamma, (i, \eta)) \rhd^{\theta \to \theta'} \iota_{\uparrow}(f)$, *entonces* $(\Gamma, (\mathtt{Push}\ n \triangleright i, \eta)) \rhd^{\theta'} f(\rho \swarrow n)$.

Demostración. Sea $d = \rho \swarrow n$ y $p = \eta \cdot n$. Suponemos además que $f\, d = \iota_{\uparrow}(d')$, luego tomamos un test $(\Delta, s) \in \mathcal{T}^{\theta'}(d')$ tal que $wf(\Delta, s)$ y $\Gamma \bowtie \Delta$, con el fin de probar $(\Gamma \cup \Delta, \mathtt{Push}\ n \triangleright i, \eta, s) \in \perp\!\!\!\perp$.

Ya que $(\Gamma, \eta) \rhd^{\pi} \rho$ y $n < |\eta|$, por el lema 36 tenemos que $(\Gamma, \Gamma\, p) \in \mathcal{R}^{\theta}(d)$. Luego podemos utilizar el lema 42, ya que $(\Gamma \cup \Delta) \bowtie \{p \mapsto \Gamma\, p\}$, para concluir $(\Gamma \cup \Delta, p :: s) \in \mathcal{T}^{\theta \to \theta'}(f)$. Por lo tanto, como $\iota_{\uparrow}(f)$ es una aproximación al realizador $(\Gamma, (i, \eta))$ entonces sabemos que $(\Gamma \cup \Delta, i, \eta, p :: s) \in \perp\!\!\!\perp$. Finalmente completamos la prueba utilizando anti-ejecución. ■

En general será útil disponer de varios de los lemas de aproximación en las versiones no cerradas, es decir una vez probado que la denotación de una expresión aplicada a un entorno aproxima a un realizador podemos generalizar el entorno para luego enunciar y probar que la denotación de una expresión (vista ahora como una función de la semántica de un entorno en la semántica de un tipo) aproxima a la instrucción del realizador. Notar que d será una función continua de $[\![\pi]\!]$ en $[\![\theta \to \theta']\!]$, además $(\_ \swarrow n) : [\![\pi]\!] \to [\![\theta]\!]$.

Lema 44. *Sea* $i \rhd^{\pi, \theta \to \theta'} d$, *entonces* $\mathtt{Push}\ n \triangleright i \rhd^{\pi, \theta'} d \circ_{app} (\_ \swarrow n)$, *cuando* $n < |\eta|$, *y* $\pi \cdot n = \theta$.

Demostración. Supongamos $(\Gamma, \eta) \rhd^{\pi} \rho$ y $wf(\Gamma)$ luego queremos probar que $(\Gamma, (\mathtt{Push}\ n \triangleright i, \eta)) \rhd^{\theta'} (f \mapsto f(\rho \swarrow n))_{\perp\!\!\!\perp}(d\, \rho)$. Cuando $d\, \rho$ no converge la prueba es directa. Supongamos entonces $d\, \rho = \iota_{\uparrow}(d')$, luego podemos probar utilizando el lema 43 que $(\Gamma, (\mathtt{Push}\ n \triangleright i, \eta))$ es un realizador de $d'(\rho \swarrow n)$. ■

Dado que los entornos dan significado a las variables libres que son compiladas usando la instrucción `Access`, es intuitivo pensar que la aproximación de un entorno puede ser usada para obtener la aproximación de cada variable con respecto a la instrucción `Access`.

Lema 45. *Sea* $(\Gamma, \eta) \rhd^{\pi} \rho$. *Para todo* $n \in \mathbb{N}$ *tal que* $n < |\eta|$,

$$(\Gamma, (\mathtt{Access}\ n, \eta)) \rhd^{\pi \cdot n} \rho \swarrow n\ .$$

Demostración. Sea $\theta = \pi \cdot n$, $\iota_{\uparrow}(d) = \rho \swarrow n$, y $p = \eta \cdot n$. Tomamos un test para d, $(\Delta, s) \in \mathcal{T}^{\theta}(d)$ tal que $wf(\Delta, s)$ y $\Gamma \bowtie \Delta$, y queremos probar

$(\Gamma \cup \Delta, \texttt{Access}\ n, \eta, s) \in \perp\!\!\!\perp$. Por el lema 36, sabemos que $(\Gamma, \Gamma\, p) \rhd^{\theta} \iota_{\uparrow}(d)$; por lo tanto $(\Gamma \cup \Delta, \Gamma\, p, s) \in \perp\!\!\!\perp$.

Aplicando la regla O_2 obtenemos $(\Gamma \cup \Delta, \Gamma\, p, \sharp p :: s) \in \perp\!\!\!\perp$. Además, como tenemos $(\Gamma \cup \Delta, \texttt{Access}\ n, \eta, s) \longmapsto (\Gamma \cup \Delta, \Gamma\, p, \sharp p :: s)$ y $\perp\!\!\!\perp$ es cerrado por anti-ejecución, hemos completado la prueba. ■

Claramente el lema anterior nos prueba de manera directa que la función continua $\_ \swarrow n$ aproxima denotacionalmente a la instrucción $\texttt{Access}\ n$; es decir que vale $\texttt{Access}\ n \rhd^{\pi,\pi\,\cdot\, n} \_ \swarrow n$. El siguiente lema nos asegura que un test para la primera componente de una tupla es efectivamente un test para la tupla donde agregamos al stack el marcador para la correspondiente proyección. Omitimos la prueba del lema análogo para el caso de la segunda proyección.

Lema 46. *Si* $(\Delta, s) \in \mathcal{T}^{\theta_1}(d_1)$ *tal que* $wf(\Delta, s)$ *y* $\Gamma \bowtie \Delta$, *entonces*

$$(\Gamma \cup \Delta, \pi_1 :: s) \in \mathcal{T}^{\theta_1 \times \theta_2}((\iota_{\uparrow}(d_1), d_2))$$

Demostración. Tomamos $(\Gamma', \alpha') \in \mathcal{R}_P^{\theta_1 \times \theta_2}((\iota_{\uparrow}(d_1), d_2))$ tal que $wf(\Gamma', \alpha')$, $wf(\Gamma \cup \Delta, \pi_1 :: s)$ y $\Gamma' \bowtie \Gamma \cup \Delta$ y queremos probar

$$(\Gamma' \cup \Gamma \cup \Delta, \alpha', \pi_1 :: s) \in \perp\!\!\!\perp\ .$$

Como en los lemas anteriores vamos a utilizar anti-ejecución para completar la prueba. Dado que (Γ', α') es un realizador primitivo, sabemos que $\alpha' = (\texttt{Pair}\ (i, i'), \eta)$; como $(\Gamma' \cup \Gamma \cup \Delta, \alpha', \pi_1 :: s) \longmapsto (\Gamma' \cup \Gamma \cup \Delta, i, \eta, s)$, vamos a probar que esta ultima configuración esta en la observación. En este punto estamos alcanzando una configuración en la cual todos los punteros de $dom(\Gamma)$ son irrelevantes, por lo tanto podemos utilizar O_1 y probar que $(\Gamma' \cup \Delta, i, \eta, s) \in \perp\!\!\!\perp$. Es directo chequear que las condiciones para aplicar la regla O_1 se satisfacen: podemos deducir $wf(\Gamma' \cup \Delta, s)$ y $wf(\Gamma' \cup \Delta, (i, \eta))$ de $wf(\Delta, s)$ y $wf(\Gamma', \alpha')$; luego $\Gamma' \cup \Delta \bowtie \Gamma$ vale directamente de notar que $\Gamma' \bowtie \Gamma$ (ya que $\Gamma' \bowtie \Gamma \cup \Delta$) y tenemos que $\Gamma \bowtie \Delta$, por lo tanto podemos concluir utilizando el lema 33.

Para completar la prueba notemos que $(\Gamma', i, \eta) \in \mathcal{R}^{\theta_1}(\iota_{\uparrow}(d_1))$, por inversión en el realizador primitivo para las tuplas, y $(\Delta, s) \in \mathcal{T}^{\theta_1}(d_1)$ es un test para el realizador (Γ', i, η). Ya que las tres propiedades ($wf(\Gamma', (i, \eta))$, $wf(\Delta, s)$ y $\Gamma' \bowtie \Delta$) de satisfactibilidad se cumplen, podemos concluir $(\Gamma' \cup \Delta, i, \eta, s) \in \perp\!\!\!\perp$. ■

Utilizando el lema anterior podemos probar que la función continua $(\rho \mapsto \pi_1 \circ_k (\rho \swarrow n)) : [\![\pi]\!] \to [\![\theta_1]\!]$ aproxima denotacionalmente a la instrucción `Fst` n.

Lema 47. *Sea* $n < |\eta|$ *y* $\pi \cdot n = \theta_1 \times \theta_2$, *entonces* `Fst` $n \rhd^{\pi,\theta_1} \pi_1 \circ_k (\_ \swarrow n)$.

Demostración. Supongamos $(\Gamma, \eta) \rhd^{\pi} \rho$ y $wf(\Gamma)$ luego queremos probar que $(\Gamma, (\texttt{Fst}\ n, \eta)) \rhd^{\theta_1} (\pi_1)_{\perp\!\!\!\perp}(\rho \swarrow n)$. Supongamos $\iota_\uparrow((d_1, d_2)) = \rho \swarrow n$; usando el lema 36 tenemos que $\eta \cdot n = p$ y $(\Gamma, \Gamma\, p) \in \mathcal{R}^{\theta_1 \times \theta_2}(\iota_\uparrow((d_1, d_2)))$.

Ahora, utilizando lo anterior probamos que $(\Gamma, \texttt{Fst}\ n, \eta) \in \mathcal{R}^{\theta_1}(d_1)$; supongamos $d_1 = \iota_\uparrow(d_1')$, luego se reduce a probar que para todo test $(\Delta, s) \in \mathcal{T}^{\theta_1}(d_1')$, tal que $wf(\Gamma, (\texttt{Fst}\ n, \eta))$, $wf(\Delta, s)$, y $\Gamma \bowtie \Delta$, la configuración está en la observación: $(\Gamma \cup \Delta, \texttt{Fst}\ n, \eta, s) \in \perp\!\!\!\perp$. Por anti-ejecución vamos a tener que probar $(\Gamma \cup \Delta, \Gamma\, p, \sharp p :: \pi_1 :: \sharp_1 p :: s) \in \perp\!\!\!\perp$. Utilizando la regla OP_1 podemos eliminar los marcadores de actualización del stack y probar que $(\Gamma \cup \Delta, \Gamma\, p, \pi_1 :: s) \in \perp\!\!\!\perp$.

El lema 46 nos ayuda a terminar la prueba ya que podemos construirnos un test para el producto utilizando el test (Δ, s) y utilizar el realizador correspondiente. Luego podemos concluir $(\Gamma \cup \Delta, \Gamma\, p, \pi_1 :: s) \in \perp\!\!\!\perp$ ■

Los realizadores de los valores enteros m y m' pueden ser combinados con la instrucción del operador binario para obtener un realizador para $m \odot m'$; los siguientes lemas testifican exactamente esto.

Lema 48. *Si* $(\Delta, s) \in \mathcal{T}^{\mathit{int}}(m \odot m')$ *tal que* $wf(\Delta, s)$, $\Gamma \bowtie \Delta$ *y* $wf(\Gamma, (m, \eta))$, *entonces*

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\, \underline{m}\, \Box, \eta)\}, \odot p :: s) \in \mathcal{T}^{\mathit{int}}(m') \ .$$

Donde $p \notin \mathrm{ptr}(\Delta)$.

Demostración. Definamos $\overline{\Gamma} = \Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\, \underline{m}\, \Box, \eta')\}$. Tomemos $(\Gamma', \underline{m'}, \eta') \in \mathcal{R}_p^{\mathbf{int}}(m')$ tal que $wf(\Gamma', (\underline{m'}, \eta'))$ y $\Gamma' \bowtie \overline{\Gamma}$ y probemos que $(\Gamma' \cup \overline{\Gamma}, \underline{m'}, \eta', \odot p :: s) \in \perp\!\!\!\perp$. Por anti-ejecución es suficiente demostrar que $(\Gamma' \cup \overline{\Gamma}, \underline{m \odot m'}, \eta, s) \in \perp\!\!\!\perp$. Usando que $\mathrm{ptr}(\eta) \subseteq \mathrm{dom}(\Gamma)$ obtenemos

$$(\Gamma, (\underline{m \odot m'}, \eta)) \in \mathcal{R}^{\mathbf{int}}(\iota_\uparrow(m \odot m'))$$

luego utilizando el lema 35 tenemos

$$(\Gamma' \cup \Gamma \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\, \underline{m}\, \Box, \eta)\}, (\underline{m \odot m'}, \eta)) \in \mathcal{R}^{\mathbf{int}}(\iota_\uparrow(m \odot m'))$$

finalmente utilizando $(\Delta, s) \in \mathcal{T}^{\mathbf{int}}(m \odot m')$ podemos completar la prueba. ■

Lema 49. *Si* $(\Gamma, i', \eta) \in \mathcal{R}^{int}(\iota_{\uparrow}(m'))$, $(\Delta, s) \in \mathcal{T}^{int}(m \odot m')$ *tal que* $\Gamma \bowtie \Delta$, $wf(\Delta, s)$ *y* $wf(\Gamma, (\mathtt{BOp}_{\oplus}\ i\ i', \eta))$, *entonces*

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus} \Box\, i', \eta)\}, \odot p :: s) \in \mathcal{T}^{int}(m)$$

donde $p \notin \mathtt{ptr}(\Gamma \cup \Delta)$

Demostración. Sea $\overline{\Gamma} = \Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus} \Box\, i', \eta)\}$. Tomamos el realizador primitivo $(\Gamma', \underline{m}, \eta') \in \mathcal{R}^{\mathbf{int}}_{p}(m)$ tal que $wf(\Gamma', (\underline{m}, \eta'))$, $wf(\overline{\Gamma}, \odot p :: s)$ y $\Gamma' \bowtie \overline{\Gamma}$, y probemos que $(\Gamma' \cup \overline{\Gamma}, (\underline{m}, \eta'), \odot p :: s) \in \perp\!\!\!\perp$. Luego por anti-ejecución es suficiente con demostrar que

$$(\Gamma' \cup \overline{\Gamma} \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\ \underline{m} \Box, \eta')\}, (i', \eta), \odot p :: s) \in \perp\!\!\!\perp\ .$$

Utilizando el lema anterior podemos construirnos el test

$$(\Gamma' \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\ \underline{m} \Box, \eta')\}, \odot p :: s) \in \mathcal{T}^{\mathbf{int}}(m')$$

y utilizar que (Γ, i', η) es un realizador de m' para completar la prueba. ■

Los dos lemas anteriores forman parte clave de la prueba del siguiente lema que enuncia, como mencionamos antes en términos de realizadores, que dadas dos aproximaciones de enteros podemos combinarlas para construirnos la aproximación de la operación binaria aplicada a estos enteros.

Lema 50. *Si* $(\Gamma, i, \eta) \in \mathcal{R}^{int}(\iota_{\uparrow}(m))$ *y* $(\Gamma, i', \eta) \in \mathcal{R}^{int}(\iota_{\uparrow}(m'))$, *entonces*

$$(\Gamma, \mathtt{BOp}_{\oplus}\ i\ i', \eta) \in \mathcal{R}^{int}(\iota_{\uparrow}(m \odot m'))\ .$$

Demostración. Tomamos un test $(\Delta, s) \in \mathcal{T}^{\mathbf{int}}(m \odot m')$ tal que $wf(\Delta, s)$, $\Gamma \bowtie \Delta$ y $wf(\Gamma, (\mathtt{BOp}_{\oplus}\ i\ i', \eta))$, y queremos probar $(\Gamma \cup \Delta, (\mathtt{BOp}_{\oplus}\ i\ i', \eta), s) \in \perp\!\!\!\perp$. Lo cual por anti-ejecución implica probar que

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus} \Box\, i', \eta)\}, (i, \eta), \odot p :: s) \in \perp\!\!\!\perp$$

. Luego podemos utilizar el lema anterior para obtener un test

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus} \Box\, i', \eta)\}, \odot p :: s) \in \mathcal{T}^{\mathbf{int}}(m)$$

y concluir la prueba combinando este último test con el hecho de que (Γ, i, η) es un realizador de m. ■

De manera directa podemos enunciar y probar el lema análogo para la aproximación denotacional no cerrada; en este caso d y d' serán funciones continuas de $[\![\pi]\!]$ en $[\![\mathbf{int}]\!]$.

