Sobre la equivalencia entre semánticas operacionales y denotacionales para lenguajes funcionales paralelos

Proyecto de Fin de Máster

Autor: **Lidia Sánchez Gil**
Profesor director: **Yolanda Ortega Mallén**
Facultad de Informática
Universidad Complutense de Madrid
Septiembre 2008

Agradecimientos

En primer lugar quiero expresar mi más sincero agradecimiento a mi directora de proyecto, Yolanda Ortega Mallén, por todo el apoyo que me ha brindado desde que la conocí y en especial por toda la ayuda que me ha prestado en los últimos meses y sobre todo en los últimos días.

También quiero agradecerle a Carlos todo el apoyo que me ha dado desde el principio y por aguantarme cada día, sobre todo en los momentos de más estrés, por acompañarme en los momentos más duros y por ocuparse de todas las tareas domésticas durante las últimas semanas. No puedo olvidar a mis padres que siempre están ahí para animarme y escucharme, ni a mis hermanas, que están pendientes de mi y me apoyan en cada momento. También quiero agradecerle a mi prima Toñi todo el cariño que me ha dado en estos últimos meses. Y cómo no a mis amigas más cercanas: Peñita, Gemita, Isi, Marta y en especial a Mónica que incluso desde la distancia siempre ha estado disponible para escucharme y ofrecerme su apoyo en cualquier momento.

Tampoco puedo olvidar a mis compañeros del despacho de al lado, Ignacio y Nando, por resolver gran parte de mis dudas con LaTeX. Y por último agradecerle a Alicia los cafés que servían para tomar un descanso.

Índice general

Capítulo 1

Introducción

1.1. Semánticas formales

Todos los lenguajes están formados por una sintaxis, que describe cómo han de formarse las expresiones que componen dicho lenguaje, y una semántica que da significado a dichos términos. En el caso de los lenguajes naturales es posible encontrar oraciones con más de un significado, sin embargo, para evitar estos conflictos, los lenguajes de programación son dotados de semánticas formales que no permiten la pluralidad de significados.

Las semánticas formales definen de manera rigurosa un lenguaje de programación, ofreciendo una definición matemáticamente precisa y sin ambigüedades. Las semánticas formales no definen solamente el significado de las expresiones del lenguaje sino que también son útiles para demostrar propiedades sobre el comportamiento de los programas e implementar compiladores e intérpretes del lenguaje, entre otros usos.

Existen distintos tipos de semánticas formales dependiendo del nivel de abstracción y el objetivo con el que se definan. Destacaremos tres de ellos por ser los más utilizados:

Semántica operacional: Describe el significado del lenguaje especificando cómo se ejecuta un programa en una máquina abstracta. Básicamente esta semántica se centra en conocer el resultado que genera el programa y el modo en que este resultado es obtenido. Puede estar basado en una máquina abstracta concreta o en una genérica. Son útiles, sobre todo, en la implementación de los lenguajes.

Semántica denotacional: Define la función que computa el programa, pero sin ocuparse de la forma en que lo hace. Tiene un nivel de abstracción mayor que el de la semántica operacional y esto permite estudiar propiedades formales de los programas como, por ejemplo, las equivalencias que pueda haber entre ellos. Aunque existen varias formas de definir una semántica denotacional la más común se centra en los siguientes aspectos:

- En primer lugar definir el espacio de significados, es decir, cada objeto sintáctico del lenguaje denota un elemento de este espacio. Estos espacios pueden estar formados por funciones [30], trazas [16], fallos [16],etc..
- Dotar a cada constante del lenguaje de un significado en el espacio definido con anterioridad y construir funciones semánticas sobre el espacio de significados para cada operador del lenguaje.
- Definir la función semántica principal que indica el valor semántico de cada programa.

Semántica axiomática: Este tipo de semántica suele definirse para lenguajes en los que la ejecución del programa se basa en cambios de las variables de estado, por ello este enfoque no es adecuado para los lenguajes funcionales. Se define por medio de un conjunto de axiomas que definen las propiedades de un sistema e indican cuándo son verificadas por la ejecución de un programa. El ejemplo más clásico de este tipo de semánticas es el sistema de pre-condiciones y post-condiciones definido para Pascal por Hoare en [15].

1.2. Lenguajes funcionales paralelos

Con la proliferación de máquinas paralelas y distribuidas surge la necesidad de diseñar lenguajes que faciliten la compleja tarea de la programación paralela. Aunque los lenguajes imperativos son eficientes, ofrecen poca ayuda en cuanto a facilidad de programación, ya que conceptos clave como sincronización y comunicación son tratados, generalmente, a un nivel demasiado bajo. En cambio, el paradigma funcional ha sido siempre considerado una buena opción en base a sus bien conocidas ventajas, como abstracción, expresividad, transparencia referencial y un modelo semántico claro.

Además, la programación declarativa en general, y la funcional en particular, ofrecen magníficas oportunidades para la evaluación paralela. De hecho, la investigación en programación funcional paralela durante las últimas décadas ha sido fructífera, habiéndose producido un amplio conjunto de lenguajes funcionales paralelos. Las alternativas para la explotación del paralelismo inherente a los programas funcionales pueden clasificarse en tres tendencias, en base al grado de libertad que se deja al programador para establecer los puntos del programa susceptibles de ser evaluados en paralelo:

Paralelismo implícito: El paralelismo inherente a la semántica de reducción —los *redexes* independientes pueden ser reducidos en un orden arbitrario o incluso en paralelo— es la base para la paralelización *automática* de los programas funcionales.

Paralelismo semi-explícito: El programador indica dónde desearía una evaluación en paralelo. Con este propósito, o bien se añaden al programa anotaciones para el compilador, o bien se utilizan construcciones paralelas de alto nivel —como esqueletos [7] o

estrategias de evaluación [33]— para expresar los algoritmos. Aunque el programador controla el paralelismo hasta cierto punto, los detalles dependen de la implementación.

Paralelismo explícito: Algunos lenguajes de programación funcional bien establecidos como Haskell [27] o ML [25] han sido extendidos con construcciones para la creación explícita de procesos, la comunicación de valores y la sincronización entre procesos, de forma que se puedan describir sistemas concurrentes en general.

Loogen realiza en [22] una clasificación similar y además ofrece una panorámica muy completa, así como una detallada discusión sobre estas alternativas.

1.2.1. Paralelismo en Haskell

El lenguaje funcional Haskell [27] es la base de múltiples versiones paralelas y distribuidas (el trabajo [35] hace un buen repaso de muchas de ellas).

Haskell es un lenguaje con evaluación *perezosa*, es decir, adopta la evaluación en orden normal pero al compartir las reducciones evita la repetición de los cálculos. Ahora bien, una evaluación perezosa restringe la explotación del paralelismo porque las expresiones solo se evalúan bajo demanda. Es por ello que las versiones paralelas de Haskell deben sortear la pereza en algunos puntos:

Trabajo especulativo: Algunos lenguajes permiten la evaluación de partes del código que todavía no han sido solicitadas. Esto no cambia necesariamente la semántica perezosa y secuencial subyacente, ya que el resultado global del programa puede ser obtenido incluso si alguna tarea especulativa se queda atascada; esto puede lograrse si se garantiza que el planificador siempre escoge evaluar los cómputos correspondientes al proceso principal. En este caso, la especulación solo influye en la eficiencia del sistema.

Ejemplos de esta clase de cómputo especulativo son el operador `par` definido en GpH [33] y la creación impaciente de procesos en Eden (ver Sección 1.3).

Introducción de estrictez: Una forma más drástica de eliminar la pereza es forzar la evaluación de ciertas porciones de código antes de que su resultado sea realmente necesario. En este caso la semántica perezosa sí se modifica.

Ejemplos de esta segunda vía son el operador estricto `seq` introducido en GpH [33], o el forzar la reducción a forma normal de los valores a transmitir a través de los canales de Eden (Sección 1.3). De forma similar, la transmisión de listas en Caliban [19, 32] es estricta en la cabeza de la lista.

1.3. El lenguaje Eden

Eden [6, 23] es un lenguaje funcional paralelo que extiende Haskell con construcciones de *coordinación* [8] para controlar la evaluación en paralelo de programas sin perder su naturaleza de "alto nivel". La coordinación en Eden se basa en dos principios: *definición explícita de procesos* y *comunicación implícita basada en streams* [18]. Las características fundamentales de Eden se resumen en:

Abstracciones de proceso: son las expresiones que de un modo puramente funcional definen el comportamiento general de un proceso.

Creaciones de proceso: son aplicaciones de las anteriores a un grupo determinado de expresiones que conformarán los valores de los canales de entrada del nuevo proceso creado.

Comunicaciones entre procesos: son asíncronas e implícitas, pues el paso de mensajes no lo ha de explicitar el programador. Además estas comunicaciones no tienen por qué ser de un único valor, sino que pueden realizarse en forma de *streams*.

Pero las construcciones de Eden se extienden además para modelizar sistemas reactivos:

Creación dinámica de canales: sin esta facilidad las comunicaciones directas se pueden establecer solamente entre padres e hijos. Los canales dinámicos permiten romper esta jerarquía, pudiéndose generar así topologías de comunicación más complejas.

No-determinismo: con el objetivo de modelizar las comunicaciones de varios a uno se introduce la abstracción de proceso predefinida `merge`, que toma varios *streams* y devuelve uno único, compuesto por una mezcla no determinista de los elementos de los anteriores.

Eden ha demostrado su idoneidad para la metodología de programación basada en *esqueletos algorítmicos* [7], con la doble ventaja de que los esqueletos pueden ser implementados y utilizados dentro del mismo lenguaje. De esta forma el programador tiene la oportunidad de utilizar directamente los esqueletos ya existentes, o modificarlos para adecuarse a sus necesidades; o incluso crear nuevos esqueletos y así extender la colección. Actualmente, la biblioteca de Eden proporciona un conjunto amplio de esqueletos que cubren muchos de los patrones de paralelismo más comunes como *map paralelo*, *divide-y-vencerás paralelo*, *búsqueda en paralelo*, y otros, así como las topologías de procesos más típicas, como por ejemplo, tuberías, rejillas, anillos, etc. En [20, 29, 24, 23, 14] pueden encontrarse las definiciones en Eden de estos y otros esqueletos, junto con sus correspondientes modelos de costes, ejemplos de aplicación, y tiempos de ejecución.

1.4. Semánticas para lenguajes funcionales paralelos

La investigación sobre semánticas formales para lenguajes funcionales paralelos ha sido poco intensa. La mayoría de los trabajos han optado por semánticas operacionales (véanse por ejemplo [2] para pH (parallel Haskell)[26] y [4] para GPH (Glasgow parallel Haskell) [34]). Quizás esto está motivado en parte porque, como se afirma en [22], si no existe una noción explícita de proceso paralelo —es decir, en los casos de paralelismo implícito y semi-explícito (ver Sección 1.2)—, la semántica denotacional no se modifica. Esto es cierto cuando se trata de una semántica denotacional estándar, donde exclusivamente se considera la relación funcional entrada-salida. Sin embargo, la calidad de una semántica denotacional reside en su nivel de abstracción, es decir, en el grado de detalle sobre la ejecución que se incorpora en la denotación de un programa; de forma que el máximo nivel de abstracción es fijarse solamente en el valor final. Se puede argüir que otros aspectos a observar en una ejecución paralela, como tiempos de ejecución, consumo de recursos, comunicaciones, etc. dependen de la implementación y, por lo tanto, no deberían tenerse en consideración al definir una semántica formal; como mucho, podrían ser descritos mediante máquinas abstractas.

En el caso particular de Eden, el objetivo fue alcanzar un nivel intermedio de abstracción, de forma que fuera posible comparar programas en términos de la cantidad de trabajo que se debe realizar para obtener el mismo resultado a partir de los mismos datos de entrada. Esto resulta especialmente interesante en el caso de lenguajes funcionales con evaluación dirigida por la demanda —como Haskell—, que es inherentemente secuencial y, por tanto, la paralelización introduce paralelismo especulativo —se evalúan algunas expresiones cuyos resultados jamás se utilizarán—; estas situaciones deberían ser detectadas y evitadas. El grado de duplicación del trabajo y la especulación pueden observarse semánticamente, así como otras propiedades relativas a la comunicación entre procesos. De esta forma, el interés se centra en tres aspectos diferentes:

- Funcionalidad: el valor final calculado.
- Paralelismo: la topología del sistema (los procesos existentes y las conexiones entre ellos) y las correspondientes interacciones generadas durante el cómputo.
- Distribución: el grado de duplicación de trabajo y la especulación.

La semántica operacional definida para Eden en [10, 9] se basa en un modelo distribuido. La ejecución de un programa Eden requerirá, en general, la creación de varios procesos paralelos. Cada uno de estos procesos, a su vez, contendrá un conjunto de hebras de ejecución independiente, cada una de estas dedicada a la producción de una de las salidas del proceso. De esta forma, y como es usual para los lenguajes paralelos y concurrentes [4, 28], es necesario un sistema de transiciones a dos niveles: el nivel inferior se ocupa del comportamiento local de los procesos, mientras que el nivel superior describe los efectos globales sobre el sistema,

más concretamente, expresa las creaciones de proceso y las comunicaciones. En la Sección 2.3 se describirá con más detalle esta semántica operacional.

Ya se ha comentado al comienzo de esta sección que en un contexto paralelo el interés no se centra solamente en la funcionalidad de los programas (es decir, la relación entrada-salida), sino que otros aspectos como la duplicación del trabajo, la distribución de las tareas o las comunicaciones entre procesos pueden merecer nuestra atención. Una semántica denotacional estándar no resulta conveniente para este propósito, ya que el sistema de procesos debe formar parte de la denotación de un programa y entonces la creación de procesos y la comunicación/sincronización se convierten en efectos colaterales producidos por la evaluación de expresiones. Una solución es definir el valor denotacional con un parámetro adicional: la *continuación* [31], es decir, un transformador de estados que recoge el efecto del contexto donde una expresión se ha evaluado. Debido a que la evaluación de una expresión puede depender a veces del valor devuelto por una evaluación previa, en [17] se utilizan *continuaciones de expresión* —funciones que reciben un valor y producen una continuación— para modelizar el orden de evaluación en una semántica denotacional para un λ-cálculo perezoso. Estos conceptos son los que se han utilizado para definir la semántica basada en continuaciones para Eden [12, 9, 13].

1.5. Motivación y objetivos

Como se ha visto en la Sección 1.1, las semánticas operacionales y las semánticas denotacionales son las dos caras de una misma moneda. Mientras las semánticas operacionales describen un modelo abstracto de ejecución de los programas y están dirigidas a diseñadores de compiladores e implementadores, una semántica denotacional construye para cada programa un objeto matemático que representa su significado, de forma que su alto nivel de abstracción permite el análisis de ciertas propiedades de los programas y diversas formas de razonamiento mediante el cálculo, todo ello de gran interés para los programadores. Por estos motivos es frecuente definir para un mismo lenguaje de programación varias semánticas formales con propósitos distintos. La cuestión es establecer una relación entre ellas y demostrar su equivalencia, es decir, probar que describen el mismo lenguaje. Típicamente esto se realiza en términos de *corrección* (de las reglas operacionales con respecto a la semántica denotacional) y de *adecuación computacional* (que significa que existe una reducción operacional válida si y sólo si el valor denotacional está definido), o en términos de *abstracción plena* (full abstraction) con respecto a alguna noción apropiada de comportamiento de programas.