Lema 51. *Si* $\mathtt{i} \rhd^{\pi,int} \mathtt{d}$ *y* $\mathtt{i}' \rhd^{\pi,int} \mathtt{d}'$ *entonces* $\mathtt{BOp}_{\oplus}\ \mathtt{i}\ \mathtt{i}' \rhd^{\pi,int} \mathtt{d} \circ_{\odot} \mathtt{d}'$.

Demostración. Supongamos $(\Gamma, \eta) \rhd^{\pi} \rho$ y $wf(\Gamma)$ luego queremos probar que $(\Gamma, (\mathtt{BOp}_{\oplus}\ \mathtt{i}\ \mathtt{i}', \eta)) \rhd^{\mathbf{int}} (\mathtt{d} \circ_{\odot} \mathtt{d}')\rho$. Supongamos $(\mathtt{d} \circ_{\odot} \mathtt{d}')\rho = \iota_{\uparrow}(m'')$; por lo tanto debe pasar que existen m y m' tal que $\mathtt{d}\rho = \iota_{\uparrow}(m)$, $\mathtt{d}'\rho = \iota_{\uparrow}(m')$ y además $m'' = m \odot m'$. Por lo tanto podemos utilizar el lema 50 para completar la prueba. ■

Utilizando una estrategia similar a la que usamos recientemente con los operadores binarios podemos probar los lemas relacionados con la expresión condicional.

Lema 52. *Si* $(\Gamma, \mathtt{i}', \eta) \in \mathcal{R}^{\theta}(\iota_{\uparrow}(\mathtt{d}'))$ *y* $(\Delta, \mathtt{s}) \in \mathcal{T}^{\theta}(\mathtt{d}')$ *tal que* $wf(\Delta, \mathtt{s})$, $\Gamma \bowtie \Delta$ *y* $wf(\Gamma, (\mathtt{IfZ}\ \mathtt{i}\ \mathtt{i}'\ \mathtt{i}'', \eta))$, *entonces*

$$(\Gamma \cup \Delta \cup \{\mathtt{p} \mapsto (\mathtt{IfZ}\,\square\,(\mathtt{i}', \mathtt{i}''), \eta)\},\ ?\mathtt{p} :: \mathtt{s}) \in \mathcal{T}^{int}(0)$$

donde $\mathtt{p} \notin \mathtt{ptr}(\Gamma \cup \Delta)$.

Demostración. Sea $\overline{\Gamma} = \Gamma \cup \Delta \cup \{\mathtt{p} \mapsto (\mathtt{IfZ}\,\square\,(\mathtt{i}', \mathtt{i}''), \eta)\}$. Tomamos el realizador primitivo $(\Gamma', \underline{0}, \eta') \in \mathcal{R}^{\mathbf{int}}_{\mathtt{p}}(0)$ tal que $wf(\Gamma', (\underline{0}, \eta'))$, $wf(\overline{\Gamma}, ?\mathtt{p} :: \mathtt{s})$ y $\Gamma' \bowtie \overline{\Gamma}$, y probemos que $(\Gamma' \cup \overline{\Gamma}, (\underline{0}, \eta'), ?\mathtt{p} :: \mathtt{s}) \in \perp\!\!\!\perp$. Luego por antiejecución es suficiente demostrar que $(\Gamma' \cup \overline{\Gamma}, (\mathtt{i}', \eta), \mathtt{s}) \in \perp\!\!\!\perp$. Notar que podemos aplicar O_1 para simplificar el heap, luego tenemos que demostrar $(\Gamma \cup \Delta, (\mathtt{i}', \eta), \mathtt{s}) \in \perp\!\!\!\perp$, esto es valido ya que $wf(\Gamma, (\mathtt{IfZ}\ \mathtt{i}\ \mathtt{i}'\ \mathtt{i}'', \eta))$, $wf(\Delta, \mathtt{s})$ y $\Gamma \bowtie \Delta$. Finalmente podemos completar la prueba utilizando que $(\Gamma, \mathtt{i}', \eta)$ y $(\Delta, \mathtt{s})$ son un realizador y test, respectivamente, del valor denotacional $\mathtt{d}'$. ■

Lema 53. *Si* $(\Gamma, \mathtt{i}'', \eta) \in \mathcal{R}^{\theta}(\iota_{\uparrow}(\mathtt{d}''))$ *y* $(\Delta, \mathtt{s}) \in \mathcal{T}^{\theta}(\mathtt{d}'')$ *tal que* $wf(\Delta, \mathtt{s})$, $\Gamma \bowtie \Delta$ *y* $wf(\Gamma, (\mathtt{IfZ}\ \mathtt{i}\ \mathtt{i}'\ \mathtt{i}'', \eta))$, *entonces*

$$(\Gamma \cup \Delta \cup \{\mathtt{p} \mapsto (\mathtt{IfZ}\,\square\,(\mathtt{i}', \mathtt{i}''), \eta)\},\ ?\mathtt{p} :: \mathtt{s}) \in \mathcal{T}^{int}(z)$$

donde $\mathtt{p} \notin \mathtt{ptr}(\Gamma \cup \Delta)$ *y* $z \neq 0$.

Demostración. Prueba análoga a la del lema 52. ■

Lema 54. *Si* $(\Gamma, \mathtt{i}, \eta) \in \mathcal{R}^{int}(\iota_{\uparrow}(0))$ *y* $(\Gamma, \mathtt{i}', \eta) \in \mathcal{R}^{\theta}(\iota_{\uparrow}(\mathtt{d}'))$, *entonces para cualquier* $\mathtt{i}''$, $(\Gamma, \mathtt{IfZ}\ \mathtt{i}\ \mathtt{i}'\ \mathtt{i}'', \eta) \in \mathcal{R}^{\theta}(\iota_{\uparrow}(\mathtt{d}'))$.

Demostración. Tomamos un test $(\Delta, s) \in \mathcal{T}^{\mathbf{int}}(d')$ tal que $wf(\Delta, s)$, $\Gamma \bowtie \Delta$ y $wf(\Gamma, (\mathtt{IfZ}\ i\ i'\ i'', \eta))$, y queremos probar $(\Gamma \cup \Delta, (\mathtt{IfZ}\ i\ i'\ i'', \eta), s) \in \perp\!\!\!\perp$. Lo cual por anti-ejecución se traduce el probar que

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{IfZ}\,\square\,(i', i''), \eta)\}, (i, \eta), ?p :: s) \in \perp\!\!\!\perp \ .$$

Utilizando el lema 52 nos podemos construir un test

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{IfZ}\,\square\,(i', i''), \eta)\}, ?p :: s) \in \mathcal{T}^{\mathbf{int}}(0)$$

y concluir la prueba combinando este ultimo test con el hecho de que (Γ, i, η) es un realizador de $\iota_\uparrow(0)$. ■

Lema 55. *Si* $(\Gamma, i, \eta) \in \mathcal{R}^{\boldsymbol{int}}(\iota_\uparrow(z))$, *con* $z \neq 0$ *y* $(\Gamma, i'', \eta) \in \mathcal{R}^{\theta}(\iota_\uparrow(d''))$, *entonces para cualquier* i', $(\Gamma, \mathtt{IfZ}\ i\ i'\ i'', \eta) \in \mathcal{R}^{\theta}(\iota_\uparrow(d''))$.

Demostración. Prueba análoga a la del lema 54. ■

Lema 56. *Si* $i \rhd^{\pi,\boldsymbol{int}} d$, $i' \rhd^{\pi,\theta} d'$ *y* $i'' \rhd^{\pi,\theta} d''$ *entonces*

$$\mathtt{IfZ}\ i\ i'\ i'' \rhd^{\pi,\theta} \boldsymbol{ifz}\,(d)\,(d')\,(d'') \ .$$

Demostración. Supongamos $(\Gamma, \eta) \rhd^{\pi} \rho$ y $wf(\Gamma)$ luego queremos probar que $(\Gamma, (\mathtt{IfZ}\ i\ i'\ i'', \eta)) \rhd^{\theta} (\mathbf{ifz}\,(d)\,(d')\,(d''))\rho$.

Además, supongamos $(\mathbf{ifz}\,(d)\,(d')\,(d''))\rho = \iota_\uparrow(d''')$; por lo tanto debe pasar que existen $z : Z$ tal que $d\rho = z$, además si $z = 0$ entonces $d' = \iota_\uparrow(d''')$, en otro caso $d'' = \iota_\uparrow(d''')$. Por lo tanto podemos utilizar los lemas 54 y 55 para completar la prueba. ■

El lema que quisiéramos probar a continuación supone dos funciones continuas $d : [\![\theta_1 :: \pi]\!] \to [\![\theta_1]\!]$ y $d' : [\![\theta_1 :: \pi]\!] \to [\![\theta_2]\!]$, que aproximan a instrucciones i e i' respectivamente, y nos permite concluir que la función continua $\mathbf{let}\,(\mathbf{Fix}(d))\,(d') : [\![\pi]\!] \to [\![\theta_2]\!]$ aproxima a la instrucción $\mathtt{Let}\ (\mathtt{Rec}\ i) \triangleright i'$. La estrategia de la prueba es la usual, suponemos $(\Gamma, \eta) \rhd^{\pi} \rho$, $wf(\Gamma)$, y $\mathbf{let}\,(\mathbf{Fix}(d))\,(d')\rho = d'(\mathbf{Fix}(d)\rho, \rho) = \iota_\uparrow(d'')$, luego tomamos un test (Δ, s) de d'' y probamos utilizando anti-ejecución que

$$((\Gamma \cup \Delta)[p \mapsto (\mathtt{Rec}\ i, \eta)], i', p :: \eta, s) \in \perp\!\!\!\perp \ .$$

Luego podemos completar la prueba utilizando que la instrucción i' es aproximada por la función continua d' bajo el contexto $\theta_1 :: \pi$ y tipo θ_2,

siempre y cuando nos construyamos una aproximación adecuada para el entorno $\theta_1 :: \pi$; la aproximación adecuada será

$$(\Gamma[p \mapsto (\mathtt{Rec}\, i, \eta)], p :: \eta) \rhd^{\theta_1 :: \pi} (\mathbf{Fix}(d)\rho, \rho) \ .$$

Utilizando el lema 38 para probar que el entorno semántico $(\mathbf{Fix}(d)\rho, \rho)$ aproxima a $(\Gamma[p \mapsto (\mathtt{Rec}\, i, \eta)], p :: \eta)$ implica probar principalmente dos cosas: 1. $(\Gamma, \eta) \rhd^{\pi} \rho$, que es directo por hipótesis 2. $(\Gamma, (\mathtt{Rec}\, i, \eta)) \rhd^{\theta_1} \mathbf{Fix}(d)\rho = \bigsqcup_{n=0}^{\infty} \langle d \rangle_n \rho$.

La necesidad de probar esto ultimo y que tenemos por hipótesis que d aproxima a la instrucción i nos induce un posible lema; si $i \rhd^{\theta_1 :: \pi, \theta_1} d$, entonces $\mathtt{Rec}\, i \rhd^{\pi, \theta_1} \mathbf{Fix}(d) = \bigsqcup_{n=0}^{\infty} \langle d \rangle_n$. Notar que si definimos el conjunto $\{ d \mid \mathtt{Rec}\, i \rhd^{\pi, \theta_1} d \}$ de valores denotacionales que aproximan a la instrucción $\mathtt{Rec}\, i$, nada nos asegura que perteneciendo todos los elementos de la cadena $\langle d \rangle_n$, el supremo pertenezca; es decir la definición de la relación lógica no nos permite asegurarnos que este conjunto sea cerrado por supremo de cadenas. La solución es definir una nueva relación de aproximación basada en la relación actual utilizando la clausura de Scott.

Definición 74 (Aproximación clausurada)**.**
La relación $\_ \blacktriangleright^{\pi,\theta} \_ \subseteq \mathit{Code} \times (\llbracket \pi \rrbracket \to \llbracket \theta \rrbracket)$ *se define como*

$$i \blacktriangleright^{\pi,\theta} d \text{ } \textit{si y solo si } d \in \mathrm{IC}(i \rhd^{\pi,\theta} \_)$$

Esta nueva relación nos asegura que dada una cadena de funciones continuas $d_n : \llbracket \pi \rrbracket \to \llbracket \theta \rrbracket$, si todo elemento de la cadena aproxima a una instrucción i entonces tenemos que el supremo $\bigsqcup_{n=0}^{\infty} d_n$ aproxima a la instrucción. En consecuencia un primer lema que nos interesará probar es aquel que dado que una función continua $d : \llbracket \theta :: \pi \rrbracket \to \llbracket \theta \rrbracket$ aproxima a una instrucción i, entonces cada miembro de la cadena $\langle d \rangle_n : \llbracket \pi \rrbracket \to \llbracket \theta \rrbracket$ aproxima a la instrucción $\mathtt{Rec}\, i$.

Lema 57. *Si* $i \rhd^{\theta :: \pi, \theta} d$*, entonces para todo* n*,* $\mathtt{Rec}\, i \rhd^{\pi,\theta} \langle d \rangle_n$.

Demostración. Procedemos por inducción en n.

CASO BASE: directo ya que $\bot : \llbracket \pi \rrbracket \to \llbracket \theta \rrbracket$ aproxima a cualquier instrucción.

CASO INDUCTIVO: suponemos $\mathtt{Rec}\,i \rhd^{\pi,\theta} \langle d\rangle_n$, $(\Gamma, \eta) \rhd^{\pi} \rho$ y $wf(\Gamma)$, luego tenemos que probar

$$(\Gamma, (\mathtt{Rec}\,i, \eta)) \rhd^{\theta} \langle d\rangle_{n+1}\rho = d\,(\langle d\rangle_n \rho, \rho)$$

Supongamos $d\,(\langle d\rangle_n\rho, \rho) = \iota_{\uparrow}(d')$ y tomemos un test (Δ, s) de d' tal que $wf(\Gamma, (\mathtt{Rec}\,i, \eta))$, $wf(\Delta, s)$ y $\Gamma \bowtie \Delta$, luego por anti-ejecución y contemplando $(\Gamma \cup \Delta)[p \mapsto (\mathtt{Rec}\,i, \eta)] = \Gamma \cup \{p \mapsto (\mathtt{Rec}\,i, \eta)\} \cup \Delta$ tenemos que probar

$$(\Gamma \cup \{p \mapsto (\mathtt{Rec}\,i, \eta)\} \cup \Delta, i, p :: \eta, s) \in \perp\!\!\!\perp,$$

Utilizando la hipótesis inductiva en conjunto con el lema 38 obtenemos $(\Gamma \cup \{p \mapsto (\mathtt{Rec}\,i, \eta)\}, p :: \eta) \rhd^{\theta :: \pi} (\langle d\rangle_n \rho, \rho)$. Por lo tanto, podemos completar la prueba haciendo uso de la aproximación $i \rhd^{\theta :: \pi, \theta} d$.

■

El lema equivalente para la relación de aproximación clausurada se desprende directamente de la utilización del lema 2 del capitulo 2; cuyo enunciado nos dice principalmente que la clausura de Scott es extensiva.

Lema 58. *Si* $i \blacktriangleright^{\theta :: \pi, \theta} d$, *entonces para todo* n, $\mathtt{Rec}\,i \blacktriangleright^{\pi,\theta} \langle d\rangle_n$.

El siguiente lema nos prueba el resultado fundamental que nos condicionó el intento de estrategia de prueba para el caso de la expresión *let* en el contexto de la prueba de la propiedad fundamental.

Lema 59. *Si* $i \blacktriangleright^{\theta :: \pi, \theta} d$, *entonces* $\mathtt{Rec}\,i \blacktriangleright^{\pi,\theta} \boldsymbol{Fix}(d)$.