En el caso de Eden ya se ha explicado en la sección anterior que existen dos semánticas formales: una operacional con reglas de transición y una denotational basada en continuaciones. Sin embargo, hasta el momento no se ha establecido ninguna relación entre ambas semánticas.

A continuación se enumeran algunos aspectos que dificultan, en mayor o menor medida,

la tarea de demostrar la equivalencia entre ambas semánticas:

Paralelismo. Recordemos que la semántica operacional para Eden está definida mediante dos niveles de transiciones: un nivel inferior que describe el cómputo funcional ejecutado por cada proceso en el sistema de forma independiente, y un nivel superior que modeliza las interacciones entre dichos procesos. Sin embargo, esta estructura a dos niveles no se refleja directamente en la semántica denotacional.

No determinismo. El paralelismo no necesariamente implica no determinismo (por ejemplo, en [4] se demuestra que la semántica dada para GpH es determinista), pero Eden introduce un operador para mezclar streams de forma no determinista. Esto significa que en la semántica denotacional de Eden hay que tratar con conjuntos de valores y que habrá que definir la forma de reunir las correspondientes reducciones en la parte operacional.

Efectos colaterales. La consideración de las creaciones de procesos y las comunicaciones como efectos colaterales de la evaluación ha obligado a utilizar continuaciones para la definición de la semántica denotacional de Eden. Sin embargo, no parece claro cómo establecer la correspondencia entre estas continuaciones y las transiciones de la semántica operacional.

Mezcla de pereza e impaciencia. Aunque el lenguaje computacional subyacente en Eden es Haskell, que tiene una estrategia de evaluación perezosa, Eden introduce impaciencia tanto en la creación de procesos como en la comunicación de valores, de forma que en Eden siempre existe demanda para la evaluación de la salida de un proceso. Esta mezcla de pereza e impaciencia debe ser tratada con cuidado en los formalismos semánticos. Además, como la pereza implica que los argumentos en una llamada a una función se evalúan como máximo una vez, hay que asegurarse de que los efectos colaterales mencionados anteriormente solo se producen la primera vez que un valor es demandado.

Trabajo especulativo. La creación de procesos y la comunicación de valores en sus versiones impacientes pueden producir cómputos *especulativos*. La cantidad de trabajo especulativo producido durante la evaluación de un programa en Eden es variable, dependiendo del número de procesadores, la velocidad de las operaciones básicas, etc. Esto se refleja en la semántica definiendo dos cotas: una semántica *mínima* donde no hay especulación, y una *máxima* en la que todo cómputo especulativo se lleva a cabo hasta el final.

La conjunción de todos estos factores sustentan el interés de la investigación, pero incrementan su dificultad de tal forma que la consecución del objetivo final de demostrar la equivalencia entre la semántica operacional y la semántica denotacional de Eden se convierte

en un reto ambicioso que hay que abordar poco a poco, por lo que el presente trabajo se centra en la consecución de algunas de las etapas iniciales.

En concreto, el estudio de la equivalencia entre las semánticas va a comenzar concentrándose en la funcionalidad de los programas, es decir, observando exclusivamente el valor final obtenido. Para este cometido se siguen algunas de las técnicas mostradas en [4], que presenta una semántica operacional para GpH basada en la semántica natural para evaluación perezosa definida por Launchbury en [21]. Launchbury demuestra que su semántica operacional es equivalente a la semántica denotacional estándar de Abramsky (para evaluación perezosa) [1]. En [4] se aprovecha este resultado para demostrar la corrección de la semántica operacional de GpH con respecto a una extensión de la semántica denotacional de Abramsky.

En realidad, la semántica operacional para Eden presentada in [11] está inspirada en esta semántica para GpH, por lo que el trabajo descrito en [4] resulta de gran utilidad para nuestros objetivos, si bien la prueba de corrección allí mostrada resulta mucho más simple que en el caso de Eden, porque GpH es un ejemplo de *paralelismo semi-explícito* (ver Sección 1.2), en el que Haskell se extiende con dos combinadores especiales `seq` y `par`; de forma que no existen procesos ni comunicaciones, y por ese motivo una semántica denotacional estándar resulta suficiente.

En el caso de Eden, se comenzará extendiendo su núcleo funcional con la *creación de procesos* que lleva implícita la *comunicación de valores* mediante canales. En el futuro se irá ampliando esta sintaxis para incluir *mezcla no determinista*, *streams* y *canales dinámicos*.

1.6. Plan de trabajo

Para alcanzar el objetivo buscado, este trabajo se ha estructurado en las siguientes partes:

- En el Capítulo 2 se exponen las tres semánticas operacionales que son la base del trabajo recogido en esta memoria. Se comienza exponiendo la semántica natural propuesta por Launchbury en [21] para el núcleo funcional. A continuación se ofrece una breve introducción a la semántica definida en [5] para GpH mostrando las diferencias del modelo utilizado por esta con respecto a la anterior. Por último, se describe la semántica operacional propuesta para Eden en [9]. En realidad, como la sintaxis de Eden es bastante extensa y compleja, para la definición de sus semánticas formales Hidalgo utilizó en [9] Jauja, un cálculo sencillo que recoge las características principales de Eden. En el Capítulo 2 se muestra también la sintaxis de Jauja, aunque restringida al núcleo funcional y la creación de procesos (que denominaremos r-Jauja).

- Una vez establecidas las bases, el Capítulo 3 se dedica a la preparación del terreno para obtener los resultados que conforman este trabajo.

Por un lado, se adapta la semántica operacional de r-Jauja mostrada en el Capítulo 2 al caso en el que se dispone de un único procesador. Esta transformación es necesaria para eliminar los dos niveles de transiciones de la semántica operacional.

Por otro lado, se extiende la semántica natural para incorporar la creación de procesos. Esta extensión supone además algunos cambios en la representación de las reglas, pero es consistente con la semántica de partida, es decir, que si la extensión se restringe al núcleo funcional entonces se obtienen los mismos resultados que con la semántica natural mostrada en el Capítulo 2.

- Los puntos clave del trabajo se encuentran en el Capítulo 4 donde se demostrarán la corrección y adecuación entre la semántica operacional de r-Jauja y una extensión de la semántica denotacional estándar de Abramsky. Esto se realizará demostrando primero la equivalencia entre la semántica operacional para un procesador y la semántica natural extendida (dadas en el Capítulo 3). Posteriormente, se modificarán y extenderán las demostraciones de corrección y adecuación dadas por Launchbury en [21].

  Además, se demuestra la *determinación* de la semántica operacional para r-Jauja, es decir que el valor obtenido para una expresión es independiente del número de procesadores de que se disponga.

- La memoria finaliza con el Capítulo 5 donde se presentarán las conclusiones del trabajo, y se indicarán las líneas a investigar y desarrollar en el futuro.

Capítulo 2

Fundamentos

En este capítulo se describen la semántica operacional natural expuesta por Launchbury en [21] y la semántica operacional de Jauja definida por Hidalgo-Herrero en [9]. Ambas semánticas serán la base de las que se van a utilizar principalmente en este trabajo.

2.1. Semántica operacional natural

El objetivo de esta sección es dar unas breves nociones de la semántica descrita en [21] para la evaluación perezosa de un λ-cálculo extendido (el núcleo funcional junto con la declaración local definida mediante la expresión `let`). Se trata en este caso de una semántica de *paso largo*, es decir, las reglas que la definen parten de una expresión y llegan a un valor; no se centra en las etapas intermedias sino que se fija en el comienzo y el final de la evaluación.

Este λ-cálculo se normaliza a una sintaxis restringida donde las λ-expresiones normalizadas verifican que todas las variables ligadas son distintas (evitando así posibles conflictos) y todas las aplicaciones son de una expresión a una variable. En la figura 2.1 se describe la sintaxis restringida del λ-cálculo estudiado.

El proceso de normalización se realiza en dos etapas:

$$
\begin{array}{lcll}
x & \in & \mathit{Var} & \\
E & \in & \mathit{Exp} & ::= \quad \lambda x.E \\
 & & & \quad\;\; | \quad E\ x \\
 & & & \quad\;\; | \quad x \\
 & & & \quad\;\; | \quad \texttt{let}\ \{x_i = E_i\}_{i=1}^{n}\ \texttt{in}\ E
\end{array}
$$

Figura 2.1: Sintaxis restringida del λ-cálculo extendido

$$
\begin{array}{lcl}
(\lambda x.E)^* & = & \lambda x.(E^*) \\
x^* & = & x \\
(\texttt{let}\ \{x_i = E_i\}_{i=1}^n\ \texttt{in}\ E)^* & = & \texttt{let}\ \{x_i = (E_i^*)\}_{i=1}^n\ \texttt{in}\ (E^*) \\
(E_1\ E_2)^* & = & \begin{cases} (E_1^*)\ E_2 & \text{si } E_2 \text{ es una variable} \\ \texttt{let}\ y = (E_2^*)\ \texttt{in}\ (E_1^*) & \text{e.o.c., donde } y \text{ es una variable fresca} \end{cases}
\end{array}
$$

Figura 2.2: Normalización del λ-cálculo extendido

- En la primera se realiza una α-conversión, es decir, un renombramiento de todas la variables ligadas en E utilizando variables frescas, esto se denota mediante $\widehat{E}$.
- La segunda denotada por E^* asegura que los argumentos de la función son siempre variables (ver la Figura 2.2).

2.1.1. Reglas de reducción

En la figura 2.3 se muestran las reglas de reducción del λ-cálculo extendido descrito anteriormente. En ellas se utiliza la siguiente notación:

$$
\begin{array}{rcccl}
\Gamma, \Delta, \Theta & \in & \mathit{Heap} & = & \mathit{Var} \mapsto \mathit{Exp} \\
W & \in & \mathit{Val} & ::= & \lambda x.E
\end{array}
$$

Definición 2.1.1. Un *heap* es una función parcial de variables a expresiones. Se puede ver como un conjunto de pares variable/expresión (que son las llamadas *ligaduras*) que asocia a distintas variables distintas expresiones. Cada ligadura se denota por $x \mapsto E$. Los *valores* (que pertenecen al conjunto *Val*) son expresiones en *weak head normal form* (*whnf*), es decir, expresiones cuya estructura más externa es una λ-abstracción.

□

Los juicios de la forma $\Gamma : E \Downarrow \Delta : W$ quieren decir que la expresión E en el contexto de las ligaduras del *heap* Γ se reduce al valor W en el que el conjunto de ligaduras se ha modificado a Δ. A lo largo de la evaluación nuevas ligaduras pueden añadirse al *heap* y algunas ligaduras que asociaban variables a expresiones sin evaluar pueden actualizarse asociando las variables a los valores de las expresiones una vez evaluadas.

La regla *Lambda* refleja el hecho de que los términos cuya componente más externa es una abstracción se reescriben a ellos mismos y el *heap* no se modifica. Estas expresiones están en *whnf*, es decir, ya son valores y por ello no es necesaria su evaluación.

$$\text{(Lambda)} \qquad \Gamma : \lambda x.E \;\Downarrow\; \Gamma : \lambda x.E$$

$$\text{(Aplicación)} \qquad \frac{\Gamma : E \;\Downarrow\; \Delta : \lambda y.E' \qquad \Delta : E'[x/y] \;\Downarrow\; \Theta : W}{\Gamma : E\ x \;\Downarrow\; \Theta : W}$$

$$\text{(Variable)} \qquad \frac{\Gamma : E \;\Downarrow\; \Delta : W}{(\Gamma, x \mapsto E) : x \;\Downarrow\; (\Delta, x \mapsto W) : \widehat{W}}$$

$$\text{(Let)} \qquad \frac{(\Gamma, x_1 \mapsto E_1, ..., x_n \mapsto E_n) : E \;\Downarrow\; \Delta : W}{\Gamma : \texttt{let}\ \{x_i = E_i\}_{i=1}^{n}\ \texttt{in}\ E \;\Downarrow\; \Delta : W}$$

Figura 2.3: Reglas de reducción

La regla *Aplicación* reduce primero el término E de la expresión, la función. En el caso en el que este término evalue a una λ-abstracción entonces se procede a realizar la aplicación mediante una β-reducción y se evalúa la expresión resultante.

En la regla *Variable* se refleja el paso de un valor de una variable a otra. Para evaluar la variable x es necesario que en el *heap* haya una ligadura de la forma $x \mapsto E$. Si la expresión E junto con el resto del *heap* evalúa a un valor W junto al *heap* Δ, entonces Δ es aumentado con la ligadura $x \mapsto W$ y como resultado se devuelve $\widehat{W}$ un renombramiento del valor W.

Por último la regla *Let* aumenta el *heap* con nuevas ligaduras y puesto que se ha realizado la normalización no puede haber conflictos entre las variables, por ello no se hace ningún renombramiento. Una vez ampliado el *heap* se procede a evaluar el cuerpo de la expresión `let`.

2.2. Semántica operacional para GpH

Siguiendo la idea expuesta por Launchbury en [21] y que se ha descrito en la sección anterior, los autores de [4] definen una semántica operacional para una extensión paralela del λ-cálculo estándar, que representa al lenguaje Glasgow Parallel Haskell (GpH).

El lenguaje que se describe es un λ-cálculo sin tipos extendido con números, la declaración local `let`, una composición secuencial `seq` y una forma de composición paralela `par`. La sintaxis normalizada de este lenguaje viene descrita en la Figura 2.4.

El comportamiento operacional de `seq` es reducir la expresión E_1 a *whnf* antes de devolver la expresión E_2, de esta forma se fuerza un orden de evaluación. Por otro lado el comportamiento operacional de `par` es evaluar la expresión E_2 y, en caso de disponer de más procesadores que puedan utilizarse, se evaluará la expresión E_1 simultáneamente.

$$
\begin{array}{lcl}
x & \in & \mathit{Var} \\
n & \in & \mathit{Num} \\
E & \in & \mathit{PExp} \\
E & ::= & n \mid x \mid E\ x \mid \lambda x.E \mid \texttt{let}\ \{x_i = E_i\}_{i=1}^{n}\ \texttt{in}\ x \mid E_1\ \texttt{seq}\ E_2 \mid x\ \texttt{par}\ E
\end{array}
$$

Figura 2.4: Sintaxis de un lenguaje funcional paralelo.

2.2.1. Notación

Lo realmente relevante para el estudio que se está realizando es la notación que se utiliza en [4]. En este trabajo, y al igual que en el desarrollado por J. Launchbury, se utiliza la idea de *heaps* como conjuntos de ligaduras. Sin embargo, las ligaduras ahora están etiquetadas, es decir, hay diferencias entre ellas dependiendo del estado de actividad en el que se encuentren y pueden verse como hebras (*threads*) de evaluación en paralelo. Otra diferencia importante con respecto a la semántica expuesta en [21] es que se trata de una semántica de *paso corto*, es decir, las reglas que la definen describen la evolución de cada expresión paso a paso; no se fija sólo en la expresión inicial y en el valor que esta obtiene, sino en cada una de las etapas por las que la expresión pasa antes de obtener el valor final. Este tipo de semántica describe una secuencia de reducción desde una configuración inicial hasta una configuración final:

$$(H,\ \mathit{main} \overset{A}{\mapsto} E) \implies \ldots \implies (H',\ \mathit{main} \overset{I}{\mapsto} W)$$

siendo *main* la variable ligada a la expresión principal y $W \in \mathit{Val}$.