Demostración. Queremos probar que $\mathtt{Rec}\,i \blacktriangleright^{\pi,\theta} \sqcup_{n>0}(\langle d\rangle_n)$, luego como esta relación de aproximación es cerrada por una propiedad básica de las clausuras de Scott. Tenemos que probar que para todo n, $\mathtt{Rec}\,i \blacktriangleright^{\pi,\theta} \langle d\rangle_n$, lo cual es directo de aplicar el lema 58. ■

Lema 60. *Si* $i \blacktriangleright^{\theta :: \pi, \theta} d$ *y* $i' \blacktriangleright^{\theta :: \pi, \theta'} d'$, *entonces*

$$\mathtt{Let}\,(\mathtt{Rec}\,i) \triangleright i' \blacktriangleright^{\pi,\theta'} \boldsymbol{let}\,(\boldsymbol{Fix}(d))\,(d')\ .$$

Demostración. Suponemos $i \blacktriangleright^{\theta :: \pi, \theta} d$ y $i' \blacktriangleright^{\theta :: \pi, \theta'} d'$. Luego utilizando el lema 2, donde fijamos $P_1 = \{d \mid i' \rhd^{\theta :: \pi, \theta'} d\}$ y $P_2 = \{d \mid \mathtt{Rec}\,i \rhd^{\pi,\theta} d\}$ nos implica probar que:

- Para todo $a : [\![\theta :: \pi]\!] \to [\![\theta']\!]$ y $b : [\![\pi]\!] \to [\![\theta]\!]$ tal que $i' \rhd^{\theta :: \pi, \theta'} a$ y $\mathtt{Rec}\, i \rhd^{\pi,\theta} b$ entonces $\mathtt{Let}\ (\mathtt{Rec}\, i) \triangleright i' \rhd^{\pi,\theta'} \mathbf{let}\,(b)\,(a)$.

  Notar que para probar esto nos sirve la estrategia de prueba antes mencionada. Supongamos $(\Gamma, \eta) \rhd^{\pi} \rho$ y $wf(\Gamma)$, luego tomamos un test (Δ, s) de d'' tal que $\mathbf{let}\,(b)\,(a)\rho = a(b\rho, \rho) = \iota_{\uparrow}(d'')$ y queremos probar, luego de utilizar anti-ejecución, que

  $$(\Gamma[p \mapsto (\mathtt{Rec}\, i, \eta)] \cup \Delta, i', p :: \eta, s) \in \perp\!\!\!\perp \ .$$

  Utilizando que b aproxima a $\mathtt{Rec}\, i$, además ρ aproxima a (Γ, η) y el lema 38 obtenemos $\Gamma[p \mapsto (\mathtt{Rec}\, i, \eta)] \rhd^{\theta :: \pi} (b\rho, \rho)$. Finalmente podemos completar la prueba de este caso utilizando que $i' \rhd^{\theta :: \pi, \theta'} a$.

- $i' \blacktriangleright^{\theta :: \pi, \theta'} d'$

  Directa por hipótesis.

- $\mathtt{Rec}\, i \blacktriangleright^{\pi,\theta} \mathbf{Fix}(d)$

  Utilizando el lema 59.

■

La propiedad fundamental de las relaciones lógicas prueba que la compilación de cualquier expresión bien tipada es aproximada por su denotación. Expusimos la dificultad de probar que la denotación de la expresión *let* aproxima a su compilación, luego definimos una nueva relación de aproximación que nos permite solucionar este inconveniente y por lo tanto la relación que esperamos cumpla con la propiedad fundamental es la aproximación clausurada. Sin embargo, para el resto de las expresiones tenemos probado que su denotación aproxima a su compilación utilizando la aproximación no clausurada. Ahora, utilizando el lema 2 podemos obtener la prueba de cualquier aproximación clausurada a partir de una aproximación no clausurada; el siguiente lema expone de manera general la estrategia de la prueba.

Lema 61. *Sea* n*, tal que* $0 \leqslant n$ *donde* $\boldsymbol{I}(i_1, i_2, \ldots, i_n)$ *representa una instrucción que se construye en términos de instrucciones* i_j *y, de manera análoga para valores denotacionales,* $\boldsymbol{OP}(d_1, d_2, \ldots, d_n)$ *construye un valor denotacional en base a valores denotacionales* d_j*.*

Si para todo $j = 1 \ldots n$*,*

$$i_j \rhd^{\pi_j,\theta_j} d_j \textit{ implica } \boldsymbol{I}(i_1, i_2, \ldots, i_n) \rhd^{\pi,\theta} \boldsymbol{OP}(d_1, d_2, \ldots, d_n)$$

entonces
Si para todo $j = 1 \ldots n$,

$$i_j \blacktriangleright^{\pi_j,\theta_j} d_j \textit{ implica } \boldsymbol{I}(i_1, i_2, \ldots, i_n) \blacktriangleright^{\pi,\theta} \boldsymbol{OP}(d_1, d_2, \ldots, d_n)$$

Demostración. Suponemos $i_j \blacktriangleright^{\pi_j,\theta_j} d_j$ para $j = 1 \ldots n$ y queremos probar, $\boldsymbol{I}(i_1, i_2, \ldots, i_n) \blacktriangleright^{\pi,\theta} \boldsymbol{OP}(d_1, d_2, \ldots, d_n)$, luego podemos utilizar el lema 2 tomando como los $P_i = \{\, d \mid i_j \rhd^{\pi_j,\theta_j} d \,\}$. Por lo tanto lo que queda por probar es directo de la hipótesis principal y el resto de n-hipótesis. ■

Para ejemplificar la utilidad de este lema consideremos lo siguiente: tenemos probado por el lema 51 que $\mathtt{BOp}_\oplus\ i\ i' \rhd^{\pi,\mathbf{int}} d \circ_\odot d'$ cuando $i \rhd^{\pi,\mathbf{int}} d$ y $i' \rhd^{\pi,\mathbf{int}} d'$, luego si instanciamos $\boldsymbol{I}$ como $\mathtt{BOp}_\oplus\ \_\ \_$ y $\boldsymbol{OP}$ como $\_ \circ_\odot \_$ podemos aplicar este ultimo resultado y concluir que si $i \blacktriangleright^{\pi,\mathbf{int}} d$ y $i' \blacktriangleright^{\pi,\mathbf{int}} d'$ entonces $\mathtt{BOp}_\oplus\ i\ i' \blacktriangleright^{\pi,\mathbf{int}} d \circ_\odot d'$.

Teorema 13. *Si* $\pi \vdash t : \theta$ *entonces* $(\!|\, t \,|\!) \blacktriangleright^{\pi,\theta} [\![\, t \,]\!]_{\pi,\theta}$.

Demostración. Procedemos por inducción en la derivación del juicio de tipado $\pi \vdash t : \theta$ y utilizamos los lemas previos para completar la prueba. ■

Notar que el teorema anterior es independiente de la *observación* $\perp\!\!\!\perp$, luego fijando una observación particular quisiéramos obtener la prueba de que el compilador es correcto con respecto a una expresión cerrada cuyo tipo sea un tipo básico. Para esto probamos un lema intermedio, antes de enunciar y probar el resultado principal, que nos permite "recuperar" la noción de aproximación original, es decir sin clausurar. Por supuesto este lema no puede probarse de manera general para cualquier entorno y tipo, si no que solo contempla los tipos básicos y expresiones cerradas, pero que es exactamente lo que necesitamos.

Lema 62. *Sea* θ *cualquier tipo básico,*

$$\textit{si } i \blacktriangleright^{[],\theta} \iota_\uparrow \circ d \textit{ entonces } i \rhd^{[],\theta} \iota_\uparrow \circ d$$

Demostración. Sea $d' = d \diamond$, ya que $d : [\![\, [] \,]\!] \to [\![\, \theta \,]\!]^\nu$. Supongamos Γ tal que $wf(\Gamma)$, luego tenemos $(\Gamma, [])\rhd^{[]} \diamond$ y queremos probar $(\Gamma, i, []) \in \mathcal{R}^\theta_p(d')^{\perp\top}$. Primero probaremos que

$$\iota_\uparrow(d') \in (i \rhd^{[],\theta} \_)\,((\Gamma, []) \rhd^{[]} \_)$$

luego utilizando los lemas 3 y 4 del capítulo 2, se traduce en probar

$$\iota_\uparrow(d') \in (i \blacktriangleright^{[],\theta} \_)\,(\mathrm{IC}((\Gamma, []) \rhd^{[]} \_))$$

Por la definición del operador tenemos que demostrar que existen $f : [\![\square]\!] \to [\![\theta]\!]$ y $a : [\![\square]\!]$ tal que $i \blacktriangleright^{\square,\theta} f$, $IC((\Gamma, \square) \rhd^{\square} a)$ y $\iota_{\uparrow}(d') = f\, a$. Tomando $f = \iota_{\uparrow} \circ d$ y $a = \diamond$ es directo por nuestras hipótesis, además usando la extensividad de la clausura de Scott podemos probar que $\diamond$ aproxima a $(\Gamma, \square)$ en la versión clausurada, ya que lo hace en la aproximación sin clausurar.

Por lo tanto, tenemos que $\iota_{\uparrow}(d') \in (i \rhd^{\square,\theta} \_)\, ((\Gamma, \square) \rhd^{\square} \_)$ lo cual nos asegura que existen $f : [\![\square]\!] \to [\![\theta]\!]$ y $a : [\![\square]\!]$ tal que $i \rhd^{\square,\theta} f$, $(\Gamma, \square) \rhd^{\square} a$ y $\iota_{\uparrow}(d') = f\, a$. Luego necesariamente $f = \iota_{\uparrow} \circ d$ y $a = \diamond$, esto nos permite completar la prueba dado que hemos probado que $(\Gamma, (i, \square)) \rhd^{\theta} \iota_{\uparrow}(d')$. ■

Teorema 14. *Sea* t *una expresión,* Γ *un heap tal que* $wf(\Gamma)$ *y* θ *un tipo básico. Si* $\square \vdash t : \theta$, $[\![t]\!]_{\square,\theta} \diamond = \iota_{\uparrow}(c)$ *con* $c : [\![\theta]\!]^{v}$*, entonces*

$$(\Gamma, (\!| t |\!), \square, \square) \longmapsto^{*} (\Delta, \underline{c}, \eta, \square)$$

para algún heap $\Delta \in Heap$ *y un entorno de la máquina* $\eta \in MEnv$.

Demostración. La observación que buscamos es el conjunto de configuraciones que alcanzan un estado bloqueante de la máquina con la instrucción $\underline{c}$ y la pila vacía:

$$\perp\!\!\!\perp = \{ w \in Conf \mid w \longmapsto^{*} (\Delta, \underline{c}, \eta, \square) \} \ .$$

Suponiendo que $\perp\!\!\!\perp$ cumple con las cuatro condiciones enunciadas en 67. Utilizando el teorema 13 obtenemos $(\!| t |\!) \blacktriangleright^{\square,\theta} [\![t]\!]_{\square,\theta}$ luego por el lema 62 concluimos $(\!| t |\!) \rhd^{\square,\theta} [\![t]\!]_{\square,\theta}$. Ya que $wf(\Gamma)$ y $(\Gamma, \square) \rhd^{\square} \diamond$, tenemos $(\Gamma, ((\!| t |\!), \square)) \rhd^{\theta} \iota_{\uparrow}(\underline{c})$. Para completar la prueba solo nos hace falta probar que $(\Gamma, \square) \in \mathcal{T}^{\theta}(\underline{c})$, lo cual es directo ya que tomando realizador primitivo $(\Gamma', \underline{c}, \eta) \in \mathcal{R}^{\theta}_{P}(c)$ tenemos que probar $(\Gamma' \cup \Gamma, \underline{c}, \eta, \square) \in \perp\!\!\!\perp$. ■

Considerando la observación $\perp\!\!\!\perp = \{ w \in Conf \mid w \longmapsto^{*} (\Delta, \underline{n}, \eta, \square) \}$ para una constante particular $\underline{n}$ podemos instanciar el teorema para el tipo básico de los **int**.

Teorema 15. *Sea* t *una expresión,* Γ *un heap tal que* $wf(\Gamma)$*. Si* $\square \vdash t : \textbf{\textit{int}}$*,* $[\![t]\!]_{\square,\textbf{\textit{int}}} \diamond = \iota_{\uparrow}(n)$*, entonces* $(\Gamma, (\!| t |\!), \square, \square) \longmapsto^{*} (\Delta, \underline{n}, \eta, \square)$ *para algún heap* $\Delta \in Heap$ *y un entorno de la máquina* $\eta \in MEnv$.

Es importante notar que la corrección del compilador para el tipo funcional es capturado por la noción de aproximación: uno puede deducir que

una función está correctamente implementada por un código de bajo nivel si computa lo que debería cuando se le proveen todos sus argumentos. Por ejemplo, dada la función matemática $\mathtt{fib}$ que aplicada a un natural n calcula el n-ésimo número de la sucesión de Fibonacci. Si consideramos el programa t del ejemplo 5.6 como la implementación de la función Fibonacci en nuestro lenguaje de alto nivel, luego podemos probar por inducción en n que para todo $0 \leqslant n$, $[\![t]\!]_{[],\mathbf{int}} \diamond = \iota_{\uparrow}(\mathtt{fib}\, n)$. Por lo tanto, utilizando el teorema 15 podemos concluir que el programa de bajo nivel $(\!| t |\!)$ computa a $\underline{\mathtt{fib}\, n}$. Notar que en la expresión t existe una meta-variable n. La expresión del ejemplo 5.7 representa el programa equivalente utilizando variables con nombre.

```
1  let n̲ in
2  let λ (ifz 0̄ then 0̲
3        else ifz (0̄ − 1̲) then 1̲
4        else let 1̄ − 1̲ in
5             let 2̄ − 2̲
6             in 3̄1̄ + 3̄0̄
7   )
8  in 0̄1̄
9
```

(5.6) Usando índices De Bruijn

```
1  let n = n̲ in
2  let fib = λ m. (ifz m then 0̲
3        else ifz (m − 1̲) then 1̲
4        else let n′ = m − 1̲ in
5             let n″ = m − 2̲
6             in fib n′ + fib n″
7   )
8  in fib n
9
```

(5.7) Usando variables con nombre

Finalmente, para probar la corrección del compilador con respecto al producto de dos tipos no alcanza con demostrar que el código de bajo nivel que implementa la definición de un par matemático computa a la instrucción `Pair`. La razón esta en que nuestro método de evaluación es lazy y por lo tanto durante la evaluación la máquina podría detenerse en la instrucción $\mathtt{Pair}\ (i, i')$, pero alguna (o ambas) de las componentes diverger si exigiéramos su computación. Por lo tanto, para demostrar que un programa t es una implementación de un par p, podemos construirnos las expresiones $t_1 = \mathit{let}\ t\ \mathit{in}\ (\mathit{fst}\ \bar{0})$ y $t_2 = \mathit{let}\ t\ \mathit{in}\ (\mathit{snd}\ \bar{0})$, probar que $[\![t_1]\!]_{[],\theta_1} \diamond = \iota_{\uparrow}(\pi_1\, p)$ y $[\![t_2]\!]_{[],\theta_2} \diamond = \iota_{\uparrow}(\pi_2\, p)$, con θ_1, θ_2 tipos básicos. Luego podemos utilizar el teorema 14 como hicimos anteriormente.

Corrección para Expresiones Divergentes

A continuación probaremos la segunda parte de la definición de un compilador correcto 63; sea t una expresión tal que existe una derivación para

el juicio de tipado $[] \vdash t : \theta$, si $[\![t]\!]_{[],\theta} \diamond = \bot$ entonces $(\Gamma, (\!| t |\!), [], []) \longmapsto^{\infty}$, para cualquier $wf(\Gamma)$.

El desarrollo para probar la corrección del compilador para expresiones divergentes será muy similar al estudiado para el caso de la expresiones convergentes, las principales diferencias estarán dadas por el uso de la noción de Step-Indexed. Utilizaremos la definición de Step-Indexed para definir una nueva noción de observación; ahora indexada. Luego podemos definir la relación de satisfactibilidad, haciendo uso de esta nueva noción, que nos induzca los operadores ortogonales que después utilizaremos para definir la relación de aproximación operacional. En este contexto una *observación* $\perp\!\!\!\perp_i$ será el conjunto de configuraciones tal que la máquina ejecuta al menos i pasos. Notar que si una configuración pertenece al conjunto $\perp\!\!\!\perp_i$ para cualquier i, esta configuración ocasiona que la máquina sea capaz de dar una cantidad arbitraria de pasos.