Cada una de las ligaduras etiquetadas que componen los *heaps* se denota por $x \overset{\alpha}{\mapsto} E$, donde α representa el estado en el que se encuentra la ligadura. Una ligadura está en estado activo, A, si en ese momento esta siendo evaluada; se encontrará bloqueada, B, si está a la espera de la evaluación de otra ligadura cuyo valor es necesario para su evaluación; estará en estado ejecutable, E, si no hay recursos suficientes para evaluarla; y en cualquier otro caso la ligadura se etiquetará como inactiva, I.

No se van a exponer ni explicar las reglas de esta semántica ya que no aportan nada nuevo a este trabajo, simplemente conviene destacar que hay dos tipos de reglas: las transiciones de paso simple, que describen cómo se reduce una ligadura activa en el contexto local de un *heap*; y las transiciones de múltiples pasos, que se aplican de forma global a todas las ligaduras del *heap*.

2.3. Semántica operacional de Jauja

La sintaxis de Eden es bastante compleja y por esta razón se trabajará con Jauja [9], un cálculo que recoge las características principales de dicho lenguaje. El estudio que se

$$
\begin{array}{rcl}
x,\ y,\ z & \in & \mathit{Var} \\
E & \in & \mathit{EExp} \\
E & ::= & x \mid \lambda x.E \mid x\ y \mid x\#y \mid \texttt{let}\ \{x_i = E_i\}_{i=1}^{n}\ \texttt{in}\ x
\end{array}
$$

Figura 2.5: Sintaxis de Jauja restringido

va a realizar se centra solamente en un subconjunto de la sintaxis de Jauja y en trabajos futuros se irán incorporando el resto de expresiones. El subconjunto de la sintaxis con el que se va a trabajar está expuesto en la Figura 2.5. Las expresiones *variable, λ-abstracción, aplicación y declaración local* son las propias de un λ-cálculo estándar ampliado con la expresión *let* a las cuales se les ha añadido la *creación de proceso*, $x\#y$. En la Figura 2.5 *Var* representa el conjunto de variables, *Val* el conjunto de valores (expresiones en *whnf*) y *EExp* representa el conjunto de *expresiones extendidas*: son las expresiones del λ-cálculo mostrado en la Sección 2.1 junto con las creaciones de procesos. A este pequeño subconjunto se le llamará *Jauja restringido* y de forma abreviada se hará referencia a él como *Jauja básico*.

La expresión *creación de procesos* es equivalente a la *aplicación funcional* en cuanto a lo que se refiere al valor final obtenido. Sin embargo, los cambios que se producen en el sistema de ejecución son distintos: por un lado la evaluación del argumento de una aplicación es perezosa en contraposición con la creación de procesos, donde es impaciente; por otro lado las variables libres de la abstracción solamente se evalúan bajo demanda, mientras que en la creación de procesos, dependiendo del enfoque que se elija, pueden evaluarse en caso de necesidad o puede darse una evaluación previa para generar el entorno inicial del nuevo proceso.

Se tomará como la semántica operacional de Jauja la expuesta en la Sección 6.7 del Capítulo 6 de [9], que está basada principalmente en la semántica comentada en la sección anterior y, como en esta última, hay dos tipos de reglas: las reglas locales y las reglas globales. Las reglas locales reflejan el comportamiento de cada proceso del sistema, es decir, cómo se comporta cada uno de estos procesos de forma independiente sin interactuar con los demás. Por el contrario las reglas globales reflejan cómo interactuan los procesos entre ellos; es como si se observara la evolución del sistema desde un nivel superior y se viera cómo entran en contacto unos procesos con otros. Otra semejanza con la semántica expuesta en [4] es que se trata de una semántica de paso corto.

La semántica de Jauja y la descrita en la Sección 2.2 son conceptualmente distintas, aunque son similares en cuanto que ambas tienen dos tipos de reglas, unas enfocadas hacia la evolución de las ligaduras activas y otras más generales en las que interactuan distintas ligaduras. En la semántica de la sección anterior no existen procesos diferenciados y todas las ligaduras pertenecen al mismo *heap*, por ello las transiciones de múltiples pasos relacionan varias ligaduras del *heap* único. Sin embargo, en Jauja la creación de procesos permite tener varios *heaps*, asociados cada uno a un proceso distinto; en este caso las reglas globales reflejan

el modo en el que interactuan los diferentes procesos entre sí.

2.3.1. Notación

En este apartado se hará una introducción a la notación que se va a utilizar a lo largo del trabajo. Siguiendo la idea expuesta en [4] se utilizarán *heaps* formados por ligaduras etiquetadas.

Las comunicaciones entre procesos se realizan a través de canales, por eso se introduce un conjunto de identificadores de canal, $\mathcal{C}han$.

Definición 2.3.1. Sean $\theta \in Var \cup \mathcal{C}han$ una variable o un canal, $E \in EExp \cup \mathcal{C}han$ una expresión extendida de la sintaxis de Jauja básico o un canal y α una etiqueta ($\alpha \in \{A, I, B\}$)[1]. Una *ligadura* es una terna de la forma $\theta \overset{\alpha}{\mapsto} E$ y se dice que la variable θ está ligada a la expresión E en el estado α.

□

Definición 2.3.2. Un *heap etiquetado* $H \in EHeap$ es un conjunto finito de ligaduras etiquetadas. Un *proceso* es un par de la forma $\langle p, H \rangle$, donde $p \in IdProc$ es un identificador de proceso y H es un *heap* etiquetado. Un *sistema* S es un conjunto no vacío de procesos.

□

En esta semántica operacional se estudia la evolución de los sistemas. A lo largo de la ejecución del programa el sistema se irá modificando hasta que, o bien la variable principal obtenga un valor, o bien hasta que se produzca algún tipo de bloqueo y no se pueda continuar, siempre y cuando se esté considerando la semántica mínima (la diferencia entre las semánticas minima y máxima se explica al final de la Sección 2.3.3).

2.3.2. Reglas locales

En la Figura 2.6 se muestran las reglas locales de la semántica operacional descrita para Jauja básico. Como en el caso de la semántica natural, la evaluación de una expresión finaliza al alcanzar una *whnf*. En las reglas locales se observa cómo evoluciona cada *heap* etiquetado con respecto al subconjunto del lenguaje formado por las variables, la abstracción funcional, la aplicación funcional y la declaración local de variables.

Cada una de las reglas es de la forma

$$\underbrace{\underbrace{H + \{\theta' \overset{\alpha}{\mapsto} E'\}}_{H^*} : \theta \overset{A}{\mapsto} E}_{H_0} \longrightarrow H', \quad \text{con } \theta, \theta' \in Var \cup \mathcal{C}han$$

[1]La etiqueta E (Ejecutable) no aparece porque en esta semántica se supone que hay recursos suficientes para ejecutar simultáneamente todas las ligaduras activas.

(valor)	$H + \{x \overset{I}{\mapsto} W\} \ : \ \theta \overset{A}{\mapsto} x \ \longrightarrow \ H + \{x \overset{I}{\mapsto} W, \theta \overset{A}{\mapsto} W\}$	
(demanda)	$H + \{x \overset{IAB}{\mapsto} E\} \ : \ \theta \overset{A}{\mapsto} x \ \longrightarrow \ H + \{x \overset{AAB}{\mapsto} E, \theta \overset{B}{\mapsto} x\}$	si $E \notin \mathit{Val}$
(agujero negro)	$H \ : \ x \overset{A}{\mapsto} x \ \longrightarrow \ H + \{x \overset{B}{\mapsto} x\}$	
(dem. aplicac.)	$H + \{x \overset{IAB}{\mapsto} E\} \ : \ \theta \overset{A}{\mapsto} x \ y \ \longrightarrow \ H + \{x \overset{AAB}{\mapsto} E, \theta \overset{B}{\mapsto} x \ y\}$	si $E \notin \mathit{Val}$
(β-reducción)	$H + \{x \overset{I}{\mapsto} \lambda z.E\} \ : \ \theta \overset{A}{\mapsto} x \ y \ \longrightarrow \ H + \{x \overset{I}{\mapsto} \lambda z.E, \theta \overset{A}{\mapsto} E[y/z]\}$	
(let)	$H \ : \ \theta \overset{A}{\mapsto} \texttt{let} \ \{x_i = E_i\}_{i=1}^n \ \texttt{in} \ x \ \longrightarrow$ $\longrightarrow \ H + \{y_i \overset{I}{\mapsto} E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n + \{\theta \overset{A}{\mapsto} x[y_j/x_j]_{j=1}^n\}$	$1 \leq i, j \leq n, y_i$ *frescas*

Figura 2.6: Reglas locales

donde un *heap* etiquetado inicial H_0 se transforma en un *heap* etiquetado final H'. El *heap* etiquetado inicial tiene dos partes diferenciadas: H^* y $\theta \overset{A}{\mapsto} E$; esta última representa la ligadura que está siendo evaluada y que va a evolucionar, mientras que H^* representa el resto del *heap* etiquetado. En algunas ocasiones además de la ligadura que está siendo evaluada entra en juego alguna otra ligadura, $\{\theta' \overset{\alpha}{\mapsto} E'\}$, de H^*, que aparece destacada en la regla. Esta ligadura puede que sea modificada o simplemente interfiera de alguna forma en la evaluación de $\theta \overset{A}{\mapsto} E$.

Algunas veces una ligadura puede aparecer con varias etiquetas. De esta forma se evita la repetición de reglas cuya única diferencia es el estado de alguna ligadura. Por ejemplo, en la regla *demanda* la ligadura $x \overset{IAB}{\mapsto} E$ indica que la variable x puede encontrarse inactiva, activa o bloqueada.

Las tres primeras reglas funcionan de forma similar, se tiene un *heap* etiquetado en el que existe una ligadura activa a una variable; dependiendo del resto del *heap* etiquetado esta ligadura evolucionará de una forma u otra. La regla *valor* copia y liga un valor que ya está calculado a la variable demandante θ. En el caso de que la variable demandante necesite un valor que no ha sido aún calculado, entonces se bloquea su ligadura y se comienza a evaluar la expresión demandada; esto viene reflejado en la regla *demanda*. En el caso de haber una autorreferencia por parte de la variable demandante no será posible avanzar y se producirá el bloqueo de la ligadura tratada (ver la regla *agujero negro*).

Las dos siguientes reglas tienen en común una variable ligada en estado activo a una aplicación $x \ y$. Dependiendo de si x está ligada a una λ-abstracción o a una expresión se aplicará una de las siguientes reglas: *β-reducción* en el primer caso o *demanda de aplicación* en el segundo. La primera realiza la aplicación mientras que la segunda bloquea la variable

demandante y comienza a evaluar la expresión ligada a la variable x.

Se termina con la regla *let*. Se trata de la declaración local de variables e introduce nuevas ligaduras en el *heap* etiquetado. Las nuevas ligaduras se etiquetan como inactivas ya que aún no han sido demandadas por el cuerpo de la declaración, que es la única expresión que puede demandarlas. El renombramiento se realiza para evitar el choque de variables.

2.3.3. Reglas globales

Como ya se ha explicado, las reglas globales son aquellas que puede ver el observador del nivel superior. Describen la creación de procesos y la comunicación entre ellos. También se ocupan del desbloqueo y desactivación de las ligaduras. En la figura 2.7 se describe el comportamiento de estas reglas.

En el trabajo [9] se definen dos versiones para la semántica operacional de Jauja. En la primera todas las ligaduras que han de copiarse de un *heap* etiquetado a otro han de estar evaluadas, mientras que en la segunda versión no es necesario que las ligaduras que se han de copiar al otro *heap* etiquetado estén previamente evaluadas sino que se copiarán en el *heap* etiquetado receptor en estado inactivo. En caso de que sea necesario evaluarlas para obtener el valor de alguna variable, será el mismo proceso receptor el que se encargue de su evaluación. La única condición que se impone es que no se realicen copias de los canales, pues éstos sólo permiten la comunicación entre dos procesos concretos. Tampoco se permite copiar las creaciones de procesos pendientes, para evitar que ciertos cálculos se realicen un número indefinido de veces; cada vez que se copiara una creación de ésta se realizaría una vez por parte del proceso en la que se encontraba y otra vez por parte del proceso en el que ha sido copiado.

En este trabajo se ha optado por la segunda versión porque su tratamiento resulta más sencillo para los objetivos perseguidos, ya que no es necesario generar demanda para todas las dependencias que podrían aparecer en una creación de proceso o una comunicación.

La creación de procesos es una expresión impaciente que hace que entre en juego el paralelismo especulativo, es decir, los procesos se crean tan pronto como sea posible y esto hace que puedan evaluarse expresiones que quizá no sean necesarias para evaluar la expresión principal. Este hecho viene reflejado en la regla *creación de proceso*, que determina las partes de la expresión que van a ser evaluadas por cada uno de los procesos involucrados. En una expresión de instanciación de proceso, $x\#y$, el proceso padre, p, se encarga de la evaluación de la variable y mientras que el proceso hijo, q, se encarga de la evaluación de la variable x y de la aplicación $x\ y$. Estos procesos no son completamente independientes en tanto en cuanto pueden necesitar algún valor producido por el otro. En el caso del hijo es posible que necesite el valor al que la variable y da lugar, mientras que el padre está a la espera de que el hijo le comunique el valor resultante de la aplicación. Por esta razón se crean los canales de entrada y salida i y o. Además, para evitar posibles conflictos futuros, todas las copias

creación de proceso

$$(S, \langle p, H + \theta \overset{\alpha}{\mapsto} x\#y\rangle) \overset{cp}{\to} (S, \langle p, H + \{\theta \overset{B}{\mapsto} o, i \overset{A}{\mapsto} y\}\rangle, \langle q, \eta(\mathsf{nh}(x, H)) + \{o \overset{A}{\mapsto} \eta(x)\ z, z \overset{B}{\mapsto} i\}\rangle)$$
si $\neg\mathsf{d}(x, H + \{\theta \overset{\alpha}{\mapsto} x\#y\})$, q, z, i, o frescas, η renombramiento fresco

comunicación de valor

$$(S, \langle p, H_p + \{ch \overset{\alpha}{\mapsto} W\}\rangle, \langle c, H_c + \{\theta \overset{B}{\mapsto} ch\}\rangle) \overset{com}{\to} (S, \langle p, H_p\rangle, \langle c, H_c + \eta(\mathsf{nh}((, W), H_p)) + \{\theta \overset{A}{\mapsto} \eta(W)\}\rangle)$$
si $\neg\mathsf{d}(W, H_p)$, η renombramiento fresco

desbloqueo

$$(S, \langle p, H + \{x \overset{A}{\mapsto} W, \theta \overset{B}{\mapsto} E^x_B\}\rangle) \overset{desb}{\to} (S, \langle p, H + \{x \overset{A}{\mapsto} W, \theta \overset{A}{\mapsto} E^x_B\}\rangle)$$

desactivación

$$(S, \langle p, H + \{\theta \overset{A}{\mapsto} W\}\rangle) \overset{desact}{\to} (S, \langle p, H + \{\theta \overset{I}{\mapsto} W\}\rangle)$$

bloqueo de creación de procesos

$$(S, \langle p, H + \{\theta \overset{IA}{\mapsto} x\#y\}\rangle) \overset{bcp}{\to} (S, \langle p, H + \{\theta \overset{B}{\mapsto} x\#y\}\rangle)$$

Figura 2.7: Reglas globales

que sean necesarias se realizarán aplicando renombramientos. La función de *heap necesario*, nh, definida en la Sección 2.3.4, devuelve las ligaduras que son necesarias para la evaluación de una expresión; en este caso indica las ligaduras que han de copiarse en el *heap* del nuevo proceso q por ser necesarias para la evaluación ligada al canal de salida o. La creación de procesos no se realizará en el caso de depender de una comunicación que aún no ha sido transmitida o de otra creación de proceso, la función de *dependencia*, d, definida también en la Sección 2.3.4, informa de estas circunstancias.