Definición 75. *Un conjunto $\perp\!\!\!\perp_i \subseteq Conf$ es una* observación *si es step-indexed y además satisface las siguientes propiedades:*

$$\text{ANTI-EJECUCIÓN}\ \frac{w' \in \perp\!\!\!\perp_i}{w \in \perp\!\!\!\perp_{i+1}}\ w \longmapsto w'$$

$$O_1\ \frac{(\Gamma, i, \eta, s) \in \perp\!\!\!\perp_i}{(\Gamma \cup \Delta, i, \eta, s) \in \perp\!\!\!\perp_i}\ wf(\Gamma, s), wf(\Gamma, (i, \eta)), \Gamma \bowtie \Delta$$

$$O_2\ \frac{(\Gamma, i, \eta, s) \in \perp\!\!\!\perp_i}{(\Gamma, i, \eta, \sharp p :: s) \in \perp\!\!\!\perp_i}\ \Gamma\, p = (i, \eta)$$

$$OP_1\ \frac{(\Gamma, i, \eta, \pi_1 :: s) \in \perp\!\!\!\perp_i}{(\Gamma, i, \eta, \sharp p :: \pi_1 :: \sharp_1 p :: s) \in \perp\!\!\!\perp_i}\ \Gamma\, p = (i, \eta)$$

$$OP_2\ \frac{(\Gamma, i, \eta, \pi_2 :: s) \in \perp\!\!\!\perp_i}{(\Gamma, i, \eta, \sharp p :: \pi_2 :: \sharp_2 p :: s) \in \perp\!\!\!\perp_i}\ \Gamma\, p = (i, \eta)$$

Luego podemos definir la relación de satisfactibilidad usando la noción de observación, donde imponemos que los heaps involucrados deben ser compatibles y bien-formados. Notar que los realizadores y tests ahora están indexados.

Definición 76 (Satisfactibilidad)**.** *Sea $\perp\!\!\!\perp_i \subseteq Conf$ una observación, luego* $\models\ \subseteq (\mathbb{N} \times (Heap \times MClos)) \times (\mathbb{N} \times (Heap \times Stack))$ *es la relación más chica que satisface*

$$\frac{wf(\Gamma, \alpha), wf(\Delta, s), \Gamma \bowtie \Delta \textit{ implica } (\Gamma \cup \Delta, \alpha, s) \in \perp\!\!\!\perp_{\min(i,j)}}{(i, (\Gamma, \alpha)) \models (j, (\Delta, s))}$$

Esta relación nos induce los operadores ortogonales, los cuales definen una conexión de Galois antítona que luego nos induce el operador de clausura $\_^{\perp\top} : \mathcal{P}(\mathbb{N} \times (\mathit{Heap} \times \mathit{MClos})) \to \mathcal{P}(\mathbb{N} \times (\mathit{Heap} \times \mathit{MClos}))$.

Definición 77. *Sea* $\perp\!\!\!\perp_i \subseteq \mathit{Conf}$ *una observación, que nos determina la relación* $\_ \models \_$.

$$
\begin{aligned}
&\_^{\perp} : \mathcal{P}(\mathbb{N} \times (\mathit{Heap} \times \mathit{MClos})) \to \mathcal{P}(\mathbb{N} \times (\mathit{Heap} \times \mathit{Stack})) \\
&X^{\perp} = \{\, (j, (\Delta, s)) \mid \textit{para todo } (i, (\Gamma, \alpha)) \in X, \\
&\qquad\qquad (i, (\Gamma, \alpha)) \models (j, (\Delta, s)) \,\} \\
&\_^{\top} : \mathcal{P}(\mathbb{N} \times (\mathit{Heap} \times \mathit{Stack})) \to \mathcal{P}(\mathbb{N} \times (\mathit{Heap} \times \mathit{MClos})) \\
&Y^{\top} = \{\, (i, (\Gamma, \alpha)) \mid \textit{para todo } (j, (\Delta, s)) \in Y, \\
&\qquad\qquad (i, (\Gamma, \alpha)) \models (j, (\Delta, s)) \,\} \ .
\end{aligned}
$$

Para cualquier θ, dada una relación $R^{\theta} \subseteq [\![\, \theta \,]\!]^{\nu} \times (\mathbb{N} \times (\mathit{Heap} \times \mathit{MClos}))$ definimos su extensión $R^{\theta}_{\perp} \subseteq [\![\, \theta \,]\!] \times (\mathbb{N} \times (\mathit{Heap} \times \mathit{MClos}))$.

Definición 78 (Relación extendida)**.**

$$(i, (\Gamma, \alpha)) \in R^{\theta}_{\perp}(d) \textit{ si existe } d' \in [\![\, \theta \,]\!]^{\nu} \textit{ tal que } d = \iota_{\uparrow}(d') \textit{ y } (\Gamma, \alpha) \in R^{\theta}(d')$$

Definición 79 (Realizadores primitivos y Realizadores)**.** *Sea* $\perp\!\!\!\perp_n \subseteq \mathit{Conf}$ *una observación. La relación* $\mathcal{R}^{\theta}_{p}(\_) \subseteq [\![\, \theta \,]\!]^{\nu} \times (\mathbb{N} \times (\mathit{Heap} \times \mathit{MClos}))$ *se define por inducción sobre* θ.

$$\frac{\mathrm{ptr}(\eta) \subseteq \mathrm{dom}(\Gamma)}{(\Gamma, \mathtt{Unit}, \eta) \in {}^{n}\mathcal{R}^{\mathit{unit}}_{p}(\diamond)} \qquad \frac{\mathrm{ptr}(\eta) \subseteq \mathrm{dom}(\Gamma)}{(\Gamma, \underline{m}, \eta) \in {}^{n}\mathcal{R}^{\mathit{int}}_{p}(m)}$$

$$\frac{\begin{array}{c}\textit{Para todo } n', \Gamma', \alpha \textit{ y } d \textit{ tal que } n' < n \\ \mathrm{ptr}(\eta) \subseteq \mathrm{dom}(\Gamma) \quad \mathit{wf}(\Gamma', \alpha) \\ \Gamma \bowtie \Gamma' \quad (\Gamma \cup \Gamma') \bowtie \{p \mapsto \alpha\} \\ (\Gamma', \alpha) \in {}^{n'}\mathcal{R}^{\theta}(d) \quad (\Gamma \cup \Gamma' \cup \{p \mapsto \alpha\}, i, p :: \eta) \in {}^{n'}\mathcal{R}^{\theta'}(f\,d)\end{array}}{(\Gamma, \mathtt{Grab} \triangleright i, \eta) \in {}^{n}\mathcal{R}^{\theta \to \theta'}_{p}(f)}$$

$$\frac{\begin{array}{c}\textit{Para todo } n', \textit{ tal que } n' < n \\ \mathrm{ptr}(\eta) \subseteq \mathrm{dom}(\Gamma) \quad (\Gamma, i, \eta) \in {}^{n'}\mathcal{R}^{\theta}(d_0) \quad (\Gamma, i', \eta) \in {}^{n'}\mathcal{R}^{\theta'}(d_1)\end{array}}{(\Gamma, \mathtt{Pair}\ (i, i'), \eta) \in {}^{n}\mathcal{R}^{\theta \times \theta'}_{p}((d_0, d_1))}$$

$(\Gamma, \alpha) \in {}^{n}\mathcal{R}^{\theta}(d)$ *si y solo si* $(n, (\Gamma, \alpha)) \in (\mathcal{R}^{\theta}_{p\perp}(d))^{\perp\top}$

Por conveniencia escribimos el índice de cualquier realizador como supraíndice de la relación correspondiente; escribiremos $(\Gamma, \alpha) \in {}^{n}\mathcal{R}^{\theta}_{p}(d)$ para referir a $(n, (\Gamma, \alpha)) \in \mathcal{R}^{\theta}_{p}(d)$, análogo para los realizadores no primitivos. El conjunto de tests queda definido como $\mathcal{T}^{\theta}(d) = (\mathcal{R}^{\theta}_{p_\perp}(d))^{\perp}$, con $d \in [\![\, \theta \,]\!]^{v}$.

La definición de aproximación operacional captura la noción de que tan bien una instrucción aproxima a un valor denotacional cuando realiza alguna cantidad de transiciones en la máquina. Notar que si una instrucción aproxima a $\perp$ para cualquier cantidad de transiciones, entonces podemos concluir que esta instrucción ocasiona que la máquina diverja.

Definición 80 (Aproximación operacional)**.** *Sea* $\Vdash_n \subseteq \mathit{Conf}$*.*

$$(\Gamma, \alpha) \lhd^{\theta}_{n} d \text{ se define como } (\Gamma, \alpha) \in {}^{n}\mathcal{R}^{\theta}(d) \ .$$

Los conjuntos de realizadores primitivos, realizadores y tests son step-indexed. En el caso de los realizadores primitivos, la prueba es directa de probar por casos en cada tipo. A continuación probamos la propiedad solo para el conjunto de realizadores (la prueba es análoga para los tests).

Lema 63. *Para todo* n*:*

- *Si* $(\Gamma, \alpha) \in {}^{n+1}\mathcal{R}^{\theta}_{p}(d)$ *entonces* $(\Gamma, \alpha) \in {}^{n}\mathcal{R}^{\theta}_{p}(d)$
- *Si* $(\Gamma, \alpha) \in {}^{n+1}\mathcal{R}^{\theta}(d)$ *entonces* $(\Gamma, \alpha) \in {}^{n}\mathcal{R}^{\theta}(d)$
- *Si* $(\Delta, s) \in {}^{n+1}\mathcal{T}^{\theta}(d)$ *entonces* $(\Delta, s) \in {}^{n}\mathcal{T}^{\theta}(d)$

Demostración. Supongamos $(\Gamma, \alpha) \in {}^{n+1}\mathcal{R}^{\theta}(d)$, tomemos un $(\Delta, s) \in {}^{j}\mathcal{T}^{\theta}(d)$ tal que $wf(\Gamma, \alpha)$, $wf(\Delta, s)$ y $\Gamma \bowtie \Delta$, tenemos que probar

$$(\Gamma \cup \Delta, \alpha, s) \in \Vdash_{\min(i,j)}$$

. Además tenemos que vale $(\Gamma \cup \Delta, \alpha, s) \in \Vdash_{\min(i+1,j)}$.

- Si $\min(i, j) = i$, podemos completar la prueba usando que nuestra observación es step-indexed.
- Si $\min(i, j) = j$, entonces la prueba es directa.

■

Haciendo uso de este lema podemos enunciar y probar el siguiente resultado más general.

Lema 64. *Para todo* n *y* n'*, tal que* $n' \leqslant n$*:*

- *Si* $(\Gamma, \alpha) \in {}^{n}\mathcal{R}_{p}^{\boldsymbol{unit}}(d)$ *entonces* $(\Gamma, \alpha) \in {}^{n'}\mathcal{R}_{p}^{\boldsymbol{unit}}(d)$
- *Si* $(\Gamma, \alpha) \in {}^{n}\mathcal{R}^{\boldsymbol{unit}}(d)$ *entonces* $(\Gamma, \alpha) \in {}^{n'}\mathcal{R}^{\boldsymbol{unit}}(d)$
- *Si* $(\Gamma, \alpha) \in {}^{n}\mathcal{T}^{\theta}(d)$ *entonces* $(\Gamma, \alpha) \in {}^{n'}\mathcal{T}^{\theta}(d)$

Demostración. La prueba es directa de hacer inducción en n y utilizar el lema 64. ■

La aproximación de entornos se define guiada por la construcción de los entornos de la máquina; un entorno vacío emparejado con cualquier heap aproximará al entorno semántico vacío, además un entorno $p :: \eta$ junto con un heap Γ, tal que $p \in \mathrm{dom}(\Gamma)$, aproximará a un entorno semántico (d, ρ) si el realizador $(\Gamma, \Gamma\, p)$ aproxima a d y el resto entorno junto con Γ aproximan a ρ.

Definición 81 (Aproximación de Entornos)**.**

$$\frac{}{(\Gamma, []) \lhd_n^{[]} \diamond} \qquad \frac{p \in \mathrm{dom}(\Gamma) \qquad (\Gamma, \Gamma\, p) \lhd_n^{\theta} d \qquad (\Gamma, \eta) \lhd_n^{\pi} \rho}{(\Gamma, p :: \eta) \lhd_n^{\theta :: \pi} (d, \rho)}$$

Las siguientes propiedades son de alguna manera análogas a aquellas que enunciamos para el caso de la aproximación denotacional, más aun las estrategias de prueba son equivalentes.

Lema 65. *Sea* $(\Gamma, \eta) \lhd_n^{\pi} \rho$*.*

1. $|\eta| = |\pi|$ *y* $\mathrm{ptr}(\eta) \subseteq \mathrm{dom}(\Gamma)$*.*
2. *Para cualquier* $n < |\eta|$*,* $(\Gamma, \Gamma\,(\eta \cdot n)) \lhd_n^{\pi \cdot n} \rho \swarrow n$*.*
3. *Si* $wf(\Gamma)$*,* $\Gamma \bowtie \Delta$ *y* $(\Gamma, \eta) \lhd_n^{\pi} \rho$*, entonces* $(\Gamma \cup \Delta, \eta) \lhd_n^{\pi} \rho$*.*
4. *Sea* $(\Gamma, \eta) \lhd_n^{\pi} \rho$ *y* $(\Gamma', \alpha) \lhd_n^{\theta} d$*, tal que* $wf(\Gamma)$*,* $wf(\Gamma', \alpha)$*, y* $\Gamma \bowtie \Gamma'$*. Si* $(\Gamma \cup \Gamma') \bowtie \{p \mapsto \alpha\}$*, entonces* $(\Gamma \cup \Gamma' \cup \{p \mapsto \alpha\}, p :: \eta) \lhd_n^{\theta :: \pi} (d, \rho)$*.*

Además podemos probar que esta relación también es step-indexed utilizando que las relaciones anteriores cumplen también con la propiedad.

Lema 66. *Para todo* $n \in \mathbb{N}$*, si* $(\Gamma, \eta) \lhd_{n+1}^{\pi} \rho$ *entonces* $(\Gamma, \eta) \lhd_n^{\pi} \rho$*.*

Demostración. Procedamos por inducción en π, cuando el contexto es vacío la prueba es directa. Si tenemos $\theta :: \pi$, luego nuestra hipótesis es que $(\Gamma, p :: \eta) \lhd^{\theta :: \pi}_{n+1} (d, \rho)$ y queremos probar $(\Gamma, p :: \eta) \lhd^{\theta :: \pi}_{n} (d, \rho)$. Por la definición de aproximación de entornos esto es probar que $(\Gamma, \Gamma\, p) \lhd^{\theta}_{n} d$ y $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$, cuando $p \in dom(\Gamma)$. La primera aproximación que tenemos que probar es directa de utilizar Ía hipótesis del lema y que el conjunto de realizadores es step-indexed como asegura el lema 63. La segunda aproximación también es directa de utilizar la hipótesis inductiva y la hipótesis principal. ■

Lema 67. *Para todo* $n, n' \in \mathbb{N}$ *tal que* $n' \leqslant n$, *si* $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$ *entonces*

$$(\Gamma, \eta) \lhd^{\pi}_{n'} \rho \ .$$

Demostración. La prueba es directa por inducción en n y el lema anterior. ■

La aproximación no cerrada define cuando una función continua $d : [\![\pi]\!] \to [\![\theta]\!]$ aproxima a una instrucción i : *Code* para cualquier cantidad de transiciones de la máquina. Suponiendo un índice n, un heap Γ tal qué $wf(\Gamma)$ y entornos η y ρ, si ρ aproxima a (Γ, η) con índice n, entonces la aplicación $d\,\rho$ aproxima a $(\Gamma, (i, \eta))$ en n; es decir $(n, (\Gamma, (i, \eta)))$ es un realizador de $d\,\rho$.

Definición 82 (Aproximación no cerrada)**.**
La relación $\_ \lhd^{\pi,\theta} \_ \subseteq Code \times ([\![\pi]\!] \to [\![\theta]\!])$ *se define como*

$i \lhd^{\pi,\theta} d$ *si y solo si* $\forall (n \in \mathbb{N})$, $wf(\Gamma)$ *y* $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$ *entonces* $(\Gamma, (i, \eta)) \lhd^{\theta}_{n} d\rho$

Para probar la propiedad fundamental de las relaciones lógicas enunciamos y probamos los siguientes lemas. En particular el primero lo enunciaremos de la forma más general posible; probaremos que cualquier instrucción i_ν de la máquina que represente un valor constante aproxima a toda función constante $\lambda\rho\,.\,d : [\![\pi]\!] \to [\![\theta]\!]^{\nu}$, con θ un tipo básico. Notar que suponiendo θ es un tipo básico entonces tenemos que $[\![\theta]\!]^{\nu}$ es un orden discreto.

Lema 68. $i_\nu \lhd^{\pi,\theta} (\lambda\rho\,.\,\iota_{\uparrow}(d))$

Demostración. Tomamos un $n \in \mathbb{N}$ tal que $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$ y queremos probar $(\Gamma, (i_\nu, \eta)) \in {}^{n}\mathcal{R}^{\theta}(\iota_{\uparrow}(d))$. Por definición $(n, (\Gamma, (i_\nu, \eta)))$ es un realizador primitivo de $\iota_{\uparrow}(d)$. Por lo tanto, completamos la prueba notando que el operador biortogonal define la mayor extensión de $\mathcal{R}^{\theta}_{P\perp}(\iota_{\uparrow}(d))$. ■

Lema 69. *Si* $i \lhd^{\pi,\theta_1} d$ *y* $i' \lhd^{\pi,\theta_2} d'$ *entonces* $\texttt{Pair}\ (i, i') \lhd^{\pi,\theta_1\times\theta_2} \iota_\uparrow \circ \langle d, d' \rangle$.

Demostración. Supongamos $n \in \mathbb{N}$ y $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$, luego tenemos que probar que $(\Gamma, (\texttt{Pair}\ (i, i'), \eta)) \lhd^{\theta_1\times\theta_2}_{n} \iota_\uparrow((d\rho, d'\rho))$. Utilizando la extensionalidad del operador biortogonal tenemos que probar que

$$(\Gamma, (\texttt{Pair}\ (i, i'), \eta)) \in {}^{n}\mathcal{R}^{\theta_1\times\theta_2}_{p}((d\rho, d'\rho))$$

lo cual por definición es probar que tomando un m tal que $m < n$ entonces $(\Gamma, (i, \eta)) \in {}^{m}\mathcal{R}^{\theta_1}_{p}(d\rho)$, $(\Gamma, (i', \eta)) \in {}^{m}\mathcal{R}^{\theta_2}_{p}(d'\rho)$ y $ptr(\alpha) \subseteq dom(\Gamma)$. Esto ultimo es directo por el lema 36. Utilizando las hipótesis en cada caso nos restaría probar que $(\Gamma, \eta) \lhd^{\pi}_{m} \rho$, lo cual se traduce a probar $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$ utilizando 67. Por lo tanto completamos la prueba. ■

Lema 70. *Si* $i \lhd^{\theta::\pi,\theta'} d$ *entonces* $\texttt{Grab} \triangleright i \lhd^{\pi,\theta\to\theta'} \iota_\uparrow \circ \langle d \rangle$.

Demostración. Tomemos $n \in \mathbb{N}$ y $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$ y probemos

$$(n, (\Gamma, (\texttt{Grab} \triangleright i, \eta)))$$

es un realizador primitivo de $\iota_\uparrow(\langle d \rangle \rho)$. Tomamos un $m \in \mathbb{N}$ tal que $m < n$, $d' \in [\![\theta]\!]$, y $(\Gamma', \alpha) \in {}^{m}\mathcal{R}^{\theta}(d')$. Además asumimos que los heaps cumplen $wf(\Gamma', \alpha)$, $\Gamma \bowtie \Gamma'$, y $(\Gamma \cup \Gamma') \bowtie \{p \mapsto \alpha\}$ con el puntero p fresco.

Sea $\Gamma'' = \Gamma \cup \Gamma' \cup \{p \mapsto \alpha\}$, utilizando los lemas 65 y 67 probamos que $(\Gamma'', p :: \eta) \lhd^{\theta::\pi}_{m} (d', \rho)$. Luego podemos utilizar la hipótesis para concluir la prueba. ■

Recurrentemente en las pruebas de varios de los siguientes lemas utilizaremos una misma estrategia inicial que sentará la base de cada prueba particular, cada prueba iniciará utilizando el lema 5, si queremos probar que un tests pertenece al algún conjunto de tests, o el lema 6 para el caso de realizadores.

Supongamos queremos probar que un realizador $(n, (\Gamma, \alpha))$ aproxima a un valor denotacional d; $(n, (\Gamma, \alpha)) \in (\mathcal{R}^{\theta}_{p_\perp}(d))^{\perp\top}$. La prueba procedería suponiendo un test que cumple $(m, (\Delta, s)) \in (\mathcal{R}^{\theta}_{p_\perp}(d))^{\perp}$ y probando

$$(\Gamma \cup \Delta, \alpha, s) \in \perp\!\!\!\perp_{min(n,m)}$$

siempre que Γ y Δ sean compatibles y bien formados. Notar que n y m son cualquiera, luego el lema 6 simplifica la prueba de manera que solo consideremos el caso en que $m \leqslant n$; $min(n, m) = m$. Luego debemos probar $(\Gamma \cup \Delta, \alpha, s) \in \perp\!\!\!\perp_{m}$. Además, para probar que una configuración

esta en el conjunto de observaciones nos interesa poder ejecutar un paso y probar que la configuración a la que avanzamos pertenece a un conjunto de observaciones con menor índice. Es cómodo entonces suponer que $m = 0$ o $m = m' + 1$, luego es directo por definición que $(\Gamma \cup \Delta, \alpha, s) \in \perp\!\!\!\perp_0$ y restaría probar $(\Gamma', \alpha', s') \in \perp\!\!\!\perp_{m'}$ con $m' \leqslant n$, dado que $(\Gamma \cup \Delta, \alpha, s) \longmapsto (\Gamma', \alpha', s')$. De manera análoga para tests utilizamos el lema 5.

Lema 71. *Para todo* $n \in \mathbb{N}$, *si* $\Gamma \bowtie \Delta$, $wf(\Delta, s)$, $wf(\Gamma, \alpha)$ $(\Gamma, \alpha) \lhd^{\theta}_{n} d$ *y* $(\Delta, s) \in {}^{n}\mathcal{T}^{\theta'}(f_{\perp\!\!\!\perp}\, d)$, *entonces*

$$(\Gamma \cup \Delta \cup \{p \mapsto \alpha\}, p :: s) \in {}^{n}\mathcal{T}^{\theta \to \theta'}(f)$$

con p *tal que* $(\Gamma \cup \Delta) \bowtie \{p \mapsto \alpha\}$.

Demostración. Tomemos $\Gamma' = \Gamma \cup \Delta \cup \{p \mapsto \alpha\}$; y probemos que

$$(n, (\Gamma', p :: s)) \in \mathcal{R}^{\theta \to \theta'}_{P_\perp}(f)^{\perp} \ .$$

Supongamos un realizador primitivo tal que $(m + 1, (\Gamma'', \hat{\alpha})) \in \mathcal{R}^{\theta \to \theta'}_{P_\perp}(f)$ con $m + 1 \leqslant n$ y tal que $wf(\Gamma'', \hat{\alpha})$, $wf(\Gamma', p :: s)$ y $\Gamma' \bowtie \Gamma''$, y queremos probar $(\Gamma' \cup \Gamma'', \hat{\alpha}, p :: s) \in \perp\!\!\!\perp_{m+1}$. Por la definición 78 que extiende nuestra relación tenemos que existe una $f' : [\![\, \theta \to \theta' \,]\!]^{v}$ tal que $f = \iota_{\uparrow}(f')$, además por la definición de realizador primitivo para el tipo flecha sabemos $\hat{\alpha} = (\texttt{Grab} \triangleright i, \eta)$ para algún $i \in \mathit{Code}$, $\eta \in \mathit{MEnv}$. Luego por anti-ejecución queremos probar $(\Gamma' \cup \Gamma'', i, p :: \eta, s) \in \perp\!\!\!\perp_m$.

Utilizando todas las suposiciones sobre realizadores y tests bien formados, y la compatibilidad entre heaps, probamos $\Gamma'' \cup \Gamma \bowtie \{p \mapsto \alpha\}$ y $\Gamma'' \bowtie \Gamma$. Además haciendo uso de que nuestras relaciones de aproximación son step-indexed tenemos que $(\Gamma, \alpha) \lhd^{\theta}_{m} d$, luego podemos utilizar, de nuevo, que $(m + 1, (\Gamma'', (\texttt{Grab} \triangleright i, \eta)))$ es un realizador primitivo de f' y concluir $(\Gamma'' \cup \Gamma \cup \{p \mapsto \alpha\}, i, p :: \eta) \in {}^{m}\mathcal{R}^{\theta'}(f'\, d)$.

Sea $\Delta' = \Gamma'' \cup \Gamma \cup \{p \mapsto \alpha\}$, claramente vale $\Delta \bowtie \Delta'$ y $wf(\Delta')$, por lo tanto podemos actualizar nuestro objetivo y pasar a probar que $(\Delta' \cup \Delta, i, p :: \eta, s) \in \perp\!\!\!\perp_m$. Por lo tanto, utilizando que tenemos un realizador $(m, (\Gamma'' \cup \Gamma \cup \{p \mapsto \alpha\}, i, p :: \eta))$ y un test $(m, (\Delta, s))$ para $f'\, d = f_{\perp\!\!\!\perp}\, d$, y completar la prueba. Notar que podemos probar que $(m, (\Delta, s))$ es efectivamente un test ya que $m < n$. ■

A continuación podemos utilizar este lema para probar que la instrucción $\texttt{Push}\ n \triangleright i$ aproxima operacionalmente a la función continua $f \circ_{app} (\_ \swarrow n)$, siempre que $n < |\eta|$.

Lema 72. *Sea* $i \lhd^{\pi,\theta\to\theta'} f$, *entonces* $\mathtt{Push}\ n \triangleright i \lhd^{\pi,\theta'} f \circ_{app} (\_ \swarrow n)$, *cuando* $n < |\eta|$, *y* $\pi \cdot n = \theta$.

Demostración. Tomemos un $n \in \mathbb{N}$ y supongamos $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$ y $wf(\Gamma)$ luego queremos probar que $(\Gamma, (\mathtt{Push}\ n \triangleright i, \eta)) \lhd^{\theta'}_{n} (f\,\rho)_{\perp\!\!\!\perp} (\rho \swarrow n)$. Tomamos un test tal que $(\Delta, s) \in {}^{m+1}\mathcal{T}^{\theta'}((f\,\rho)_{\perp\!\!\!\perp} (\rho \swarrow n))$ donde se cumple $wf(\Delta, s)$ y $\Gamma \bowtie \Delta$, con el fin de probar $(\Gamma \cup \Delta, \mathtt{Push}\ n \triangleright i, \eta, s) \in \perp\!\!\!\perp_{m+1}$ cuando $m+1 \leqslant n$. Usando anti-ejecución probaremos $(\Gamma \cup \Delta, i, \eta, p :: s) \in \perp\!\!\!\perp_{m}$.

Por step-indexed tenemos $(\Gamma, \eta) \lhd^{\pi}_{m} \rho$, además usando el lema 65 obtenemos $(\Gamma, \Gamma\,p) \in {}^{m}\mathcal{R}^{\theta}(\rho \swarrow n)$. Luego podemos utilizar el lema anterior 71 y construirnos un nuevo test $(\Gamma \cup \Delta, p :: s) \in {}^{m}\mathcal{T}^{\theta\to\theta'}(f\,\rho)$. Tenemos además que $(m, (\Gamma, (i, \eta)))$ es un realizador de $f\,\rho$, luego combinando este mismo con el test reciente podemos concluir la prueba. ■

Lema 73. $\mathtt{Access}\ n \lhd^{\pi,\pi\cdot n} \_ \swarrow n$, *cuando* $n < |\eta|$.

Demostración. Tomemos un $n' \in \mathbb{N}$ y supongamos $\pi \cdot n = \theta$, queremos probar $(\Gamma, (\mathtt{Access}\ n, \eta)) \lhd^{\theta}_{n'} (\rho \swarrow n)$, suponiendo además que $(\Gamma, \eta) \lhd^{\pi}_{n'} \rho$ y $wf(\Gamma)$. Tomamos un test, $(\Delta, s) \in {}^{m+1}\mathcal{T}^{\theta}(\rho \swarrow n)$ tal que $m+1 \leqslant n$, $wf(\Delta, s)$ y $\Gamma \bowtie \Delta$, y queremos probar $(\Gamma \cup \Delta, \mathtt{Access}\ n, \eta, s) \in \perp\!\!\!\perp_{m+1}$, luego por anti-ejecución tenemos que demostrar $(\Gamma \cup \Delta, \Gamma\,p, \sharp p :: s) \in \perp\!\!\!\perp_{m}$ con $p = \eta \cdot n$.

Utilizamos el lema 65 para obtener $(\Gamma, \Gamma\,p) \lhd^{\theta}_{m} (\rho \swarrow n)$, combinando esta aproximación, con el test obtenido gracias a la propiedad de step-indexed, $(\Delta, s) \in {}^{m}\mathcal{T}^{\theta}(\rho \swarrow n)$ podemos obtener $(\Gamma \cup \Delta, \Gamma\,p, s) \in \perp\!\!\!\perp_{m}$. Luego completamos la prueba haciendo uso de la regla O_2. ■

Lema 74. *Si* $(\Delta, s) \in {}^{n}\mathcal{T}^{\theta_1}(\pi_1 {}_{\perp\!\!\!\perp}\, d)$ *tal que* $wf(\Delta, s)$ *y* $\Gamma \bowtie \Delta$, *entonces*

$$(\Gamma \cup \Delta, \pi_1 :: s) \in {}^{n}\mathcal{T}^{\theta_1\times\theta_2}(d)\ .$$

Demostración. Sea $m \in \mathbb{N}$, tomemos $(m+1, (\Gamma', \alpha')) \in \mathcal{R}^{\theta_1\times\theta_2}_{p_\perp}(d)$ el realizador primitivo tal que $wf(\Gamma', \alpha')$, $wf(\Gamma \cup \Delta, \pi_1 :: s)$ y $\Gamma' \bowtie \Gamma \cup \Delta$ y queremos probar $(\Gamma' \cup \Gamma \cup \Delta, \alpha', \pi_1 :: s) \in \perp\!\!\!\perp_{m+1}$, con $m+1 \leqslant n$. Además tenemos que existe un par (d_1, d_2) tal que $(m+1, (\Gamma', \alpha')) \in \mathcal{R}^{\theta_1\times\theta_2}_{p}(((d_1, d_2)))$ y $d = \iota_{\uparrow}((d_1, d_2))$, por lo tanto $\alpha' = (\mathtt{Pair}\ (i, i'), \eta)$, luego por anti-ejecución debemos probar $(\Gamma' \cup \Delta, i, \eta, s) \in \perp\!\!\!\perp_{m}$, donde además podemos probar que los punteros de $dom(\Gamma)$ son irrelevantes y podemos prescindir de Γ en la configuración utilizando la regla O_1.

Para completar la prueba solo resta notar que el realizador primitivo que supusimos nos permite obtener $(\Gamma', i, \eta) \in {}^{m}\mathcal{R}^{\theta_1}(d_1)$, el cual podemos

combinar con el test $(\Delta, s) \in {}^{m}\mathcal{T}^{\theta_1}(d_1)$, que es resultado de utilizar que el conjunto de test es step-indexed. ■

Este lema nos permite probar que la instrucción $\texttt{Fst}\ n$ aproxima operacionalmente a la función continua $\rho \mapsto \pi_1 \circ_k (\rho \swarrow n)$.