Como ya se ha explicado antes, los procesos se relacionan mediante canales de comunicación por los que se envían valores. La única condición que ha de verificarse para transmitir un valor a través de un canal es que dicho valor no dependa de otra comunicación o alguna creación de proceso, hecho que viene reflejado por la función d en la regla *comunicación de valor*. Como en las demás ocasiones siempre que se copien ligaduras de un *heap* etiquetado a otro se renombrarán para evitar conflictos. Esta regla tiene un significado similar al de la regla local *valor* pues su objetivo es transmitir un valor de una ligadura a otra. La diferencia radica en que al estar el valor en el *heap* etiquetado de otro proceso distinto al de la ligadura que lo demanda han de realizarse las copias y los renombramientos necesarios. Una vez que la comunicación ha sido realizada el canal desaparece porque ya ha cumplido su función.

DEFINICIÓN 2.3.3. Se llama *expresión inmediatamente bloqueada en la variable* x y se denota por E^x_B a aquellas de la forma: x o $x\ y$, donde y es una variable cualquiera. E^x_B indica que la expresión E, para proseguir su evaluación, necesita que la variable x esté evaluada.

□

Una variable $\theta \in \mathit{Var} \cup \mathcal{C}\mathit{han}$ se bloquea en una expresión E_B^x cuando necesita el valor ligado a x para continuar con su evaluación. Una vez que x ya ha obtenido su valor se podrá continuar con la evaluación de θ, por ello en la regla *desbloqueo* se procede a cambiar el estado de la ligadura de θ de bloqueado a activo.

La regla *desactivación* modifica el estado de una ligadura que ya ha alcanzado un valor pasándolo de activo a inactivo, pues una vez que una variable obtiene un valor no va a sufrir ningún otro cambio.

Por último la regla *bloqueo de creación de proceso* bloquea, como su nombre indica, las creaciones de procesos que no hayan podido realizarse por depender de algún valor que ha de ser comunicado a través de un canal. No es necesaria una regla que desbloquee la creación de proceso pues en cuanto no dependa de ninguna comunicación el proceso será creado, ya que en la regla *creación de proceso* no se exige que la ligadura se encuentre en un estado concreto sino que la única condición es que no dependa de canales de comunicación.

Además de estas reglas están las que describen el paralelismo del lenguaje, que no sólo consiste en la creación de procesos paralelos, sino que también se preocupa de la ejecución en paralelo de distintas ligaduras activas (esta parte es similar a la recogida en [5] y [3] con las transiciones de múltiples pasos). En principio, la posibilidad de evolucionar varias ligaduras activas al mismo tiempo viene determinada por el número de procesadores disponibles, pero en este caso se supone que se dispone de un número ilimitado de procesadores. Antes de mostrar estas reglas y dar su descripción se definirá el significado de *conjunto de ligaduras desarrollables*.

DEFINICIÓN 2.3.4. Dado un *heap* H, se define el *conjunto de ligaduras desarrollables de* H, denotado por $\mathcal{LD}(H)$, como aquel que está formado por las ligaduras activas de H que pueden evolucionar aplicando alguna de las reglas locales descritas en la Figura 2.6.

□

La evolución de cada proceso p del sistema consiste en aplicar a cada ligadura desarrollable una regla local:

paralelo-p

$$\frac{\{H_p^{(i,1)} + H_p^{(i,2)} \,:\, \theta_p^i \overset{A}{\mapsto} E_p^i \longrightarrow H_p^{(i,1)} + K_p^{(i,2)} | H_p = H_p^{(i,1)} + H_p^{(i,2)} + \{\theta_p^i \overset{A}{\mapsto} E_p^i\} \wedge \theta_p^i \overset{A}{\mapsto} E_p^i \in \mathcal{LD}(H_p)\}_{i=1}^{n_p}}{\langle p, H_p\rangle \overset{par}{\Longrightarrow}_p \langle p, (\bigcap_{i=1}^{n_p} H_p^{(i,1)}) \bigcup (\bigcup_{i=1}^{n_p} K_p^{(i,2)})\rangle}$$

donde $n_p = |\mathcal{LD}(H_p)|$, es decir el número de ligaduras desarrollables que hay en el *heap* etiquetado del proceso p; $H_p^{(i,1)}$ es la parte de H_p que permanece sin modificar tras la aplicación de la correspondiente regla local; $H_p^{(i,2)}$ es la parte de H_p que sufrirá algún cambio al aplicar la regla local; y, por último, $K_p^{(i,2)}$ corresponde a la parte del *heap* etiquetado que ha sido modificada tras aplicar la regla. Es decir cada *heap* etiquetado H_p se divide en dos partes, una que permanecerá invariable, $H_p^{(i,1)}$, y otra que se modificará pasando de ser $H_p^{(i,2)}$ a ser $K_p^{(i,2)}$.

En el Capítulo 6 del trabajo [9] se demuestra que no hay interferencias entre las evoluciones locales. Es decir que no importa el orden en que se apliquen las reglas locales, pues las partes modificadas de cada *heap* etiquetado no impiden que se apliquen las demás reglas.

A continuación se mostrará la regla de paralelo del sistema. Esta regla indica que la regla *paralelo-p* se va a aplicar a cada uno de los procesos que forman el sistema de forma simultánea. En la parte superior de la regla se indica la aplicación de *paralelo-p* a cada proceso que compone el sistema, en la parte inferior se muestra la evolución del sistema inicial S al nuevo sistema en el que el *heap* de cada proceso, H_p, ha evolucionado a un nuevo *heap* H'_p:

(paralelo)

$$\frac{\{\langle p, H_p\rangle \overset{par}{\Longrightarrow}_p \langle p, H'_p\rangle\}_{\langle p,H_p\rangle \in S}}{S \overset{par}{\Longrightarrow} \{\langle p, H'_p\rangle\}_{\langle p,H_p\rangle \in S}}$$

Las transiciones que se aplican para pasar de un sistema S a otro S' se descomponen de la siguiente forma:

$$\Longrightarrow \; = \overset{par}{\Longrightarrow};\ \overset{com}{\Longrightarrow};\ \overset{cp}{\Longrightarrow};\ \overset{desb}{\Longrightarrow};\ \overset{desact}{\Longrightarrow};\overset{bcp}{\Longrightarrow}$$

donde $\overset{com}{\Longrightarrow}$ indica que la regla de comunicación tiene que ser aplicada repetidamente hasta que no sea posible ninguna comunicación. Lo mismo ocurre con $\overset{cp}{\Longrightarrow}$, $\overset{desb}{\Longrightarrow}$, $\overset{desact}{\Longrightarrow}$ y $\overset{bcp}{\Longrightarrow}$; todas las reglas se aplicarán una y otra vez hasta que no sea posible aplicarlas, es decir, son reglas de paso reiterado.

Definición 2.3.5. Sean S un sistema, y $\diamond$ un nombre de una regla global. Para cada regla de paso simple $S \overset{\diamond}{\longrightarrow} S'$. Se define una *regla de paso reiterado* $S \overset{\diamond}{\Longrightarrow} S'$ como aquella que verifica:

1. $S \overset{\diamond}{\longrightarrow}^* S'$ y,
2. no existe S'' tal que $S \overset{\diamond}{\longrightarrow} S''$.

□

En [9] se demuestra que cuando se parte de un sistema finito cada una de estas transiciones termina y en consecuencia también termina la transición principal.

Definición 2.3.6. Dada una expresión E, se define una *secuencia de reducción* para dicha expresión como una secuencia, finita o infinita, de sistemas tales que:

$$\langle p_0, \{main \overset{A}{\mapsto} E\}\rangle \Longrightarrow^* \langle p_0, H + \{main \overset{I}{\mapsto} W\}\rangle,\ \langle p_1, H_1\rangle,\ \ldots,\ \langle p_n, H_n\rangle \Longrightarrow^*$$

□

La configuración o sistema final depende de la semántica que se esté considerando:

$$\mathsf{d}(x,H) = \begin{cases} \textit{verdadero} & \text{si } \theta \overset{\alpha}{\mapsto} E \in \mathsf{nh}(x,H) \text{ donde } (\theta \in \mathit{Var} \cup \mathit{Chan}) \wedge (E = ch \vee E = y\#z) \\ \textit{falso} & \text{e.o.c.} \end{cases}$$

Figura 2.8: Función de dependencia

$$\begin{gathered} \mathsf{dexp}(\theta) = \emptyset \\ \mathsf{dexp}(\lambda x.E) = \{E\} \\ \mathsf{dexp}(E_1\ E_2) = \{E_1,\ E_2\} \\ \mathsf{dexp}(E_1 \# E_2) = \{E_1,\ E_2\} \\ \mathsf{dexp}(\texttt{let}\ \{x_i = E_i\}_{i=1}^{n}\ \texttt{in}\ E) = \{E_1,\ \ldots,\ E_n,\ E\} \end{gathered}$$

$$\mathsf{nh}(E,H) \models \{\text{(i)},\text{(ii)}\} \wedge \forall A.(A \subseteq H \wedge A \models \{\text{(i)},\text{(ii)}\} \Rightarrow \mathsf{nh}(E,H) \subseteq A)$$

$$\begin{aligned} &\text{(i)} \quad x \overset{\alpha}{\mapsto} E \in H \Rightarrow \{x \overset{I}{\mapsto} E\} \cup \mathsf{nh}(E,H) \subseteq \mathsf{nh}(x,H) \\ &\text{(ii)} \quad \bigcup_{E' \in \mathsf{dexp}(E)} \mathsf{nh}(E',H) \subseteq \mathsf{nh}(E,H) \end{aligned}$$

Figura 2.9: Función de heap necesario

- *Semántica mínima:* el sistema final será el primero en el que la ligadura de la variable *main* se encuentra en estado inactivo.
- *Semántica máxima:* en este caso la configuración final se alcanzará cuando no quede ninguna ligadura activa, puede haber dos causas: que todos los procesos hayan terminado su ejecución, o que el sistema esté bloqueado y no pueda evolucionar.

2.3.4. Funciones auxiliares

A continuación se describirán las funciones auxiliares d y nh que se han utilizado en las reglas de la Figura 2.7.

La función de dependencia d (mostrada en la figura 2.8) indica si la evaluación de la expresión ligada a la variable en estudio depende de alguna comunicación que aún no ha sido realizada o de alguna creación de proceso que aún no haya podido realizarse. El significado de esta función es similar al expuesto en [10], sin embargo aquí está definida en términos de la función nh que se expone a continuación.

En la figura 2.9 se muestra la función de *heap* necesario nh que indica la parte del *heap* etiquetado que se necesita para evaluar la expresión ligada a la variable en estudio. Es decir, recoge todas las ligaduras que pueden ser necesarias para la evaluación de la variable.

Por lo tanto la función d devolverá verdadero si en el *heap* necesario hay alguna ligadura a un canal (comunicación pendiente) o a una creación de proceso. En cualquier otro caso la

función de dependencia devolverá falso.

2.3.5. Ejemplos

La semántica operacional de Jauja que se ha elegido es aquella en la que no se exige que todas las dependencias libres estén evaluadas. Por ello se podría pensar que las reglas *creación de proceso* y *comunicación de valor* no deberían tener ninguna restricción y que la regla global *bloqueo de creación de proceso* no es necesaria. Sin embargo esto no es así. En primer lugar, las reglas *creación de proceso* y *comunicación de valor* no exigen que las dependencias libres estén evaluadas pero sí que no haya dependencia de canales o alguna creación de proceso que pudiera dar lugar a algún canal.

Aún así se podría seguir pensando que *bloqueo de creación de proceso* no es necesaria, ya que en el caso de una creación de proceso que tenga alguna dependencia, en cuanto ésta fuera resuelta se podría aplicar la regla *creación de proceso* y continuar con la ejecución sin necesidad de haber bloqueado la ligadura en cuestión. Esto es cierto siempre y cuando no exista ningún tipo de interbloqueo entre las creaciones de procesos del sistema, pues de haberlos se podría llegar a un sistema que no pudiera evolucionar pero en el que no se pudiera abortar la ejecución por tener alguna ligadura activa. Esto queda ilustrado mediante los siguientes ejemplos.

EJEMPLO 2.3.7. *Sin interbloqueos entre las creaciones de procesos*

En este primer ejemplo se muestra un caso en el que no hay interbloqueos entre las creaciones de procesos. Se verá que en este caso concreto (y en otros similares) se podría prescindir de la regla *bloqueo de creación de proceso*. Para verlo, la evaluación se realizará simultáneamente prescindiendo de la regla *bloqueo de creación de proceso* (parte izquierda) y sin prescindir de ella (parte derecha). En el lado izquierdo la transición que hace evolucionar el sistema S a S' viene definida por:

$$\Longrightarrow = \overset{par}{\Longrightarrow};\ \overset{com}{\Longrightarrow};\ \overset{cp}{\Longrightarrow};\ \overset{desb}{\Longrightarrow};\ \overset{desact}{\Longrightarrow}\ .$$

En el lado derecho la transición que se aplica es la que se ha explicado en la Sección 2.3.3:

$$\Longrightarrow = \overset{par}{\Longrightarrow};\ \overset{com}{\Longrightarrow};\ \overset{cp}{\Longrightarrow};\ \overset{desb}{\Longrightarrow};\ \overset{desact}{\Longrightarrow};\overset{bcp}{\Longrightarrow}\ .$$

Mientras los *heaps* etiquetados evolucionen de igual forma no se harán distinciones y los sistemas sólo aparecerán una vez. Además tanto en este ejemplo, como en todos los posteriores, las ligaduras que han sido modificadas con respecto al *heap* anterior se destacarán en negrita.

La expresión a evaluar es la siguiente:

$$\begin{array}{l}\texttt{let}\ x_0 = x_1 \# x_2,\ x_1 = \lambda y_1.y_1,\\ \qquad x_2 = \lambda y_2.y_2,\ x_3 = x_0 \# x_2\\ \texttt{in}\ \ x_5\end{array}$$

Se comienza ligando la expresión inicial a la variable principal *main* en estado activo, siendo esta la única ligadura en el *heap*.

main
$main \overset{\mathbf{A}}{\mapsto} \texttt{let}\ \mathbf{x_0} = \mathbf{x_1}\#\mathbf{x_2},\ \mathbf{x_1} = \lambda\mathbf{y_1}.\mathbf{y_1}$ $\mathbf{x_2} = \lambda\mathbf{y_2}.\mathbf{y_2},\ \mathbf{x_3} = \mathbf{x_0}\#\mathbf{x_2}$ $\texttt{in}\ \mathbf{x_3}$

Primero se aplicará la regla local *let* para crear el *heap* etiquetado.