Lema 75. *Sea* $n \in \mathbb{N}$ *tal que* $n < |\eta|$ *y* $\pi \cdot n = \theta_1 \times \theta_2$, *entonces*

$$\texttt{Fst}\ n \lhd^{\pi,\theta_1} \pi_1 \circ_k (\_ \swarrow n) \ .$$

Demostración. Supongamos $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$ y $wf(\Gamma)$, y probemos

$$(\Gamma, (\texttt{Fst}\ n, \eta)) \lhd^{\theta_1}_{n} (\pi_1)_{\perp\!\!\!\perp}(\rho \swarrow n) \ .$$

Tomemos un test $(\Delta, s) \in {}^{m+1}\mathcal{T}^{\theta_1}((\pi_1)_{\perp\!\!\!\perp}(\rho \swarrow n))$, tal que $wf(\Gamma, (\texttt{Fst}\ n, \eta))$, $wf(\Delta, s)$, y $\Gamma \bowtie \Delta$, y probemos que $(\Gamma \cup \Delta, \texttt{Fst}\ n, \eta, s) \in \perp\!\!\!\perp_{m+1}$, con $m + 1 \leqslant n$. Por anti-ejecución y la regla OP_1 tenemos que probar

$$(\Gamma \cup \Delta, \Gamma\, p, \pi_1 :: s) \in \perp\!\!\!\perp_m \ .$$

Utilizando el lema 74 sobre el test supuesto obtenemos

$$(\Gamma \cup \Delta, \pi_1 :: s) \in {}^{n}\mathcal{T}^{\theta_1 \times \theta_2}(\rho \swarrow n) \ .$$

Además usando el lema 65 tenemos $(\Gamma, \Gamma\, p) \in {}^{m}\mathcal{R}^{\theta_1 \times \theta_2}(\rho \swarrow n)$. Luego podemos combinar este test y realizador para completar la prueba. ■

Los últimos dos lemas tienen su versión análoga para el caso de la segunda proyección, donde además las pruebas son equivalentes.

Lema 76. *Si* $(\Delta, s) \in {}^{n}\mathcal{T}^{\theta_2}(\pi_{2\perp\!\!\!\perp} d)$ *tal que* $wf(\Delta, s)$ *y* $\Gamma \bowtie \Delta$, *entonces*

$$(\Gamma \cup \Delta, \pi_2 :: s) \in {}^{n}\mathcal{T}^{\theta_1 \times \theta_2}(d) \ .$$

Lema 77. *Sea* $n \in \mathbb{N}$ *tal que* $n < |\eta|$ *y* $\pi \cdot n = \theta_1 \times \theta_2$, *entonces*

$$\texttt{Snd}\ n \lhd^{\pi,\theta_2} \pi_2 \circ_k (\_ \swarrow n) \ .$$

Los siguientes lemas nos permiten probar que la instrucción $\texttt{BOp}_{\oplus}\ i\ i'$ aproxima operacionalmente a $m \odot m'$ siempre que i e i' aproximen a m y m' respectivamente. El resultado final lo conseguiremos enunciado dos lemas previos que nos permitan construir un test en base a otro.

Lema 78. *Si* $(\Delta, s) \in {}^{n}\mathcal{T}^{int}(\iota_{\uparrow}(m) \odot_{\perp\!\!\!\perp} d')$ *tal que* $wf(\Delta, s)$, $wf(\Gamma, (m, \eta))$ *y* $\Gamma \bowtie \Delta$ *entonces*

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\, \underline{m}\, \Box, \eta)\}, \odot p :: s) \in {}^{n}\mathcal{T}^{int}(d')$$

con $p \notin \mathtt{ptr}(\Delta)$.

Demostración. Tomemos $\overline{\Gamma} = \Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\, \underline{m}\, \Box, \eta)\}$. Supongamos el realizador primitivo $(m+1, (\Gamma', \alpha')) \in \mathcal{R}^{\mathbf{int}}_{P_\perp}(d')$ tal que $wf(\Gamma', (\underline{m'}, \eta'))$ y $\Gamma' \bowtie \overline{\Gamma}$ y probemos que $(\Gamma' \cup \overline{\Gamma}, \alpha', \odot p :: s) \in \perp\!\!\!\perp_{m+1}$, con $m+1 \leqslant n$. Utilizando el realizador primitivo podemos obtener que $d' = \iota_{\uparrow}(m')$ y además $(m+1, (\Gamma', (\underline{m'}, \eta'))) \in \mathcal{R}^{\mathbf{int}}_{P}((m'))$, es decir $\alpha = (\underline{m'}, \eta')$. Luego por anti-ejecución debemos probar $(\Gamma' \cup \overline{\Gamma}, \underline{m \odot m'}, \eta, s) \in \perp\!\!\!\perp_{m}$.

Para completar la prueba podemos construirnos el realizador primitivo $(m, (\Gamma' \cup \Gamma \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\, \underline{m}\, \Box, \eta)\}, (\underline{m \odot m'}, \eta))) \in \mathcal{R}^{\mathbf{int}}_{P_\perp}(\iota_{\uparrow}(m \odot m'))$ que nos permite combinarlo con el test $(\Delta, s) \in {}^{m}\mathcal{T}^{\mathbf{int}}(\iota_{\uparrow}(m \odot m'))$, notando que $\iota_{\uparrow}(m) \odot_{\perp\!\!\!\perp} \iota_{\uparrow}(m') = \iota_{\uparrow}(m \odot m')$. ■

Lema 79. *Si* $(\Gamma, i', \eta) \in {}^{n}\mathcal{R}^{int}(d')$ *y* $(\Delta, s) \in {}^{n}\mathcal{T}^{int}(d \odot_{\perp\!\!\!\perp} d')$ *tal que* $\Gamma \bowtie \Delta$, $wf(\Delta, s)$ *y* $wf(\Gamma, (\mathtt{BOp}_{\oplus}\, i\, i', \eta))$, *entonces*

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\, \Box\, i', \eta)\}, \odot p :: s) \in {}^{n}\mathcal{T}^{int}(d)$$

donde $p \notin \mathtt{ptr}(\Gamma \cup \Delta)$

Demostración. Sea $\overline{\Gamma} = \Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\, \Box\, i', \eta)\}$. Tomamos el realizador primitivo $(n'+1, (\Gamma', \alpha)) \in \mathcal{R}^{\mathbf{int}}_{P_\perp}(d)$ tal que $wf(\Gamma', \alpha)$, $wf(\overline{\Gamma}, \odot p :: s)$ y $\Gamma' \bowtie \overline{\Gamma}$, y probemos que $(\Gamma' \cup \overline{\Gamma}, \alpha, \odot p :: s) \in \perp\!\!\!\perp_{n'+1}$ con $n'+1 \leqslant n$. Por el realizador primitivo tenemos que $d = \iota_{\uparrow}(m)$ y $(\Gamma', \alpha) \in {}^{n'+1}\mathcal{R}^{\mathbf{int}}_{P}(\underline{m})$, con $\alpha = (\underline{m}, \eta')$. Luego por anti-ejecución es suficiente con demostrar que

$$(\Gamma' \cup \overline{\Gamma} \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\, \underline{m}\, \Box, \eta')\}, (i', \eta), \odot p :: s) \in \perp\!\!\!\perp_{n'}\ .$$

Utilizando el lema anterior podemos construirnos el test

$$(\Gamma' \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus}\, \underline{m}\, \Box, \eta')\}, \odot p :: s) \in {}^{n'}\mathcal{T}^{\mathbf{int}}(m')$$

y combinarlo con $(\Gamma, i', \eta) \in {}^{n'}\mathcal{R}^{\mathbf{int}}(d')$ para completar la prueba. ■

Lema 80. *Si* $i \lhd^{\pi, int} d$ *y* $i' \lhd^{\pi, int} d'$ *entonces* $\mathtt{BOp}_{\oplus}\, i\, i' \lhd^{\pi, int} d \circ_{\odot} d'$.

Demostración. Supongamos $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$ y $wf(\Gamma)$, y probemos

$$(\Gamma, (\mathtt{BOp}_{\oplus}\ i\ i', \eta)) \lhd^{\mathbf{int}}_{n} (d \circ_{\odot} d')\rho\ .$$

Tomemos un test $(\Delta, s) \in {}^{m+1}\mathcal{T}^{\theta_1}((d \circ_{\odot} d')\rho)$, tal que $wf(\Gamma, (\mathtt{Fst}\ n, \eta))$, $wf(\Delta, s)$, y $\Gamma \bowtie \Delta$, y probemos que $(\Gamma \cup \Delta, \mathtt{BOp}_{\oplus}\ i\ i', \eta, s) \in \perp\!\!\!\perp_{m+1}$, con $m+1 \leqslant n$. Por anti-ejecución tenemos que probar

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus} \square i', \eta)\}, (i, \eta), \odot p :: s) \in \perp\!\!\!\perp_{m}\ .$$

Utilizando el lema anterior tenemos

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{BOp}_{\oplus} \square i', \eta)\}, \odot p :: s) \in {}^{m}\mathcal{T}^{\mathbf{int}}(d\,\rho)$$

y podemos concluir la prueba combinando este ultimo test con el hecho de que (Γ, i, η) es un realizador de $d\,\rho$. ■

Utilizando una estrategia similar a la que usamos recientemente con los operadores binarios podemos probar los lemas relacionados con la expresión condicional.

Lema 81. *Si* $(\Gamma, i', \eta) \in {}^{n}\mathcal{R}^{\theta}(d')$ *y* $(\Delta, s) \in {}^{n}\mathcal{T}^{\theta}(d')$ *tal que* $wf(\Delta, s)$, $\Gamma \bowtie \Delta$ *y* $wf(\Gamma, (\mathtt{IfZ}\ i\ i'\ i'', \eta))$, *entonces*

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{IfZ} \square (i', i''), \eta)\}, ?p :: s) \in {}^{n}\mathcal{T}^{\boldsymbol{int}}(\iota_{\uparrow}(0))$$

donde $p \notin \mathtt{ptr}(\Gamma \cup \Delta)$.

Demostración. Sea $\overline{\Gamma} = \Gamma \cup \Delta \cup \{p \mapsto (\mathtt{IfZ} \square (i', i''), \eta)\}$. Supongamos el realizador primitivo $(m+1, (\Gamma', \underline{0}, \eta')) \in \mathcal{R}^{\mathbf{int}}_{p_\perp}(\iota_{\uparrow}(0))$ tal que $wf(\Gamma', (\underline{0}, \eta'))$, $wf(\overline{\Gamma}, ?p :: s)$ y $\Gamma' \bowtie \overline{\Gamma}$, y probemos que $(\Gamma' \cup \overline{\Gamma}, (\underline{0}, \eta'), ?p :: s) \in \perp\!\!\!\perp_{m+1}$. Luego por anti-ejecución y utilizando la regla O_1 es suficiente con demostrar que $(\Gamma \cup \Delta, (i', \eta), s) \in \perp\!\!\!\perp_{m}$. Por lo tanto, completamos la prueba combinando el realizador y test para el valor denotacional d'. ■

El lema anterior nos permitía construir un test en el caso de que valor denotacional para la condición del *ifz* era cero, de forma análoga podemos enunciar y probar el caso en el que el valor denotacional es distinto de cero.

Lema 82. *Si* $(\Gamma, i'', \eta) \in {}^{n}\mathcal{R}^{\theta}(\iota_{\uparrow}(d''))$ *y* $(\Delta, s) \in {}^{n}\mathcal{T}^{\theta}(d'')$ *tal que* $wf(\Delta, s)$, $wf(\Gamma, (\mathtt{IfZ}\ i\ i'\ i'', \eta))$ *y* $\Gamma \bowtie \Delta$ *entonces*

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{IfZ} \square (i', i''), \eta)\}, ?p :: s) \in {}^{n}\mathcal{T}^{\boldsymbol{int}}(z)$$

donde $p \notin \mathtt{ptr}(\Gamma \cup \Delta)$ *y* $z \neq \iota_{\uparrow}(0)$.

El siguiente lema es una generalización que combina los últimos dos resultados previos y que nos permite construir un nuevo test.

Lema 83.
Si $(\Gamma, i', \eta) \in {}^{n}\mathcal{R}^{\theta}(d')$ *y* $(\Gamma, i'', \eta) \in {}^{n}\mathcal{R}^{\theta}(d'')$, *además tenemos el test*

$$(\Delta, s) \in {}^{n}\mathcal{T}^{\theta}(\textit{ifz}\,(d)\,(d')\,(d''))$$

tal que $wf(\Gamma, (\mathtt{IfZ}\ i\ i'\ i'', \eta))$, $wf(\Delta, s)$ *y* $\Gamma \bowtie \Delta$ *entonces*

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{IfZ}\,\square\,(i', i''), \eta)\}, ?p :: s) \in {}^{n}\mathcal{T}^{\textit{int}}(d)$$

donde $p \notin \mathtt{ptr}(\Gamma \cup \Delta)$.

Demostración. Supongamos $(m+1, (\Gamma', i, \eta')) \in \mathcal{R}^{\mathbf{int}}_{P_\perp}(d)$ realizador primitivo tal que $wf(\Gamma', (i, \eta'))$, $wf(\overline{\Gamma}, ?p :: s)$ y $\Gamma' \bowtie \overline{\Gamma}$, y probemos que

$$(\Gamma' \cup \overline{\Gamma}, (i, \eta'), ?p :: s) \in \perp\!\!\!\perp_{m+1} .$$

Luego existe un $\hat{d}$ tal que $d = \iota_{\uparrow}(\hat{d})$ y además $(m+1, (\Gamma', i, \eta')) \in \mathcal{R}^{\mathbf{int}}_{P}(\hat{d})$. Si suponemos que $\hat{d} = 0$ completamos la prueba utilizando el lema 81, en caso contrario utilizamos el lema 82. ■

Lema 84. *Si* $i \lhd^{\pi,\textit{int}} d$, $i' \lhd^{\pi,\theta} d'$ *y* $i'' \lhd^{\pi,\theta} d''$ *entonces*

$$\mathtt{IfZ}\ i\ i'\ i'' \lhd^{\pi,\theta} \textit{ifz}\,(d)\,(d')\,(d'') .$$

Demostración. Supongamos $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$ y $wf(\Gamma)$, y probemos

$$(\Gamma, (\mathtt{IfZ}\ i\ i'\ i'', \eta)) \lhd^{\theta}_{n} (\mathbf{ifz}\,(d)\,(d')\,(d''))\rho .$$

Tomemos un test $(\Delta, s) \in {}^{m+1}\mathcal{T}^{\theta}((\mathbf{ifz}\,(d)\,(d')\,(d''))\rho)$, tal que $\Gamma \bowtie \Delta$, $wf(\Delta, s)$ y $wf(\Gamma, (\mathtt{Fst}\ n, \eta))$, y probemos que $(\Gamma \cup \Delta, \mathtt{IfZ}\ i\ i'\ i'', \eta, s) \in \perp\!\!\!\perp_{m+1}$, con $m+1 \leqslant n$. Por anti-ejecución tenemos que probar

$$(\Gamma \cup \Delta \cup \{p \mapsto (\mathtt{IfZ}\,\square\,(i', i''), \eta)\}, (i, \eta), ?p :: s) \in \perp\!\!\!\perp_{m} .$$

Utilizando el test $(\Delta, s) \in {}^{m+1}\mathcal{T}^{\theta}((\mathbf{ifz}\,(d)\,(d')\,(d''))\rho)$ en conjunto con el lema 83 podemos construirnos un nuevo test que al combinarlo con la aproximación $i \lhd^{\pi,\mathbf{int}} d$ nos permite completar la prueba. ■

A continuación probaremos el último de una serie de lemas necesarios para probar la propiedad fundamental de las relaciones lógicas. Primero probaremos que si una instrucción i aproxima operacionalmente a un valor denotacional $d : [\![\theta :: \pi]\!] \to [\![\theta]\!]$ entonces la instrucción $\mathtt{Rec}\, i$ aproxima al punto fijo $\mathbf{Fix}(d) : [\![\pi]\!] \to [\![\theta]\!]$. Luego utilizando este resultado probamos casi inmediatamente que la instrucción $\mathtt{Let}\ (\mathtt{Rec}\, i) \triangleright i'$ aproxima operacionalmente a la función continua $\mathbf{let}\,(\mathbf{Fix}(d))\,(d') : [\![\pi]\!] \to [\![\theta']\!]$, cuando i' aproxima a $d' : [\![\theta :: \pi]\!] \to [\![\theta']\!]$ y i aproxima a $d : [\![\theta :: \pi]\!] \to [\![\theta]\!]$.

Lema 85. *Si* $i \lhd^{\theta::\pi,\theta} d$, *entonces* $\mathtt{Rec}\, i \lhd^{\pi,\theta} \boldsymbol{Fix}(d)$.

Demostración. La prueba procederá por inducción en $n \in \mathbb{N}$, donde tenemos que probar $(\Gamma, (\mathtt{Rec}\, i, \eta)) \lhd^{\theta}_{n} \mathbf{Fix}(d)\,\rho$ suponiendo $wf(\Gamma)$ y $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$ valen.