LET

main	
$main \overset{\mathbf{A}}{\mapsto} \mathbf{x_3}$	
$\mathbf{x_0} \overset{\mathbf{I}}{\mapsto} \mathbf{x_1}\#\mathbf{x_2}$	$\mathbf{x_1} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_1}.\mathbf{y_1}$
$\mathbf{x_2} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_2}.\mathbf{y_2}$	$\mathbf{x_3} \overset{\mathbf{I}}{\mapsto} \mathbf{x_0}\#\mathbf{x_2}$

Hay dos creaciones de procesos una ligada a x_0 y otra ligada x_3. La ligadura de x_0 puede evolucionar por no tener dependencias ($\neg\mathsf{d}(x_1, H)$), en cambio la ligadura de x_3 sí tiene dependencias ($\mathsf{d}(x_0, H)$), y por lo tanto no es posible aplicarle la regla *creación de proceso*.

CP

main		**p1**
$main \overset{A}{\mapsto} x_3$		
$\mathbf{x_0} \overset{\mathbf{B}}{\mapsto} \mathbf{o_1}$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$\mathbf{x_4} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_3}.\mathbf{y_3}$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_3 \overset{I}{\mapsto} x_0\#x_2$	$\mathbf{x_5} \overset{\mathbf{B}}{\mapsto} \mathbf{i_1}$
	$\mathbf{i_1} \overset{\mathbf{A}}{\mapsto} \mathbf{x_2}$	$\mathbf{o_1} \overset{\mathbf{A}}{\mapsto} \mathbf{x_4}\ \mathbf{x_5}$

Si no estuviera la regla *bloqueo de creación de proceso* terminaría la primera transición $\Longrightarrow$ (parte izquierda), sin embargo si esta regla existe la creación de proceso que no se ha podido realizar se bloquea (parte derecha).

sin bcp

main		**p1**
$main \overset{A}{\mapsto} x_3$		
$x_0 \overset{B}{\mapsto} o_1$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_3 \overset{I}{\mapsto} x_0\#x_2$	$x_5 \overset{B}{\mapsto} i_1$
	$i_1 \overset{A}{\mapsto} x_2$	$o_1 \overset{A}{\mapsto} x_4\ x_5$

BCP

main		**p1**
$main \overset{A}{\mapsto} x_3$		
$x_0 \overset{B}{\mapsto} o_1$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$\mathbf{x_3} \overset{\mathbf{B}}{\mapsto} \mathbf{x_0}\#\mathbf{x_2}$	$x_5 \overset{B}{\mapsto} i_1$
	$i_1 \overset{A}{\mapsto} x_2$	$o_1 \overset{A}{\mapsto} x_4\ x_5$

Ahora se aplica la regla *paralelo* y evolucionan todas las ligaduras que estén activas en ese momento.

PARALELO

main		**p₁**
$main \overset{\mathbf{B}}{\mapsto} \mathbf{x_3}$		
$x_0 \overset{B}{\mapsto} o_1$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$\mathbf{x_3} \overset{\mathbf{A}}{\mapsto} \mathbf{x_0 \# x_2}$	$x_5 \overset{B}{\mapsto} i_1$
	$\mathbf{i_1} \overset{\mathbf{A}}{\mapsto} \lambda \mathbf{y_2.y_2}$	$\mathbf{o_1} \overset{\mathbf{A}}{\mapsto} \mathbf{x_5}$

PARALELO

main		**p₁**
$main \overset{\mathbf{B}}{\mapsto} \mathbf{x_3}$		
$x_0 \overset{B}{\mapsto} o_1$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$\mathbf{x_3} \overset{\mathbf{B}}{\mapsto} \mathbf{x_0 \# x_2}$	$x_5 \overset{B}{\mapsto} i_1$
	$\mathbf{i_1} \overset{\mathbf{A}}{\mapsto} \lambda \mathbf{y_2.y_2}$	$\mathbf{o_1} \overset{\mathbf{A}}{\mapsto} \mathbf{x_5}$

Puesto que el canal i_1 ya está ligado a un valor en *whnf* se procede a realizar la comunicación.

COMUNICACIÓN

main		**p₁**
$main \overset{B}{\mapsto} x_3$		
$x_0 \overset{B}{\mapsto} o_1$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_3 \overset{A}{\mapsto} x_0 \# x_2$	$\mathbf{x_5} \overset{\mathbf{A}}{\mapsto} \lambda \mathbf{y_4.y_4}$
		$o_1 \overset{A}{\mapsto} x_5$

COMUNICACIÓN

main		**p₁**
$main \overset{B}{\mapsto} x_3$		
$x_0 \overset{B}{\mapsto} o_1$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_3 \overset{B}{\mapsto} x_0 \# x_2$	$\mathbf{x_5} \overset{\mathbf{A}}{\mapsto} \lambda \mathbf{y_4.y_4}$
		$o_1 \overset{A}{\mapsto} x_5$

DESACTIVACIÓN

main		**p₁**
$main \overset{B}{\mapsto} x_3$		
$x_0 \overset{B}{\mapsto} o_1$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_3 \overset{A}{\mapsto} x_0 \# x_2$	$\mathbf{x_5} \overset{\mathbf{I}}{\mapsto} \lambda \mathbf{y_4.y_4}$
		$o_1 \overset{A}{\mapsto} x_5$

DESACTIVACIÓN

main		**p₁**
$main \overset{B}{\mapsto} x_3$		
$x_0 \overset{B}{\mapsto} o_1$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_3 \overset{B}{\mapsto} x_0 \# x_2$	$\mathbf{x_5} \overset{\mathbf{I}}{\mapsto} \lambda \mathbf{y_4.y_4}$
		$o_1 \overset{A}{\mapsto} x_5$

En el lado izquierdo, donde no existe la regla *bloqueo de creación de proceso*, la ligadura correspondiente a x_3 ha quedado activa. Al aplicar la regla *paralelo* no evoluciona ya que no hay ninguna regla que se pueda aplicar a ella (no es una ligadura desarrollable).

PARALELO

main		**p₁**
$main \overset{B}{\mapsto} x_3$		
$x_0 \overset{B}{\mapsto} o_1$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_3 \overset{A}{\mapsto} x_0 \# x_2$	$x_5 \overset{I}{\mapsto} \lambda y_4.y_4$
		$\mathbf{o_1} \overset{\mathbf{A}}{\mapsto} \lambda \mathbf{y_4.y_4}$

PARALELO

main		**p₁**
$main \overset{B}{\mapsto} x_3$		
$x_0 \overset{B}{\mapsto} o_1$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_3 \overset{B}{\mapsto} x_0 \# x_2$	$x_5 \overset{I}{\mapsto} \lambda y_4.y_4$
		$\mathbf{o_1} \overset{\mathbf{A}}{\mapsto} \lambda \mathbf{y_4.y_4}$

COMUNICACIÓN

main		**p₁**
$main \overset{B}{\mapsto} x_3$		
$\mathbf{x_0} \overset{\mathbf{A}}{\mapsto} \lambda \mathbf{y_5}.\mathbf{y_5}$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_3 \overset{A}{\mapsto} x_0 \# x_2$	$x_5 \overset{I}{\mapsto} \lambda y_4.y_4$

COMUNICACIÓN

main		**p₁**
$main \overset{B}{\mapsto} x_3$		
$\mathbf{x_0} \overset{\mathbf{A}}{\mapsto} \lambda \mathbf{y_5}.\mathbf{y_5}$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_3 \overset{B}{\mapsto} x_0 \# x_2$	$x_5 \overset{I}{\mapsto} \lambda y_4.y_4$

La ligadura de x_3 ya no tiene dependencias y, como la regla *creación de proceso* se aplica a la ligadura sin importar el estado en el que se encuentre, se crea el proceso ligado a x_3.

CP

main		**p₁**	**p₂**
$main \overset{B}{\mapsto} x_3$			
$x_0 \overset{A}{\mapsto} \lambda y_5.y_5$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$	$\mathbf{x_6} \overset{\mathbf{I}}{\mapsto} \lambda \mathbf{y_6}.\mathbf{y_6}$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$\mathbf{x_3} \overset{\mathbf{B}}{\mapsto} \mathbf{o_2}$	$x_5 \overset{I}{\mapsto} \lambda y_4.y_4$	$\mathbf{x_7} \overset{\mathbf{B}}{\mapsto} \mathbf{i_2}$
	$\mathbf{i_2} \overset{\mathbf{A}}{\mapsto} \mathbf{x_2}$		$\mathbf{o_2} \overset{\mathbf{A}}{\mapsto} \mathbf{x_6}\ \mathbf{x_7}$

Tras aplicar las reglas correspondientes varias veces, en ambos casos se obtiene el mismo resultado:

SISTEMA FINAL

main		**p₁**	**p₂**
$main \overset{I}{\mapsto} \lambda y_8.y_8$			
$x_0 \overset{I}{\mapsto} \lambda y_5.y_5$	$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$	$x_6 \overset{I}{\mapsto} \lambda y_6.y_6$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_3 \overset{I}{\mapsto} \lambda y_8.y_8$	$x_5 \overset{I}{\mapsto} \lambda y_4.y_4$	$x_7 \overset{I}{\mapsto} \lambda y_7.y_7$

□

EJEMPLO 2.3.8. *Con interbloqueos entre las creaciones de procesos*

En este ejemplo se muestra la importancia de la regla *bloqueo de creación de proceso* y se ve que no es indiferente aplicarla o no (esto ocurre cuando hay interbloqueos entre las creaciones de procesos).

La expresión a evaluar es la siguiente:

$$\texttt{let}\ x_0 = x_1 \# x_2,\ x_1 = \lambda t_1.t_1, x_2 = x_0 \# x_1$$
$$\texttt{in}\ \ x_2$$

Al igual que el ejemplo anterior se resolverá a la vez tanto si se considera la regla *bloqueo de creación de proceso* como si no. Se comienza, como siempre, creando el *heap* inicial que consta de una única ligadura activa en la variable *main*.

SISTEMA INICIAL

main
$main \overset{\mathbf{A}}{\mapsto} \texttt{let } \mathbf{x_0} = \mathbf{x_1}\#\mathbf{x_2},\ \mathbf{x_1} = \lambda\mathbf{t_1}.\mathbf{t_1}, \mathbf{x_2} = \mathbf{x_0}\#\mathbf{x_1} \texttt{ in } \mathbf{x_2}$

LET

main	
$main \overset{\mathbf{A}}{\mapsto} \mathbf{x_2}$	$\mathbf{x_0} \overset{\mathbf{I}}{\mapsto} \mathbf{x_1}\#\mathbf{x_2}$
$\mathbf{x_1} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_1}.\mathbf{y_1}$	$\mathbf{x_2} \overset{\mathbf{I}}{\mapsto} \mathbf{x_0}\#\mathbf{x_1}$

CP

main		**p1**
$main \overset{\mathbf{A}}{\mapsto} \mathbf{x_2}$	$\mathbf{x_0} \overset{\mathbf{B}}{\mapsto} \mathbf{o_1}$	$\mathbf{x_3} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_2}.\mathbf{y_2}$
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_2 \overset{I}{\mapsto} x_0\#x_1$	$\mathbf{x_4} \overset{\mathbf{B}}{\mapsto} \mathbf{i_1}$
	$\mathbf{i_1} \overset{\mathbf{A}}{\mapsto} \mathbf{x_2}$	$\mathbf{o_1} \overset{\mathbf{A}}{\mapsto} \mathbf{x_3}\ \mathbf{x_4}$

sin bcp

main		**p1**
$main \overset{A}{\mapsto} x_2$	$x_0 \overset{B}{\mapsto} o_1$	$x_3 \overset{I}{\mapsto} \lambda y_2.y_2$
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_2 \overset{I}{\mapsto} x_0\#x_1$	$x_4 \overset{B}{\mapsto} i_1$
	$i_1 \overset{A}{\mapsto} x_2$	$o_1 \overset{A}{\mapsto} x_3\ x_4$

BCP

main		**p1**
$main \overset{A}{\mapsto} x_2$	$x_0 \overset{B}{\mapsto} o_1$	$x_3 \overset{I}{\mapsto} \lambda y_2.y_2$
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$\mathbf{x_2} \overset{\mathbf{B}}{\mapsto} \mathbf{x_0}\#\mathbf{x_1}$	$x_4 \overset{B}{\mapsto} i_1$
	$i_1 \overset{A}{\mapsto} x_2$	$o_1 \overset{A}{\mapsto} x_3\ x_4$

PARALELO

main		**p1**
$main \overset{\mathbf{B}}{\mapsto} \mathbf{x_2}$	$x_0 \overset{B}{\mapsto} o_1$	$x_3 \overset{I}{\mapsto} \lambda y_2.y_2$
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$\mathbf{x_2} \overset{\mathbf{A}}{\mapsto} \mathbf{x_0}\#\mathbf{x_1}$	$x_4 \overset{B}{\mapsto} i_1$
	$\mathbf{i_1} \overset{\mathbf{B}}{\mapsto} \mathbf{x_2}$	$\mathbf{o_1} \overset{\mathbf{A}}{\mapsto} \mathbf{x_4}$

PARALELO

main		**p1**
$main \overset{\mathbf{B}}{\mapsto} \mathbf{x_2}$	$x_0 \overset{B}{\mapsto} o_1$	$x_3 \overset{I}{\mapsto} \lambda y_2.y_2$
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$\mathbf{x_2} \overset{\mathbf{B}}{\mapsto} \mathbf{x_0}\#\mathbf{x_1}$	$x_4 \overset{B}{\mapsto} i_1$
	$\mathbf{i_1} \overset{\mathbf{B}}{\mapsto} \mathbf{x_2}$	$\mathbf{o_1} \overset{\mathbf{A}}{\mapsto} \mathbf{x_4}$

PARALELO

main		**p1**
$main \overset{B}{\mapsto} x_2$	$x_0 \overset{B}{\mapsto} o_1$	$x_3 \overset{I}{\mapsto} \lambda y_2.y_2$
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_2 \overset{A}{\mapsto} x_0\#x_1$	$x_4 \overset{B}{\mapsto} i_1$
	$i_1 \overset{B}{\mapsto} x_2$	$\mathbf{o_1} \overset{\mathbf{B}}{\mapsto} \mathbf{x_4}$

PARALELO

main		**p1**
$main \overset{B}{\mapsto} x_2$	$x_0 \overset{B}{\mapsto} o_1$	$x_3 \overset{I}{\mapsto} \lambda y_2.y_2$
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_2 \overset{B}{\mapsto} x_0\#x_1$	$x_4 \overset{B}{\mapsto} i_1$
	$i_1 \overset{B}{\mapsto} x_2$	$\mathbf{o_1} \overset{\mathbf{B}}{\mapsto} \mathbf{x_4}$

↓

Pese a que ninguna regla puede aplicarse al quedar una ligadura activa la ejecución continúa indefinidamente.

↓

Al no quedar ninguna ligadura activa se interrumpe la ejecución. □

Como se ha mostrado en los ejemplos no se puede prescindir de la regla *bloqueo de creación de proceso*, ya que, entonces, se podría entrar en un bucle infinito en el que el sistema, sin estar bloqueado, no pudiera evolucionar.

Capítulo 3

Preparación

Este capítulo es la preparación del plato fuerte del trabajo, que se expondrá en el Capítulo 4, donde se realizarán las demostraciones de una serie de propiedades de la semántica operacional de Jauja básico: *corrección, adecuación* y *determinación*. Estas demostraciones se harán siguiendo los pasos dados en [4]. Para ello es necesario adaptar convenientemente las semánticas expuestas en el capítulo anterior. Por un lado, se definirá una restricción de la semántica operacional de Jauja básico para el caso en que sólo se dispone de un procesador, y por otro lado, se extenderá la semántica natural de Launchbury para incorporar las creaciones de proceso, demostrando que se trata de una extensión consistente con la semántica explicada en la Sección 2.1.

3.1. Semántica operacional para un procesador

Para demostrar la equivalencia entre la semántica operacional de Jauja básico y una semántica denotacional clásica como la de Abramsky [1] se va a recurrir a la semántica natural, para aprovechar el trabajo de Launchbury que en [21] ya demostró la correspondencia de su semántica operacional con esta semántica denotacional. Para esto, primero hay que relacionar la semántica operacional de Jauja básico con la semántica natural. Como la semántica natural está definida para un solo procesador, lo primero que se hará será modificar la semántica de Jauja básico adecuándola al caso en el que se disponga de un único procesador. De esta forma será más sencillo establecer la relación entre ambas semánticas.

La restricción a un único procesador supone la secuenciación del cómputo. Tradicionalmente la secuenciación de un cómputo paralelo se realiza mediante técnicas de intercalación (*interleaving*), de forma que un cómputo paralelo da lugar a un conjunto de cómputos secuenciales que representan las diferentes formas permitidas de intercalar las acciones básicas. Por ejemplo, la ejecución en paralelo de las acciones a y b produce las intercalaciones ab y ba. Esto da lugar, en general, a no determinismo. Pero en este caso concreto, se ha eliminado el

$$\text{(CPI)}\quad \frac{H : \theta \overset{I}{\mapsto} x\#y \overset{CPI}{\longrightarrow}_1}{\overset{CPI}{\longrightarrow}_1\ H + \{\theta \overset{I}{\mapsto} o,\ i \overset{I}{\mapsto} y,\ o \overset{I}{\mapsto} \eta(x)\ z,\ z \overset{I}{\mapsto} i\} + \eta(\mathsf{nh}(x, H))} \quad \begin{array}{l}\text{si } \neg\mathsf{d}(x, H + \{\theta \overset{I}{\mapsto} x\#y\}) \\ o, i, z, \eta \text{ frescos}\end{array}$$

Figura 3.1: Regla de impaciencia en la creación de procesos

operador no determinista de Jauja, de forma que los cómputos son deterministas con respecto al valor final alcanzado para la expresión inicial, aunque el número de procesos creados y las ligaduras evaluadas puedan variar de un cómputo a otro, en el rango permitido por las semánticas mínima y máxima (Sección 2.3.3).

Además, en esta primera aproximación, para estudiar la equivalencia con respecto a una semántica denotacional sólo se considera el valor obtenido, por lo que para la secuenciación se impondrá un orden de evaluación compatible con la semántica mínima.

Para expresar la secuenciación se permitirá que haya como máximo una ligadura activa en cada momento (si no hay ninguna significa que el cómputo se ha bloqueado y no dará lugar a ningún valor). De esta forma la regla $\overset{par}{\Longrightarrow}$ (Sección 2.3.3) deja de tener sentido. Es más, ya no es necesaria la distinción de las reglas en dos niveles. Por otra parte, gracias a las copias con renombramiento en la creación de procesos y en la comunicación de valores a través de canales, los conjuntos de ligaduras en cada proceso son disjuntos entre sí, de forma que se pueden reunir todas las ligaduras en un *heap* etiquetado común único y el concepto de sistema formado por varios procesos también desaparece.

Otra de las características principales de Jauja es la impaciencia asociada a la creación de procesos que da lugar al paralelismo especulativo. En el caso de un solo procesador esta característica también pierde su sentido por la connotación de paralelismo que lleva implícita. Sin embargo, se desea adaptar la semántica de Jauja básico al caso de un procesador intentando no perder todas sus propiedades. Además, se pretende continuar este trabajo hasta lograr demostrar una equivalencia entre semánticas más completa. Por tanto, se va a reflejar la impaciencia de la creación de procesos mediante la regla *creación de proceso impaciente*, *CPI* (Figura 3.1), que no hace evolucionar la ligadura activa sino que sólo produce cambios en el *heap* etiquetado añadiendo las ligaduras que se generarían al crear un proceso aunque, claro está, sin activar ninguna nueva ligadura. Puede verse como una regla que va a preparar el *heap* etiquetado para los pasos futuros creando ligaduras que pueden necesitarse más adelante, es decir, los *heaps* etiquetados se modifican de forma especulativa. Una creación de proceso se realiza cuando no tenga dependencias, es decir, cuando en su *heap* necesario no exista ninguna variable o canal que esté a la espera de alguna comunicación u otra creación de proceso (la función $\mathsf{dependencia}\ \_\mathsf{d}$ definida en la figura 2.8—, calcula, como su nombre indica, si existen o no algunas dependencias de ese tipo).

En la Figura 3.2 se muestran las reglas de Jauja básico adaptadas al caso particular en el que se dispone de un único procesador. Todas estas reglas hacen evolucionar la ligadura

(valor)	$H + \{x \overset{I}{\mapsto} W\} \ : \ \theta \overset{A}{\mapsto} x \ \longrightarrow_1 \ H + \{x \overset{I}{\mapsto} W,\ \theta \overset{A}{\mapsto} W\}$	
(demanda)	$H + \{\tau \overset{IB}{\mapsto} E\} \ : \ \theta \overset{A}{\mapsto} \tau \ \longrightarrow_1 \ H + \{\tau \overset{AB}{\mapsto} E,\ \theta \overset{B(\tau)}{\mapsto} \tau\}$	si $E \notin \mathit{Val}$
(bloqueo I)	$H \ : \ x \overset{A}{\mapsto} x \ \longrightarrow_1 \ H + \{x \overset{B(x)}{\mapsto} x\}$	
(dem. aplic.)	$H + \{x \overset{IB}{\mapsto} E\} \ : \ \theta \overset{A}{\mapsto} x\ y \ \longrightarrow_1 \ H + \{x \overset{AB}{\mapsto} E,\ \theta \overset{B(x)}{\mapsto} x\ y\}$	si $E \notin \mathit{Val}$
(β-reducción)	$H + \{x \overset{I}{\mapsto} \lambda z.E\} \ : \ \theta \overset{A}{\mapsto} x\ y \ \longrightarrow_1 \ H + \{x \overset{I}{\mapsto} \lambda z.E,\ \theta \overset{A}{\mapsto} E[y/z]\}$	
(let)	$H : \ \theta \overset{A}{\mapsto} \mathtt{let}\ \{x_i = E_i\}_{i=1}^n\ \mathtt{in}\ x \longrightarrow_1$ $\longrightarrow_1 \ H + \{y_i \overset{I}{\mapsto} E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n + \{\theta \overset{A}{\mapsto} x[y_j/x_j]_{j=1}^n\}$	$1 \le i, j \le n$, y_i frescas
(desb-desact)	$H + \{\theta \overset{B(\tau)}{\mapsto} E\} : \tau \overset{A}{\mapsto} W \longrightarrow_1 \ H + \{\tau \overset{I}{\mapsto} W,\ \theta \overset{A}{\mapsto} E\}$	
(creac. proc.)	$H : \theta \overset{A}{\mapsto} x\#y \longrightarrow_1$ $\longrightarrow_1 \ H + \{\theta \overset{A}{\mapsto} o,\ i \overset{I}{\mapsto} y,\ o \overset{I}{\mapsto} \eta(x)\ z,\ z \overset{I}{\mapsto} i\} + \eta(\mathsf{nh}(x, H))$	si $\neg\mathsf{d}(x, H^*)$, o, i, z, η frescos $H^* = H + \{\theta \overset{A}{\mapsto} x\#y\}$
(comunicación)	$H + \{ch \overset{I}{\mapsto} W\} \ : \ \theta \overset{A}{\mapsto} ch \ \longrightarrow_1 \ H + \{\theta \overset{A}{\mapsto} \eta(W)\} + \eta(\mathsf{nh}(W, H))$	si $\neg\mathsf{d}(W, H)$
(dem. cp)	$H + \{z \overset{IB}{\mapsto} E\} : \theta \overset{A}{\mapsto} x\#y \longrightarrow_1 \ H + \{z \overset{AB}{\mapsto} E,\ \theta \overset{B(z)}{\mapsto} x\#y\}$	si $\mathsf{d}(x, H^*) \wedge$ $\mathsf{pld}(x, H^*) = z \overset{IB}{\mapsto} E$, $H^* = H + \{z \overset{IB}{\mapsto} E, \theta \overset{A}{\mapsto} x\#y\}$
(bloqueo II)	$H \ : \ \theta \overset{A}{\mapsto} x\#y \ \longrightarrow_1 \ H + \{\theta \overset{B(\theta)}{\mapsto} x\#y\}$	si $\mathsf{d}(x, H^*) \wedge$ $\mathsf{pld}(x, H^*) = \theta \overset{A}{\mapsto} x\#y$ $H^* = H + \{\theta \overset{A}{\mapsto} x\#y\}$
(dem. com.)	$H + \{z \overset{IB}{\mapsto} E, ch \overset{I}{\mapsto} W\} : \theta \overset{A}{\mapsto} ch \longrightarrow_1 \ H + \{z \overset{AB}{\mapsto} E,\ ch \overset{I}{\mapsto} W, \theta \overset{B(z)}{\mapsto} ch\}$	si $\mathsf{d}(W, H^*) \wedge$ $\mathsf{pld}(W, H^*) = z \overset{IB}{\mapsto} E$ $H^* = H + \{z \overset{IB}{\mapsto} E\}$

donde $H \in \mathit{EHeap}$; $E, E_i \in \mathit{EExp} \cup \mathcal{C}\mathit{han}$; $W \in \mathit{Val}$;
$x, y, z \in \mathit{Var}$; $\theta, \tau \in \mathit{Var} \cup \mathcal{C}\mathit{han}$; $i, o, ch \in \mathcal{C}\mathit{han}$

Figura 3.2: Reglas de la semántica operacional para un procesador

activa en el momento, modificando y/o ampliando si fuera necesario el *heap* etiquetado de partida. Las transiciones de un solo paso que se aplican para pasar de un *heap* etiquetado a

otro se descomponen de la siguiente forma:

$$\Longrightarrow_1 \;=\; \longrightarrow_1;\; \overset{CPI}{\Longrightarrow}_1,$$

es decir, en cada paso se realiza una aplicación simple de una regla $\longrightarrow_1$ seguida de la regla $\overset{CPI}{\Longrightarrow}_1$, que representa la aplicación reiterada de la regla *creación de proceso impaciente*, *CPI*, hasta que no pueda ser aplicada más veces (incluyendo el caso en el que no se puede aplicar ninguna vez). Se demuestra (Proposición 3.1.3) que si dos creaciones de proceso son factibles, tras realizar una de ellas la otra seguirá siendo factible (considerando que una creación de proceso es factible si no tiene dependencias).

DEFINICIÓN 3.1.1. Sean $H \in \mathit{EHeap}$ un *heap* etiquetado y $\theta \overset{\alpha}{\mapsto} x\#y \in H$. Se dice que la creación de proceso $x\#y$ es *factible* si $\neg \mathsf{d}(x, H)$.

□

En las reglas de la figura 3.2 las etiquetas de las ligaduras bloqueadas se han modificado para incorporar información sobre la causa del bloqueo. Es decir, ahora una ligadura bloqueada es de la forma $\theta \overset{B(\tau)}{\mapsto} E$, que indica que la ligadura está bloqueada porque necesita el valor de τ para continuar con su evaluación. Esta información adicional es necesaria para el desbloqueo de las ligaduras. Para ello cada vez que se produzca un bloqueo se indicará en la etiqueta el canal o variable en el que la ligadura ha sido bloqueada. Por ejemplo, en la regla *demanda* la ligadura de θ queda bloqueada en la variable o canal τ y por eso se tiene $\theta \overset{B(\tau)}{\mapsto} \tau$, mientras que en la regla *demanda de aplicación* se bloquea en la variable x y se tiene $\theta \overset{B(x)}{\mapsto} x\ y$.

Las primeras reglas de esta figura provienen de las reglas locales de la semántica operacional de Jauja básico (véase figura 2.6) y no sufren modificaciones salvo por el hecho de que, como ya se ha comentado antes, en cada momento puede haber tan solo una ligadura activa, y por lo tanto ni en la parte izquierda ni en la parte derecha de las reglas se da la opción de que haya más de una ligadura en ese estado. En la semántica dada en la Sección 2.3.2 todos los canales se mantenían activos, pero ahora sólo se activan bajo demanda, por lo que la regla *demanda* ya no es específica para la demanda de variables sino que también pueden demandarse canales y por ello $x \in \mathit{Var}$ se ha cambiado por $\tau \in \mathit{Var} \cup \mathcal{C}\mathit{han}$.

En el caso general de múltiples procesadores, puede ocurrir que varias ligaduras estén bloqueadas en la misma variable, por lo que son necesarias dos etapas: primero se desbloquean todas las ligaduras que se puedan y después se desactivan aquellas que ya tienen asociado un valor en *whnf*. Pero cuando se limitan los recursos a un único procesador, como máximo habrá una ligadura bloqueada en cada variable (ver Proposición 3.1.12), así que se pueden realizar el desbloqueo y la desactivación en un único paso, formando así la regla *desbloqueo y desactivación*.

En las reglas *creación de proceso* y *comunicación* el sistema y los procesos involucrados (padre/hijo o productor/consumidor) ya no aparecen por la desaparición del concepto de

sistema explicado anteriormente. Esto no supone ningún problema, pues se garantiza que cada canal aparece exactamente en dos ligaduras del *heap* etiquetado común: una en el lado derecho y otra en el lado izquierdo (ver Proposición 3.1.14). La regla *creación de proceso* refleja la creación de un proceso que ha sido demandado y que no está a la espera de ninguna comunicación ni depende de otra creación de proceso (estas condiciones se hallan mediante la función dependencia). En las reglas globales dadas para varios procesadores (Figura 2.7), al efectuar una creación de proceso, en el *heap* resultante se activan los canales y se bloquea la variable que dió lugar a la creación; sin embargo, en el caso de un solo procesador, cuando una creación de proceso se realiza porque es demandada conviene mantener activa la ligadura de la que provenía, es decir, el canal o la variable θ que estaba activo en el *heap* de partida en la forma $\theta \overset{A}{\mapsto} x\#y$, continuará activo en el *heap* resultante, aunque ahora ligado al nuevo canal o. Este pequeño cambio simplemente implica que ahora se realiza en dos pasos lo que anteriormente se realizaba en uno. En la regla *comunicación* la comunicación sólo se lleva a cabo si el canal está activo, y esto sólo ocurre si su valor ha sido demandado, es decir, es necesario para calcular el valor de la expresión principal. Al igual que en el caso de la creación de procesos, una comunicación sólo podrá realizarse si carece de dependencias.