- Si $n = 0$, entonces tomando cualquier test $(\Delta, s) \in {}^{0}\mathcal{T}^{\theta}(\mathbf{Fix}(d)\,\rho)$ y suponiendo $wf(\Gamma, (\mathtt{Rec}\, i, \eta))$, $wf(\Delta, s)$ y $\Gamma \bowtie \Delta$ entonces

  $$(\Gamma \cup \Delta, \mathtt{Rec}\, i, \eta, s) \in \perp\!\!\!\perp_{0} \ .$$

  Pero esto es directo por definición.

- Si $n = m + 1$, tomamos un test $(\Delta, s) \in {}^{m'+1}\mathcal{T}^{\theta}(\mathbf{Fix}(d)\,\rho)$ y suponiendo $wf(\Gamma, (\mathtt{Rec}\, i, \eta))$, $wf(\Delta, s)$ y $\Gamma \bowtie \Delta$ entonces $(\Gamma \cup \Delta, \mathtt{Rec}\, i, \eta, s) \in \perp\!\!\!\perp_{m'+1}$, cuando $m' + 1 \leqslant m + 1$. Por anti-ejecución tenemos que probar

  $$(\Gamma \cup \{p \mapsto (\mathtt{Rec}\, i, \eta)\} \cup \Delta, i, p :: \eta, s) \in \perp\!\!\!\perp_{m'}$$

  Por hipótesis inductiva tenemos que $(\Gamma, (\mathtt{Rec}\, i, \eta)) \lhd^{\theta}_{m} \mathbf{Fix}(d)\,\rho$, además tenemos que vale $(\Gamma, \eta) \lhd^{\pi}_{m'} \rho$ y por lo tanto utilizando el lema 65 podemos construirnos la aproximación

  $$(\Gamma \cup \{p \mapsto (\mathtt{Rec}\, i, \eta)\}, p :: \eta) \lhd^{\theta::\pi}_{m'} (\mathbf{Fix}(d)\,\rho, \rho) \ .$$

  Luego utilizando que i aproxima a d tenemos que

  $$(\Gamma \cup \{p \mapsto (\mathtt{Rec}\, i, \eta)\}, i, p :: \eta) \lhd^{\theta}_{m'} d\,(\mathbf{Fix}(d)\,\rho, \rho) \ .$$

  Notar que $\mathbf{Fix}(d)\,\rho = d\,(\mathbf{Fix}(d)\,\rho, \rho)$. Luego podemos completar la prueba utilizando el test $(\Delta, s) \in {}^{m'}\mathcal{T}^{\theta}(d\,(\mathbf{Fix}(d)\,\rho, \rho))$.

■

Lema 86. *Si* $i \lhd^{\theta :: \pi, \theta} d$ *y* $i' \lhd^{\theta :: \pi, \theta'} d'$, *entonces*

$$\mathtt{Let}\ (\mathtt{Rec}\, i) \triangleright i' \lhd^{\pi, \theta'} \textbf{\textit{let}}\, (\textbf{\textit{Fix}}(d))\, (d') \ .$$

Demostración. Supongamos $(\Gamma, \eta) \lhd^{\pi}_{n} \rho$ y $wf(\Gamma)$, tomamos un test $(\Delta, s) \in {}^{m+1}\mathcal{T}^{\theta'}(\textbf{let}\, (\textbf{Fix}(d))\, (d'))$ tal que $wf(\Gamma, (\mathtt{Let}\ (\mathtt{Rec}\, i) \triangleright i', \eta))$, $wf(\Delta, s)$ y $\Gamma \bowtie \Delta$ y queremos probar, luego de aplicar anti-ejecución que

$$(\Gamma[p \mapsto (\mathtt{Rec}\, i, \eta)] \cup \Delta, i', p :: \eta, s) \in \perp\!\!\!\perp_m$$

Por hipótesis y el lema anterior obtenemos $(\Gamma, (\mathtt{Rec}\, i, \eta)) \lhd^{\theta}_{m} \textbf{Fix}(d)\rho$. Luego además combinando esta aproximación con la aproximación de entornos concluimos mediante el lema 65 que

$$(\Gamma \cup \{ p \mapsto (\mathtt{Rec}\, i, \eta) \}, p :: \eta) \lhd^{\theta :: \pi}_{m} (\textbf{Fix}(d)\, \rho, \rho) \ .$$

Notar que $\textbf{let}\, (\textbf{Fix}(d))\, (d')\, \rho = d'\, (\textbf{Fix}(d)\, \rho, \rho)$. Por lo tanto combinando la ultima aproximación construida y una de nuestras hipótesis tenemos

$$(\Gamma \cup \{ p \mapsto (\mathtt{Rec}\, i, \eta) \}, (i', p :: \eta)) \lhd^{\theta'}_{m} d'\, (\textbf{Fix}(d)\, \rho, \rho) \ .$$

Finalmente, utilizando el test $(\Delta, s) \in {}^{m}\mathcal{T}^{\theta'}(\textbf{let}\, (\textbf{Fix}(d))\, (d'))$ completamos la prueba. ■

Finalmente probaremos que la compilación de cualquier programa bien tipado aproxima a su valor denotacional. La prueba es el resultado de utilizar inducción en la derivación del juicio de tipado y utilizar los lemas anteriores.

Teorema 16. *Si* $\pi \vdash t : \theta$ *entonces* $(\!|\, t \,|\!) \lhd^{\pi, \theta} [\![\, t \,]\!]_{\pi, \theta}$.

Utilizaremos la propiedad fundamental para probar la corrección del compilador cuando el valor denotacional de una expresión es igual a $\perp$; en ese caso probaremos que el código resultado de compilar tal expresión ocasiona que la máquina de infinita cantidad de transiciones. Es interesante notar que a diferencia de la relación de aproximación denotacional (que nos permitía probar la corrección con respecto a un tipo básico particular), la aproximación operacional que definimos nos permite enunciar y probar la corrección con respecto a cualquier tipo. Esto se debe a que sin importar el tipo de nuestra expresión, el comportamiento de la máquina debería ser el mismo.

Para probar que la máquina da una cantidad infinita de transiciones vamos a probar que la configuración inicial pertenece al conjunto de observaciones $\perp\!\!\!\perp_n$ para cualquier $n \in \mathbb{N}$. Recordemos que si una configuración pertenece a $\perp\!\!\!\perp_n$, significa que la máquina (partiendo de esa configuración) realiza al menos n transiciones.

Teorema 17. *Sea* t *una expresión y* Γ *un heap. Si* $\vdash t : \theta$, $[\![t]\!]_{[],\theta} \diamond = \bot$ *y* $wf(\Gamma)$, *entonces* $(\Gamma, (\!| t |\!), [], []) \longmapsto^{\infty}$

Demostración. Utilizando el teorema 16 obtenemos $(\!| t |\!) \lhd^{[],\theta} [\![t]\!]_{[],\theta}$. Ya que $wf(\Gamma)$, luego trivialmente $(\Gamma, []) \lhd^{[]}_n \diamond$ para cualquier $n \in \mathbb{N}$ y por lo tanto tenemos $(\Gamma, ((\!| t |\!), [])) \lhd^{\theta}_n \bot$. Por la definición 78 que nos permite extender los realizadores primitivos es directo que $(\Gamma, []) \in {}^{n}\mathcal{T}^{\theta}(\bot)$, luego combinando este test con la aproximación anterior podemos concluir que $(\Gamma, (\!| t |\!), [], []) \in \perp\!\!\!\perp_n$. ■

Considerando el programa del ejemplo 5.6, hemos probado que este computa a la función matemática $\mathtt{fib}$ siempre y cuando el argumento proveído sea mayor o igual que cero. Utilizando el ultimo teorema 17 podemos probar además que sucede en el caso de que el argumento sea menor que cero. Teniendo en cuenta interpretamos el valor denotacional $\bot$ como aquel que no nos brinda información, podemos probar que si $\mathtt{fib}\, n = \bot$ para cualquier $n < 0$, entonces nuestra expresión de alto nivel implementa exactamente esta función, más aun utilizando la corrección de compilador podemos concluir que el programa resultado de compilar tal expresión también implementa exactamente a esta función. Notar que esto no vale si por ejemplo quisiéramos probar que nuestro programa de alto nivel implementa la función $\mathtt{fib}$, pero tal que $\mathtt{fib}\, n = 0$, para cualquier $n < 0$.

6

CONCLUSIONES

En el desarrollo general del doctorado hemos aplicado herramientas matemáticas sofisticadas para estudiar propiedades de lenguajes de programación y demostrar la corrección de compiladores. Un contenido común a lo largo de los capítulos fue el uso de sistemas de tipos y relaciones lógicas como estrategia de prueba. Las principales contribuciones del trabajo son: (I) la extensión de alguna de esas técnicas a construcciones que no habían sido consideradas anteriormente (Cap. 4 donde analizamos un lenguaje con subtipado), (II) la adaptación de biortogonalidad a una semántica extrínseca (Cap. 4) y (III) la definición de un compilador correcto para un lenguaje con estrategia de evaluación lazy (Cap. 5). A continuación, hacemos comentarios más detallados de cada capítulo.

El capítulo 3 engloba los primeros pasos de la investigación que comenzó con el estudio del artículo de Launchbury[25] donde se presenta un lenguaje con estrategia de evaluación lazy y define semánticas denotacional y operacional para las cuales prueba su corrección y adecuación. Durante este aprendizaje encontramos que la definición de semántica para heaps presentada por Launchbury desembocaba en un contra-ejemplo para la prueba de corrección. Nuestro desarrollo entonces se basó en dar una definición correcta de la semántica para heaps (interpretando la verdadera intención de la definición original) y reemplazar la semántica operacional original por la presentada por Sestoft[46] cuya definición permite probar efectivamente que las configuraciones involucradas en una evaluación son distintivamente nombradas. Finalmente, probamos de manera exhaustiva todos los resultados expuesto en el artículo, incluida la prueba del teorema de corrección.

El desarrollo presentado en el capítulo 4 comenzó como una excusa para aprender las técnicas de biortogonalidad, step-indexing y la mecanización en Coq de la meta-teoría de lenguajes de programación. En determinado momento notamos que la mayor parte de la literatura sobre biortogonalidad para lenguajes tipados consideraba solo semántica intrínseca y decidimos intentar explorar la estrategia de prueba utilizando semántica extrínseca; lo cual derivo en la prueba de adecuación con respecto a esta semántica. Además, la intención fue probar la correspondencia entre las semánticas

denotacionales intrínseca y extrínseca con el objetivo de poder transferir los resultados de adecuación con respecto a una semántica en base a otra; Reynolds[38, 39] propone una estrategia utilizando relaciones lógicas para un lenguaje call-by-name, nosotros adaptamos la estrategia para un lenguaje call-by-value y dimos una mecanizamos en Coq. Concretamente, en una primera etapa definimos un lenguaje muy simple y presentamos las herramientas matemáticas utilizadas por Reynolds para probar el teorema de bracketing entre las semánticas intrínseca y extrínseca; además introducimos el uso de biortogonalidad para probar adecuación. Posteriormente, ejemplificamos la modularidad de las técnicas mediante la extensión del lenguaje enriqueciendo el sistema de tipos con nuevas construcciones y subtipado. Gran parte de las pruebas se mantiene invariante: la extensión solo agrega algunos casos que no interfieren con los casos inductivos de las pruebas originales.

MECANIZACIÓN La formalización de estos resultados fue uno de los enfoques principales del desarrollo, en ese sentido es interesante mencionar que la mecanización en Coq de todos estos resultados para el lenguaje completo tomó alrededor de 12000 líneas (contando solamente nuestro código y no las librerías externas utilizadas). Además, podemos comparar el incremento en lineas de código para módulos específicos considerando el lenguaje simple contra el extendido; notar que el factor de incremento es casi el mismo.

	Especificaciones		Pruebas	
Módulo	Simple	Extendido	Simple	Extendido
Teorema de Bracketing	340	550	1000	2000
Aproximación Operacional	170	220	550	1100
Aproximación Denotacional	230	320	540	1000

Finalmente, en el capítulo 5 combinamos realizabilidad y biortogonalidad para probar la corrección de un compilador hacia una versión extendida con pares, operaciones binarias y recursión de la máquina de Sestoft. Esto implica una extensión notable respecto al trabajo original de Rodríguez [41]. La principal dificultad se encontraba en definir los realizadores y tests ya que tanto las clausuras como los stacks necesitan del heap para estar bien formados; por lo tanto la relación de satisfactibilidad, que nos induce los operadores ortogonales, termina siendo más compleja que

el simple hecho de pertenecer a un conjunto de observaciones como en el capítulo 4. Hasta donde sabemos este es el primer desarrollo que utiliza esta técnica para un lenguaje con estrategia de evaluación lazy. A todo este desarrollo lo mecanizamos en Coq.

TRABAJOS FUTUROS

EN GENERAL Un idea general sobre la utilización de biortogonalidad que parece desafiante pensar es qué tan generalizable es la técnica con respecto a las semánticas. Observando detenidamente los desarrollos, con principal atención en las mecanizaciones, de los capítulos 4 y 5 se puede notar que para cualquier construcción sintáctica, supongamos la *aplicación*, la prueba de la propiedad fundamental de las relaciones lógicas en el contexto de cada capítulo son relativamente parecidas en la parte técnica relacionada con biortogonalidad y "sólo" difiere en los tecnicismos semánticos de la construcción sintáctica; es decir, en el caso de la semántica operacional al aplicar anti-ejecución para continuar con la prueba lo hacemos según la regla operacional de la aplicación y luego tal vez lo que reste para completar la prueba es probar equivalencias entre algunos componentes de la configuración involucrada. De igual manera, si consideramos el aspecto denotacional podemos pensar que tenemos un morfismo que es la interpretación de la aplicación en la categoría que corresponda. Por lo tanto, ¿es posible entonces generalizar las pruebas de la propiedad fundamental de las relaciones lógicas de manera que solo requerimos instanciar la categoría que utilizamos para definir la parte denotacional y qué pasos operacionales evalúan cada construcción junto con las posibles pruebas de equivalencia de las componentes de las configuraciones? Es una pregunta difícil e interesante.

SOBRE LA MECANIZACIÓN La implementación en general de las mecanizaciones tiene mucho margen de mejora. Esto es importante considerando que uno de los beneficios de la naturaleza constructiva de nuestras mecanizaciones en Coq, y que apenas investigamos, es que deberíamos ser capaces de obtener los "objetos" que decimos, en teoremas y pruebas, que existen. Por ejemplo, probamos que la semántica operacional es adecuada con respecto a la semántica denotacional; es decir que si la semántica denotacional de una expresión es igual a un valor, entonces existe una derivación. Lamentablemente, en parte seguramente por nuestra implementación

poco eficiente, construir la derivación parece llevar mucho tiempo y estaría interesante entender el por qué.

SOBRE DOMINIOS Durante la mecanización de la semántica denotacional extrínseca del capítulo 4 ganamos una experiencia considerable con la formalización de dominios realizada por Benton et al. [2]. Parece interesante entonces explotar esta librería para mecanizar Normalización por Evaluación utilizando teoría de dominios y contrastarla con la formalización realizada por Wieczorek et al. [50].

CORRECCIÓN SEMÁNTICA OPERACIONAL LAZY El desarrollo del capítulo 3 es el único de este trabajo que no posee formalización, es ese sentido parece una buena idea explorar la librería alternativa de dominios desarrollada por Dockins[14], para dar una mecanización; un resultado interesante de esta mecanización puede ser la comparación con la librería de dominios de Benton et la. [2] utilizada durante este trabajo. Esto además permitiría contrastar nuestra propuesta con la de Breitner[7].

COHERENCIA Y SUBTIPADO Respecto del capítulo 4 es interesante explorar la utilización de una metodología más general para extender el lenguaje. Delaware et al. [13] reconocen que uno de los obstáculos para la mecanización de la meta-teoría de lenguajes se encuentra en la dificultad para extender pruebas; puede ser interesante considerar la aplicación de su framework MTC para nuestras estrategias de prueba. Además, podemos considerar más extensiones a nuestro lenguaje como, co-productos, registros y tipos recursivos. En ese sentido, extender con intersección o polimorfismo el sistema de tipos parece una posibilidad más intrincada como menciona Reynolds[38, 39].