Hay que tener en cuenta que ya no hay demanda impaciente de los valores ligados a los canales de comunicación por lo que son necesarias algunas reglas para iniciar la demanda de estas comunicaciones en el caso de necesitarlas. Esto da lugar a las dos reglas: *demanda de creación de proceso* y *demanda de comunicación*, que se ocupan de bloquear respectivamente la creación de proceso o la comunicación que no puede realizarse (por tener dependencias pendientes) y de activar la ligadura correspondiente a la primera variable de la que dependan (que estará ligada a un canal o a una creación de proceso). En el caso de la demanda de creación de proceso puede ocurrir que el proceso que ha de crearse dependa de sí mismo; este es un caso particular de bloqueo que viene reflejado en la regla *bloqueo II*, en este caso y, al igual que ocurre en la regla *bloqueo I*, la ligadura activa se bloquea en la propia variable.

La primera dependencia necesaria para poder aplicar las reglas *demanda de creación de proceso* y *demanda de comunicación*, se calcula aplicando la función primera ligadura de dependencia (pld). Esta función viene definida en la Figura 3.3 y es la responsable de indicar el orden en que deben activarse las comunicaciones y creaciones de procesos pendientes. La función pld viene definida en términos de dos funciones auxiliares. Por un lado, la función lista de expresiones dependientes (ldep) calcula la lista de las subexpresiones de una expresión dada. Por otro lado, la función primera ligadura (priml) que, dada una lista de expresiones y un *heap* etiquetado, calcula la primera ligadura $x \overset{\alpha}{\mapsto} E$ (con $E \equiv ch$ o $E \equiv y\#z$) de la que depende la lista, empezando la búsqueda por la cabeza y donde $\equiv$ representa la equivalencia sintáctica.

Las cuatro reglas de demanda (*demanda*, *demanda de aplicación*, *demanda de creación de proceso* y *demanda de comunicación*) marcarán la secuenciación del cómputo que será compatible con la semántica mínima. Todas estas reglas de demanda bloquean la ligadura activa, pero es posible que si la ligadura que se necesita evaluar está previamente bloqueada (y, por

$$
\begin{aligned}
\mathsf{ldep}(x) &= [\,] \\
\mathsf{ldep}(\lambda x.E) &= [E] \\
\mathsf{ldep}(E_1\ E_2) &= [E_1, E_2] \\
\mathsf{ldep}(E_1 \# E_2) &= [E_1, E_2] \\
\mathsf{ldep}(\texttt{let}\ \{x_i = E_i\}_{i=1}^{n}\ \texttt{in}\ E) &= [E_1,\ \ldots,\ E_n, E]
\end{aligned}
$$

$$
\mathsf{pld}(x, H) = \begin{cases} x \overset{\alpha}{\mapsto} E & \text{si } x \overset{\alpha}{\mapsto} E \in H \wedge (E \equiv ch \vee E \equiv y\#z) \\ \mathsf{pld}(E, H) & \text{si } x \overset{\alpha}{\mapsto} E \in H \wedge E \not\equiv ch \wedge E \not\equiv y\#z \\ \bot & \text{e.o.c.} \end{cases}
$$

$$
\mathsf{pld}(E, H) = \mathsf{priml}(\mathsf{ldep}(E), H) \quad \text{si } E \notin \mathit{Var}
$$

$$
\mathsf{priml}([\,], H) = \bot
$$

$$
\mathsf{priml}(E : L, H) = \begin{cases} \mathsf{pld}(E, H) & \text{si } \mathsf{pld}(E, H) \neq \bot \\ \mathsf{priml}(L, H) & \text{si } \mathsf{pld}(E, H) = \bot \end{cases}
$$

Figura 3.3: Función de primera dependencia

lo tanto, no se puede activar) el *heap* etiquetado resultante no tenga ninguna ligadura activa. Esto se debe a una circularidad en la demanda y no se obtendrá ningún valor final (ver Proposición 3.1.10).

EJEMPLO 3.1.2. *Comunicación dependiente de un canal.*

Este ejemplo muestra el caso en el que una comunicación no puede realizarse por depender de un canal. A partir de ahora, como en los ejemplos de la Sección 2.3.5, aparecerán en negrita las ligaduras que son nuevas en el *heap* y las que sufran alguna modificación. La expresión a evaluar es la siguiente:

$$
\begin{aligned}
&\texttt{let}\ x_0 = x_1 \# x_2,\ x_1 = \lambda y_1.y_1,\ x_2 = \lambda y_2.y_2, \\
&\qquad x_3 = x_4 \# x_5,\ x_4 = \lambda y_3.y_3,\ x_5 = \lambda y_4.(y_4\ x_0) \\
&\texttt{in}\ x_3
\end{aligned}
$$

La expresión inicial se liga a la variable principal *main* en estado activo, siendo esta la única ligadura en el *heap* etiquetado al comienzo.

$$
\begin{aligned}
&\mathbf{main} \overset{\mathbf{A}}{\mapsto} \texttt{let}\ \mathbf{x_0 = x_1 \# x_2},\ \mathbf{x_1 = \lambda y_1.y_1},\ \mathbf{x_2 = \lambda y_2.y_2}, \\
&\qquad\qquad \mathbf{x_3 = x_4 \# x_5},\ \mathbf{x_4 = \lambda y_3.y_3},\ \mathbf{x_5 = \lambda y_4.(y_4\ x_0)} \\
&\qquad \texttt{in}\ \mathbf{x_3}
\end{aligned}
$$

Se aplica una transición completa que se dividirá en dos pasos, primero se aplicará la regla *let* y después la *creación de proceso impaciente*.

LET

$$\begin{array}{|lll|}\hline
main \overset{\mathbf{A}}{\mapsto} \mathbf{x_3} & & \\
\mathbf{x_0} \overset{\mathbf{I}}{\mapsto} \mathbf{x_1}\#\mathbf{x_2} & \mathbf{x_3} \overset{\mathbf{I}}{\mapsto} \mathbf{x_4}\#\mathbf{x_5} & \\
\mathbf{x_1} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_1}.\mathbf{y_1} & \mathbf{x_4} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_3}.\mathbf{y_3} & \\
\mathbf{x_2} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_2}.\mathbf{y_2} & \mathbf{x_5} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_4}.(\mathbf{y_4}\ \mathbf{x_0}) & \\ \hline
\end{array}$$

CPI

$$\begin{array}{|lll|}\hline
main \overset{A}{\mapsto} x_3 & & \\
\mathbf{x_0} \overset{\mathbf{I}}{\mapsto} \mathbf{o_1} & \mathbf{x_3} \overset{\mathbf{I}}{\mapsto} \mathbf{o_2} & \\
x_1 \overset{I}{\mapsto} \lambda y_1.y_1 & x_4 \overset{I}{\mapsto} \lambda y_3.y_3 & \mathbf{i_1} \overset{\mathbf{I}}{\mapsto} \mathbf{x_2} \\
x_2 \overset{I}{\mapsto} \lambda y_2.y_2 & x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0) & \mathbf{i_2} \overset{\mathbf{I}}{\mapsto} \mathbf{x_5} \\
\mathbf{o_1} \overset{\mathbf{I}}{\mapsto} \mathbf{x_6}\ \mathbf{x_7} & \mathbf{x_6} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_5}.\mathbf{y_5} & \mathbf{x_7} \overset{\mathbf{I}}{\mapsto} \mathbf{i_1} \\
\mathbf{o_2} \overset{\mathbf{I}}{\mapsto} \mathbf{x_8}\ \mathbf{x_9} & \mathbf{x_8} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_6}.\mathbf{y_6} & \mathbf{x_9} \overset{\mathbf{I}}{\mapsto} \mathbf{i_2} \\ \hline
\end{array}$$

Como no quedan creaciones de procesos sin realizar tan sólo se irán aplicando las reglas $\longrightarrow_1$:

DEMANDA

$$\begin{array}{|lll|}\hline
main \overset{\mathbf{B(x_3)}}{\mapsto} \mathbf{x_3} & & \\
x_0 \overset{I}{\mapsto} o_1 & \mathbf{x_3} \overset{\mathbf{A}}{\mapsto} \mathbf{o_2} & \\
x_1 \overset{I}{\mapsto} \lambda y_1.y_1 & x_4 \overset{I}{\mapsto} \lambda y_3.y_3 & i_1 \overset{I}{\mapsto} x_2 \\
x_2 \overset{I}{\mapsto} \lambda y_2.y_2 & x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0) & i_2 \overset{I}{\mapsto} x_5 \\
o_1 \overset{I}{\mapsto} x_6\ x_7 & x_6 \overset{I}{\mapsto} \lambda y_5.y_5 & x_7 \overset{I}{\mapsto} i_1 \\
o_2 \overset{I}{\mapsto} x_8\ x_9 & x_8 \overset{I}{\mapsto} \lambda y_6.y_6 & x_9 \overset{I}{\mapsto} i_2 \\ \hline
\end{array}$$

DEMANDA

$$\begin{array}{|lll|}\hline
main \overset{B(x_3)}{\mapsto} x_3 & & \\
x_0 \overset{I}{\mapsto} o_1 & \mathbf{x_3} \overset{\mathbf{B(o_2)}}{\mapsto} \mathbf{o_2} & \\
x_1 \overset{I}{\mapsto} \lambda y_1.y_1 & x_4 \overset{I}{\mapsto} \lambda y_3.y_3 & i_1 \overset{I}{\mapsto} x_2 \\
x_2 \overset{I}{\mapsto} \lambda y_2.y_2 & x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0) & i_2 \overset{I}{\mapsto} x_5 \\
o_1 \overset{I}{\mapsto} x_6\ x_7 & x_6 \overset{I}{\mapsto} \lambda y_5.y_5 & x_7 \overset{I}{\mapsto} i_1 \\
\mathbf{o_2} \overset{\mathbf{A}}{\mapsto} \mathbf{x_8}\ \mathbf{x_9} & x_8 \overset{I}{\mapsto} \lambda y_6.y_6 & x_9 \overset{I}{\mapsto} i_2 \\ \hline
\end{array}$$

β-REDUCCIÓN

$$\begin{array}{|lll|}\hline
main \overset{B(x_3)}{\mapsto} x_3 & & \\
x_0 \overset{I}{\mapsto} o_1 & x_3 \overset{B(o_2)}{\mapsto} o_2 & \\
x_1 \overset{I}{\mapsto} \lambda y_1.y_1 & x_4 \overset{I}{\mapsto} \lambda y_3.y_3 & i_1 \overset{I}{\mapsto} x_2 \\
x_2 \overset{I}{\mapsto} \lambda y_2.y_2 & x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0) & i_2 \overset{I}{\mapsto} x_5 \\
o_1 \overset{I}{\mapsto} x_6\ x_7 & x_6 \overset{I}{\mapsto} \lambda y_5.y_5 & x_7 \overset{I}{\mapsto} i_1 \\
\mathbf{o_2} \overset{\mathbf{A}}{\mapsto} \mathbf{x_9} & x_8 \overset{I}{\mapsto} \lambda y_6.y_6 & x_9 \overset{I}{\mapsto} i_2 \\ \hline
\end{array}$$

DEMANDA

$$\begin{array}{|lll|}\hline
main \overset{B(x_3)}{\mapsto} x_3 & & \\
x_0 \overset{I}{\mapsto} o_1 & x_3 \overset{B(o_2)}{\mapsto} o_2 & \\
x_1 \overset{I}{\mapsto} \lambda y_1.y_1 & x_4 \overset{I}{\mapsto} \lambda y_3.y_3 & i_1 \overset{I}{\mapsto} x_2 \\
x_2 \overset{I}{\mapsto} \lambda y_2.y_2 & x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0) & i_2 \overset{I}{\mapsto} x_5 \\
o_1 \overset{I}{\mapsto} x_6\ x_7 & x_6 \overset{I}{\mapsto} \lambda y_5.y_5 & x_7 \overset{I}{\mapsto} i_1 \\
\mathbf{o_2} \overset{\mathbf{B(x_9)}}{\mapsto} \mathbf{x_9} & x_8 \overset{I}{\mapsto} \lambda y_6.y_6 & \mathbf{x_9} \overset{\mathbf{A}}{\mapsto} \mathbf{i_2} \\ \hline
\end{array}$$

DEMANDA

$$\begin{array}{|lll|}\hline
main \overset{B(x_3)}{\mapsto} x_3 & & \\
x_0 \overset{I}{\mapsto} o_1 & x_3 \overset{B(o_2)}{\mapsto} o_2 & \\
x_1 \overset{I}{\mapsto} \lambda y_1.y_1 & x_4 \overset{I}{\mapsto} \lambda y_3.y_3 & i_1 \overset{I}{\mapsto} x_2 \\
x_2 \overset{I}{\mapsto} \lambda y_2.y_2 & x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0) & \mathbf{i_2} \overset{\mathbf{A}}{\mapsto} \mathbf{x_5} \\
o_1 \overset{I}{\mapsto} x_6\ x_7 & x_6 \overset{I}{\mapsto} \lambda y_5.y_5 & x_7 \overset{I}{\mapsto} i_1 \\
o_2 \overset{B(x_9)}{\mapsto} x_9 & x_8 \overset{I}{\mapsto} \lambda y_6.y_6 & \mathbf{x_9} \overset{\mathbf{B(i_2)}}{\mapsto} \mathbf{i_2} \\ \hline
\end{array}$$

VALOR

$$\begin{array}{|lll|}\hline
main \overset{B(x_3)}{\mapsto} x_3 & & \\
x_0 \overset{I}{\mapsto} o_1 & x_3 \overset{B(o_2)}{\mapsto} o_2 & \\
x_1 \overset{I}{\mapsto} \lambda y_1.y_1 & x_4 \overset{I}{\mapsto} \lambda y_3.y_3 & i_1 \overset{I}{\mapsto} x_2 \\
x_2 \overset{I}{\mapsto} \lambda y_2.y_2 & x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0) & \mathbf{i_2} \overset{\mathbf{A}}{\mapsto} \lambda\mathbf{y_4}.(\mathbf{y_4}\ \mathbf{x_0}) \\
o_1 \overset{I}{\mapsto} x_6\ x_7 & x_6 \overset{I}{\mapsto} \lambda y_5.y_5 & x_7 \overset{I}{\mapsto} i_1 \\
o_2 \overset{B(x_9)}{\mapsto} x_9 & x_8 \overset{I}{\mapsto} \lambda y_6.y_6 & x_9 \overset{B(i_2)}{\mapsto} i_2 \\ \hline
\end{array}$$

El canal i_2 está ligado a un valor en *whnf*, sin embargo no puede ser transmitido porque depende del valor de x_0 que está ligado al canal o_1. La función pld devuelve la primera ligadura que ha de activarse: $x_0 \overset{I}{\mapsto} o_1$.