BIORTOGONALIDAD LAZY En el capítulo 5 presentamos la corrección de un compilador para un lenguaje con recursión y estrategia de evaluación lazy, nos es de muchísimo interés continuar con esta linea de trabajo con la idea de extender el lenguaje hasta alcanzar un fragmento considerable de System F_C, el lenguaje núcleo del compilador GHC[49], con el fin de aproximarnos a la corrección del popular compilador de Haskell; la mecanización de la máquina abstracta STG[23] utilizada por este compilador ha sido estudiada por [17, 34], en ese sentido un objetivo es acercar nuestra máquina abstracta y la máquina STG. Un posible camino para lograr esto es: (1) Refinar la noción de observación relacionada con la técnica de

biortogonalidad. En nuestra adaptación de la técnica agregamos algunas condiciones que debe satisfacer la observación para poder obtener el resultado final de corrección del compilador. Una mejora importante sería relajar esas condiciones. Por otro lado, y como ya dijimos, consideramos importante establecer una metodología general para la verificación de la satisfacción de esas condiciones. (II) La primera etapa de extensión para el lenguaje cuyo sistema de tipos soporta tipos básicos, exponenciales, co-productos y productos, es sumar la posibilidad de declarar tipos algebraicos con definiciones usando pattern-matching, para esto un primer avance es estudiar el aspecto relacionado por Sestoft[46]. Una posibilidad interesante es explorar el uso de productos y co-productos para definir tal extensión. (III) En una siguiente etapa de extensión planeamos extender el sistema de tipos con polimorfismo, esto conlleva considerar parametricidad en las relaciones lógicas; la técnica de biortogonalidad ha sido utilizada exitosamente para lidiar con parametricidad como mostraron Jaber y Tabareau [22]. (IV) Completadas las etapas anteriores tendremos definida una máquina abstracta distinta a la máquina STG[23] pero comparable. Por lo tanto el paso lógico es probar la equivalencia entre nuestra máquina abstracta y la máquina STG, dando una función de traducción entre ambas y probando la bisimulación entre ellas.

BIBLIOGRAFÍA

[1] Benton, Nick and Hur, Chung-Kil. «ble». En: *SIGPLAN Not.* 44.9 (ago. de 2009), 97-108. ISSN: 0362-1340.

[2] Nick Benton, Andrew Kennedy y Carsten Varming. «Some Domain Theory and Denotational Semantics in Coq». En: *Proceedings of the 22Nd International Conference on Theorem Proving in Higher Order Logics*. TPHOLs '09. Munich, Germany: Springer-Verlag, 2009, págs. 115-130. ISBN: 978-3-642-03358-2. DOI: 10.1007/978-3-642-03359-9_10. URL: http://dx.doi.org/10.1007/978-3-642-03359-9_10.

[3] Nick Benton, Chung-Kil Hur, Andrew Kennedy y Conor McBride. «Strongly Typed Term Representations in Coq». En: *J. Autom. Reasoning* 49.2 (2012), págs. 141-159. DOI: 10.1007/s10817-011-9219-0. URL: https://doi.org/10.1007/s10817-011-9219-0.

[4] U. Berger. «Realisability for Induction and Coinduction with Applications to Constructive Analysis». En: *Journal of Universal Computer Science* 16.18 (2010), págs. 2535-2555.

[5] Garret Birkhoff. *Lattice Theory*. Vol. 25. 1940.

[6] Joachim Breitner. «The Correctness of Launchbury's Natural Semantics for Lazy Evaluation». En: *CoRR* abs/1405.3099 (2014). arXiv: 1405.3099. URL: http://arxiv.org/abs/1405.3099.

[7] Joachim Breitner. «Lazy Evaluation: From natural semantics to a machine-checked compiler transformation». Inglés. Tesis doct. Karlsruher Institut für Technologie (KIT), 2016. 226. DOI: 10.5445/IR/1000054251. URL: http://dx.doi.org/10.5445/KSP/1000056002.

[8] Adam Chlipala. «A Certified Type-preserving Compiler from Lambda Calculus to Assembly Language». En: *SIGPLAN Not.* 42.6 (jun. de 2007), págs. 54-65. ISSN: 0362-1340. DOI: 10.1145/1273442.1250742. URL: http://doi.acm.org/10.1145/1273442.1250742.

[9] Alonzo Church. «An Unsolvable Problem of Elementary Number Theory». En: *American Journal of Mathematics* 58.2 (abr. de 1936), págs. 345-363. ISSN: 00029327. DOI: 10.2307/2371045. URL: http://dx.doi.org/10.2307/2371045.

[10] Daniel Fridlender and Alejandro Gadea and Miguel Pagano and Leonardo Rodríguez. «Biorthogonality for a Lazy language». En: *IFL 2017: 29th Symposium on the Implementation and Application of Functional Programming Languages*. Bristol, United Kindom, 2017. ISBN: 978-1-4503-6343-3. DOI: 10.1145/3205368.3205374. URL: https://doi.org/10.1145/3205368.3205374.

[11] Olivier Danvy. «From Reduction-Based to Reduction-Free Normalization». En: *Advanced Functional Programming, 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*. Ed. por Pieter W. M. Koopman, Rinus Plasmeijer y S. Doaitse Swierstra. Vol. 5832. Lecture Notes in Computer Science. Springer, 2008, págs. 66-164. ISBN: 978-3-642-04651-3. DOI: 10.1007/978-3-642-04652-0_3. URL: https://doi.org/10.1007/978-3-642-04652-0_3.

[12] Maulik A. Dave. «Compiler Verification: A Bibliography». En: *SIGSOFT Softw. Eng. Notes* 28.6 (nov. de 2003), págs. 2-2. ISSN: 0163-5948. DOI: 10.1145/966221.966235. URL: http://doi.acm.org/10.1145/966221.966235.

[13] Benjamin Delaware, Bruno C. d. S. Oliveira y Tom Schrijvers. «Meta-theory à la carte». En: *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*. Ed. por Roberto Giacobazzi y Radhia Cousot. ACM, 2013, págs. 207-218. ISBN: 978-1-4503-1832-7. DOI: 10.1145/2429069.2429094. URL: http://doi.acm.org/10.1145/2429069.2429094.

[14] Robert Dockins. «Formalized, Effective Domain Theory in Coq». En: *Interactive Theorem Proving*. Ed. por Gerwin Klein y Ruben Gamboa. Cham: Springer International Publishing, 2014, págs. 209-225. ISBN: 978-3-319-08970-6.

[15] Eric Eide y John Regehr. «Volatiles Are Miscompiled, and What to Do About It». En: *Proceedings of the 8th ACM International Conference on Embedded Software*. EMSOFT '08. Atlanta, GA, USA: ACM, 2008, págs. 255-264. ISBN: 978-1-60558-468-3. DOI: 10.1145/1450058.1450093. URL: http://doi.acm.org/10.1145/1450058.1450093.

[16] Alberto de la Encina y Ricardo Peña-Marí. «From natural semantics to C: A formal derivation of two STG machines». En: *J. Funct. Program.* 19.1 (2009), págs. 47-94. DOI: 10.1017/S0956796808006746. URL: https://doi.org/10.1017/S0956796808006746.

[17] Alberto de la Encina y Ricardo Peña-Marí. «From natural semantics to C: A formal derivation of two STG machines». En: *J. Funct. Program.* 19.1 (2009), págs. 47-94. DOI: 10.1017/S0956796808006746. URL: https://doi.org/10.1017/S0956796808006746.

[18] Gadea, Alejandro and Gunther, Emmanuel and Pagano, Miguel. «Mechanization of Coherence and Adequacy for Subtyping». En: *Science of Computer Programming* (2018). URL: https://cs.famaf.unc.edu.ar/~mpagano/coherence-subtyping/.

[19] Alejandro Gadea, Emmanuel Gunther y Miguel Pagano. «The Importance of Being Extrinsic: Coherence and Adequacy for a Call-by-value Language». En: *Proceedings of the 21st Brazilian Symposium on Programming Languages*. SBLP 2017. Presentado en conferencia por Alejandro Gadea. https://cs.famaf.unc.edu.ar/~mpagano/being-extrinsic/. Fortaleza, CE, Brazil: ACM, 2017, 6:1-6:8. ISBN: 978-1-4503-5389-2. DOI: 10.1145/3125374.3125378. URL: http://doi.acm.org/10.1145/3125374.3125378.

[20] George C. Necula and Peter Lee. «The design and implementation of a certifying compiler (with retrospective)». En: *Best of PLDI*. 1998.

[21] Guilhem Jaber y Nicolas Tabareau. «Krivine realizability for compiler correctness». En: *Workshop LOLA 2010, Syntax and Semantics of Low Level Languages*. Edinburgh, United Kingdom, jul. de 2010. URL: https://hal.archives-ouvertes.fr/hal-00475210.

[22] Guilhem Jaber y Nicolas Tabareau. «The Journey of Biorthogonal Logical Relations to the Realm of Assembly Code». En: *Workshop LOLA 2011, Syntax and Semantics of Low Level Languages*. Toronto, Canada, jun. de 2011. URL: https://hal.archives-ouvertes.fr/hal-00594386.

[23] Simon L. Peyton Jones. «Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine - Version 2.5». En: *Journal of Functional Programming* 2 (1992), págs. 127-202.

[24] Krivine, Jean-Louis. «A Call-by-name Lambda-calculus Machine». En: *Higher Order Symbol. Comput.* 20.3 (sep. de 2007), 199-207. ISSN: 1388-3690. DOI: 10.1007/s10990-007-9018-9. URL: http://dx.doi.org/10.1007/s10990-007-9018-9.

[25] John Launchbury. «A Natural Semantics for Lazy Evaluation». En: (1993). Ed. por Mary S. Van Deusen y Bernard Lang, 144-154. DOI: 10.1145/158511.158618. URL: http://doi.acm.org/10.1145/158511.158618.

[26] Leonardo Rodríguez and Miguel Pagano and Daniel Fridlender. «Proving Correctness of a Compiler Using Step-indexed Logical Relations». En: *Electr. Notes Theor. Comput. Sci.* 323 (2016), 197-214. DOI: 10.1016/j.entcs.2016.06.013. URL: http://dx.doi.org/10.1016/j.entcs.2016.06.013.

[27] Xavier Leroy. «Formal Verification of a Realistic Compiler». En: *Commun. ACM* 52.7 (jul. de 2009), págs. 107-115. ISSN: 0001-0782. DOI: 10.1145/1538788.1538814. URL: http://doi.acm.org/10.1145/1538788.1538814.

[28] T. A. Linden. «A Summary of Progress Toward Proving Program Correctness». En: *Proceedings of the December 5-7, 1972, Fall Joint Computer Conference, Part I*. AFIPS '72 (Fall, part I). Anaheim, California: ACM, 1972, págs. 201-211. DOI: 10.1145/1479992.1480019. URL: http://doi.acm.org/10.1145/1479992.1480019.

[29] John Mccarthy y James Painter. «Correctness of a compiler for arithmetic expressions». En: American Mathematical Society, 1967, págs. 33-41.

[30] F. Lockwood Morris. «Advice on Structuring Compilers and Proving Them Correct». En: *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '73. Boston, Massachusetts: ACM, 1973, págs. 144-152. DOI: 10.1145/512927.512941. URL: http://doi.acm.org/10.1145/512927.512941.

[31] Francis Lockwood Morris. «Correctness of Translations of Programming Languages–an Algebraic Approach». AAI7304560. Tesis doct. Stanford, CA, USA, 1972.

[32] Jon Mountjoy. «The Spineless Tagless G-machine, naturally». En: *Proceedings of the third ACM SIGPLAN International Conference on Functional Programming (ICFP '98), Baltimore, Maryland, USA, September 27-29, 1998*. Ed. por Matthias Felleisen, Paul Hudak y Christian Queinnec. ACM, 1998, págs. 163-173. DOI: 10.1145/289423.289439. URL: http://doi.acm.org/10.1145/289423.289439.

[33] Maciej Piróg y Dariusz Biernacki. «A systematic derivation of the STG machine verified in Coq». En: *Proceedings of the 3rd ACM SIGPLAN Symposium on Haskell, Haskell 2010, Baltimore, MD, USA, 30 September 2010*. Ed. por Jeremy Gibbons. ACM, 2010, págs. 25-36. DOI: 10.1145/1863523.1863528. URL: http://doi.acm.org/10.1145/1863523.1863528.

[34] Maciej Piróg y Dariusz Biernacki. «A systematic derivation of the STG machine verified in Coq». En: *Proceedings of the 3rd ACM SIGPLAN Symposium on Haskell, Haskell 2010*. Ed. por Jeremy Gibbons. ACM, 2010, págs. 25-36. DOI: 10.1145/1863523.1863528. URL: http://doi.acm.org/10.1145/1863523.1863528.

[35] Andrew M. Pitts. «Parametric polymorphism and operational equivalence». En: *Mathematical Structures in Computer Science* 10.3 (2000), págs. 321-359. URL: http://journals.cambridge.org/action/displayAbstract?aid=44651.

[36] Andrew Pitts. *Step-Indexed Biorthogonality: a Tutorial Example*. 2008.

[37] Gordon D. Plotkin. «A structural approach to operational semantics». En: *J. Log. Algebr. Program.* 60-61 (2004), págs. 17-139.

[38] John C Reynolds. *The Meaning of Types — From Intrinsic to Extrinsic Semantics*. Inf. téc. Department of Computer Science, University of Aarhus, 2000.

[39] John C. Reynolds. *What do types mean? — From intrinsic to extrinsic semantics*. Ed. por Annabelle McIver y Carroll Morgan. New York, NY, 2003. DOI: 10.1007/978-0-387-21798-7_15. URL: https://doi.org/10.1007/978-0-387-21798-7_15.

[40] Robin Milner and R.W. Weyhrauch. «Proving compiler correctness in a mechanised logic». English. En: *Machine Intelligence* 7 (1972), 51-73.

[41] Leonardo Rodríguez. «Compilación Certificada sobre Máquinas Abstractas de Evaluación Normal». Tesis doct. FaMAF, Universidad Nacional de Córdoba, 2017.

[42] Dana S. Scott. *Outline of a Mathematical Theory of Computation*. Inf. téc. PRG–2. Oxford, England, 1970.

[43] Dana S. Scott. «Data Types as Lattices». En: *SIAM J. Comput.* 5.3 (1976), págs. 522-587. DOI: 10.1137/0205037. URL: http://dx.doi.org/10.1137/0205037.

[44] Dana Scott y Christopher Strachey. *Toward a Mathematical Semantics for Computer Languages*. Inf. téc. PRG-6. 1971.

[45] Peter Selinger. *From Continuation Passing Style to Krivine's Abstract Machine*. Manuscript. Available in Peter Selinger's web site. 2003.

[46] Sestoft, Peter. «Deriving a lazy abstract machine». En: *Journal of Functional Programming* 7.3 (1997), 231-264.

[47] Michael B. Smyth y Gordon D. Plotkin. «The Category-Theoretic Solution of Recursive Domain Equations». En: *SIAM J. Comput.* 11.4 (1982), págs. 761-783. DOI: 10.1137/0211062. URL: http://dx.doi.org/10.1137/0211062.

[48] Yong Kiam Tan, Magnus O. Myreen, Ramana Kumar, Anthony Fox, Scott Owens y Michael Norrish. «A New Verified Compiler Backend for CakeML». En: *SIGPLAN Not.* 51.9 (sep. de 2016), págs. 60-73. ISSN: 0362-1340. DOI: 10.1145/3022670.2951924. URL: http://doi.acm.org/10.1145/3022670.2951924.

[49] *The Glasgow Haskell Compiler*. https://www.haskell.org/ghc/.

[50] Wieczorek, Pawełand Biernacki, Dariusz. «A Coq Formalization of Normalization by Evaluation for Martin-LöF Type Theory». En: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2018. Los Angeles, CA, USA: ACM, 2018, 266-279. ISBN: 978-1-4503-5586-5. DOI: 10.1145/3167091. URL: http://doi.acm.org/10.1145/3167091.

[51] Xuejun Yang, Yang Chen, Eric Eide y John Regehr. «Finding and Understanding Bugs in C Compilers». En: *SIGPLAN Not.* 46.6 (jun. de 2011), págs. 283-294. ISSN: 0362-1340. DOI: 10.1145/1993316.1993532. URL: http://doi.acm.org/10.1145/1993316.1993532.