DEMANDA DE COMUNICACIÓN

$main \overset{B(x_3)}{\mapsto} x_3$		
$\mathbf{x_0} \overset{\mathbf{A}}{\mapsto} \mathbf{o_1}$	$x_3 \overset{B(o_2)}{\mapsto} o_2$	
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$	$i_1 \overset{I}{\mapsto} x_2$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0)$	$\mathbf{i_2} \overset{\mathbf{B(x_0)}}{\mapsto} \lambda\mathbf{y_4}.(\mathbf{y_4\ x_0})$
$o_1 \overset{I}{\mapsto} x_6\ x_7$	$x_6 \overset{I}{\mapsto} \lambda y_5.y_5$	$x_7 \overset{I}{\mapsto} i_1$
$o_2 \overset{B(x_9)}{\mapsto} x_9$	$x_8 \overset{I}{\mapsto} \lambda y_6.y_6$	$x_9 \overset{B(i_2)}{\mapsto} i_2$

Tras varios pasos de cómputo se obtiene el *heap* etiquetado siguiente:

$main \overset{B(x_3)}{\mapsto} x_3$		
$x_0 \overset{A}{\mapsto} \lambda y_8.y_8$	$x_3 \overset{B(o_2)}{\mapsto} o_2$	$x_8 \overset{I}{\mapsto} \lambda y_6.y_6$
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$	$x_9 \overset{B(i_2)}{\mapsto} i_2$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0)$	$i_2 \overset{B(x_0)}{\mapsto} \lambda y_4.(y_4\ x_0)$
$o_2 \overset{B(x_9)}{\mapsto} x_9$	$x_6 \overset{I}{\mapsto} \lambda y_5.y_5$	$x_7 \overset{I}{\mapsto} \lambda y_7.y_7$

Como x_0 ya está ligado a un valor en *whnf* sin dependencias entonces i_2 ya no depende de ningún canal ni creación de proceso; primero se activa la ligadura y en el siguiente paso se podrá comunicar dicho valor.

DESBLOQUEO-DESACTIVACIÓN

$main \overset{B(x_3)}{\mapsto} x_3$		
$\mathbf{x_0} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_8}.\mathbf{y_8}$	$x_3 \overset{B(o_2)}{\mapsto} o_2$	$x_8 \overset{I}{\mapsto} \lambda y_6.y_6$
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$	$x_9 \overset{B(i_2)}{\mapsto} i_2$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0)$	$\mathbf{i_2} \overset{\mathbf{A}}{\mapsto} \lambda\mathbf{y_4}.(\mathbf{y_4\ x_0})$
$o_2 \overset{B(x_9)}{\mapsto} x_9$	$x_6 \overset{I}{\mapsto} \lambda y_5.y_5$	$x_7 \overset{I}{\mapsto} \lambda y_7.y_7$

COMUNICACIÓN

$main \overset{B(x_3)}{\mapsto} x_3$		
$x_0 \overset{I}{\mapsto} \lambda y_8.y_8$	$x_3 \overset{B(o_2)}{\mapsto} o_2$	$x_8 \overset{I}{\mapsto} \lambda y_6.y_6$
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$	$\mathbf{x_9} \overset{\mathbf{A}}{\mapsto} \lambda\mathbf{y_9}.(\mathbf{y_9\ x_{10}})$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0)$	$\mathbf{x_{10}} \overset{\mathbf{I}}{\mapsto} \lambda\mathbf{y_{10}}.\mathbf{y_{10}}$
$o_2 \overset{B(x_9)}{\mapsto} x_9$	$x_6 \overset{I}{\mapsto} \lambda y_5.y_5$	$x_7 \overset{I}{\mapsto} \lambda y_7.y_7$

Tras varios pasos se alcanza el *heap* etiquetado final.

$main \overset{\mathbf{A}}{\mapsto} \lambda\mathbf{y_{11}}.(\mathbf{y_{11}}\ \mathbf{x_{12}})$		
$x_0 \overset{I}{\mapsto} \lambda y_8.y_8$	$x_3 \overset{I}{\mapsto} \lambda y_{11}.(y_{11}\ x_{12})$	$x_8 \overset{I}{\mapsto} \lambda y_6.y_6$
$x_1 \overset{I}{\mapsto} \lambda y_1.y_1$	$x_4 \overset{I}{\mapsto} \lambda y_3.y_3$	$x_9 \overset{I}{\mapsto} \lambda y_9.(y_9\ x_{10})$
$x_2 \overset{I}{\mapsto} \lambda y_2.y_2$	$x_5 \overset{I}{\mapsto} \lambda y_4.(y_4\ x_0)$	$x_{10} \overset{I}{\mapsto} \lambda y_{10}.y_{10}$
$x_{11} \overset{I}{\mapsto} \lambda y_{11}.y_{11}$	$x_6 \overset{I}{\mapsto} \lambda y_5.y_5$	$x_7 \overset{I}{\mapsto} \lambda y_7.y_7$

□

3.1.1. Propiedades

A continuación se demostrarán algunas propiedades de la semántica que acabamos de definir. Se empieza demostrando que no importa el orden en el que se realizan las creaciones de procesos.

PROPOSICIÓN 3.1.3. *Sean $H \in EHeap$ un heap etiquetado y $\theta_1 \overset{I}{\mapsto} x_1\#y_1, \theta_2 \overset{I}{\mapsto} x_2\#y_2 \in H$ dos creaciones de proceso factibles tales que $\theta_1 \neq \theta_2$. Después de crear $x_1\#y_1$, $x_2\#y_2$ sigue siendo factible.*

Demostración

Si $x_2\#y_2$ es factible entonces x_2 no tiene dependencias, es decir, $\neg\mathsf{d}(x_2, H)$. Hay que probar que después de crear $x_1\#y_1$ x_2 sigue sin tener dependencias.

Tras crear $x_1\#y_1$ el *heap* etiquetado $H = H^* + \{\theta_1 \overset{I}{\mapsto} x_1\#y_1\}$ se transforma en $H' = H^* + \{\theta_1 \overset{I}{\mapsto} o_1, i_1 \overset{I}{\mapsto} y_1, o_1 \overset{I}{\mapsto} \eta(x_1), z_1 \overset{I}{\mapsto} i_1\} + \eta(\mathsf{nh}(x_1, H))$. Si x_2 no tenía dependencias antes de crear $x_1\#y_1$, entonces no dependía de θ_1. Ahora no se ha modificado ninguna ligadura de H (salvo θ_1 que ahora está ligada a un canal), por lo que si x_2 antes no dependía de ella ahora tampoco lo hace. Aunque se han añadido nuevas ligaduras, estas corresponden a variables y canales nuevos, por lo que x_2 tampoco depende de ellas. Así que $\neg\mathsf{d}(x_2, H')$ y $x_2\#y_2$ sigue siendo factible.

c.q.d.

PROPOSICIÓN 3.1.4. *Sean $H \in EHeap$ un heap etiquetado, $\theta_1 \overset{I}{\mapsto} x_1\#y_1 \in H$ una creación de proceso factible y $\theta_2 \overset{I}{\mapsto} x_2\#y_2 \in H$ una creación de proceso no factible tal que $\mathsf{pId}(x_2, H) = \theta_1 \overset{I}{\mapsto} x_1\#y_1$. Después de crear $x_1\#y_1$, $x_2\#y_2$ sigue siendo no factible.*

Demostración

Tras la creación de proceso correspondiente a $x_1\#y_1$ se tiene que $\theta_1 \overset{I}{\mapsto} o_1$, siendo o_1 un nuevo canal. Así pues, si H' es el nuevo *heap* se tendrá $\mathsf{d}(x_2, H')$ y $\mathsf{pId}(x_2, H') = \theta_1 \overset{I}{\mapsto} o_1$, es decir, sigue sin ser factible la creación de proceso $x_2\#y_2$.

c.q.d.

Como las creaciones de proceso de un *heap* etiquetado no interfieren unas con otras, es decir, cuando una de ellas es factible sigue siéndolo aunque otra se realize antes, y si no es factible sigue sin serlo al procesarse las demás, no importa el orden en el que se efectuen. Este resultado viene reflejado en el siguiente corolario:

COROLARIO 3.1.5. *Sea $H \in EHeap$ un heap etiquetado con varias creaciones de proceso. El orden de aplicación de las reglas $\xrightarrow{CPI}_1$ es irrelevante.*

Demostración
Trivial por las proposiciones 3.1.3 y 3.1.4. c.q.d.

Antes de demostrar otras propiedades se definirán un par de conceptos:

DEFINICIÓN 3.1.6. Sea $H \in EHeap$. Se dice que H es un *heap alcanzable* si $\exists E \in EExp$ tal que

$$\{main \overset{A}{\mapsto} E\} \Longrightarrow_1^* H.$$

□

LEMA 3.1.7. *Sea $H \in EHeap$ un heap etiquetado. Si H es alcanzable entonces $H \overset{CPI}{\not\longrightarrow}_1$*

Demostración
Trivial por las definiciones de *heap* alcanzable y de $\Longrightarrow_1$. Si $H \xrightarrow{CPI}_1 H'$ entonces H no sería alcanzable pues la transición $\Longrightarrow_1 = \longrightarrow_1; \overset{CPI}{\Longrightarrow}_1$ no se habría completado, contradiciendo la definición de *heap* alcanzable. c.q.d.

DEFINICIÓN 3.1.8. Sea $H \in EHeap$ un *heap* etiquetado. Se definen los siguientes conjuntos:

- *ligaduras inactivas de H*: $\mathcal{LI}(H) = \{\theta \overset{I}{\mapsto} E \mid \theta \overset{I}{\mapsto} E \in H\}$;
- *ligaduras bloqueadas de H*: $\mathcal{LB}(H) = \{\theta \overset{B(\tau)}{\mapsto} E \mid \theta \overset{B(\tau)}{\mapsto} E \in H\}$;
- *ligaduras activas de H*: $\mathcal{LA}(H) = \{\theta \overset{A}{\mapsto} E \mid \theta \overset{A}{\mapsto} E \in H\}$;
- *variables que aparecen en H*: $Var(H) = \{x \in Var \mid x \overset{\alpha}{\mapsto} E \in H\}$;
- *canales que aparecen en H*: $Chan(H) = \{ch \in \mathcal{C}han \mid ch \overset{\alpha}{\mapsto} E \in H \vee \theta \overset{\alpha}{\mapsto} ch \in H\}$.

□

La siguiente propiedad indica que en cada *heap* alcanzable como mucho hay una ligadura activa.

PROPOSICIÓN 3.1.9. *Sea $H \in EHeap$ un heap alcanzable. Entonces $|\mathcal{LA}(H)| \leq 1$, donde $|\ .\ | : Set \to \mathbb{N}$ es la función cardinal de un conjunto.*

Demostración
Trivial por inducción sobre las reglas. c.q.d.

PROPOSICIÓN 3.1.10. *Sean $H \in EHeap$ un heap etiquetado alcanzable y $\mathcal{LAB}(H) = \mathcal{LA}(H) \cup \mathcal{LB}(H)$ el conjunto formado por las ligaduras activas y bloqueadas de H. Este conjunto se puede organizar en una secuencia dada por:*

$$C_H = \{main \overset{B(\tau_1)}{\mapsto} E_1,\ \tau_1 \overset{B(\tau_2)}{\mapsto} E_2,\ \ldots, \tau_{n-1} \overset{B(\tau_n)}{\mapsto} E_n, \tau_n \overset{\alpha}{\mapsto} E_{n+1}\}$$

donde la última ligadura es

1. *o bien* $\tau_n \overset{A}{\mapsto} E_{n+1}$,
2. *o bien* $\tau_n \overset{B(\tau_{n+1})}{\mapsto} E_{n+1}$, *donde* $\tau_{n+1} \in \{\tau_i \mid 1 \leq i \leq n\}$.

Demostración
Se demuestra fácilmente por inducción sobre la aplicación de las reglas. c.q.d.

C_H se denomina *cadena de ligaduras necesarias para la evaluación de main en* H, y significa que a partir de la variable *main*, se va generando una cadena de bloqueos que indica las variables o canales que deben ser evaluados para la obtención del valor asociado a *main*. La última ligadura de C_H puede estar activa o bloqueada. En el caso de estar bloqueada indica que se ha producido algún tipo de circularidad a la hora de evaluar las expresiones y por lo tanto el *heap* etiquetado queda bloqueado y no se puede alcanzar ningún valor para *main*.

DEFINICIÓN 3.1.11. Se dice que $H \in EHeap$ un *heap* etiquetado alcanzable es *desarrollable* si la última ligadura de la secuencia C_H se encuentra en estado activo. En caso contrario se dice que H está *bloqueado*.

□

La siguiente proposición establece que si un *heap* etiquetado no está bloqueado, entonces para cualquier variable o canal, como máximo habrá una ligadura bloqueada en ella.

PROPOSICIÓN 3.1.12. *Sea* $H \in EHeap$ *un heap etiquetado alcanzable y desarrollable, entonces*

$$\forall \theta \in Var \cup \mathcal{C}han : (\tau \overset{B(\theta)}{\mapsto} E \in H \wedge \tau' \overset{B(\theta)}{\mapsto} E' \in H \Rightarrow \tau = \tau' \wedge E \equiv E').$$

Demostración
La demostración se hará por inducción sobre las reglas.
REGLA DEMANDA:
Sea $H' = H + \{\tau \overset{IB}{\mapsto} E, \theta \overset{A}{\mapsto} \tau\}$, se bloquea $\theta \overset{B(\tau)}{\mapsto} \tau$; supongamos que ya existe $\theta' \overset{B(\tau)}{\mapsto} E'$ en H, entonces por la proposición 3.1.10 se tiene la cadena

$$C_H = \{main \overset{B(\tau_1)}{\mapsto} E_1, \ldots, \theta' \overset{B(\tau)}{\mapsto} E', \tau \overset{B(\tau')}{\mapsto} E, \ldots, \theta \overset{A}{\mapsto} \tau\}$$

Es decir, τ está bloqueada, así que al aplicar la regla, el heap alcanzado no es desarrollable (ya que todas las ligaduras están bloqueadas).
Para las reglas *demanda de aplicación, demanda cp* y *demanda de comunicación* el razonamiento es análogo. El resto de reglas son triviales. c.q.d.

Si una ligadura $\theta \overset{B(\tau)}{\mapsto} E$ está bloqueada en τ, es necesario que τ esté evaluado para poder desbloquear θ.

PROPOSICIÓN 3.1.13. *Sea $H \in EHeap$ un heap etiquetado alcanzable. Si $H + \{\theta \overset{B(\tau)}{\mapsto} E\} \Longrightarrow_1^* H' + \{\theta \overset{A}{\mapsto} W\}$, entonces existe un cómputo $H + \{\theta \overset{B(\tau)}{\mapsto} E\} \Longrightarrow_1^* H'' + \{\theta \overset{B(\tau)}{\mapsto} E, \tau \overset{A}{\mapsto} W'\}$.*

Demostración
Se observa que la única regla que desbloquea las ligaduras es *desbloqueo y desactivación* y, para poder aplicarla, si $\theta \overset{B(\tau)}{\mapsto} E \in H$ entonces es necesario que $\tau \overset{A}{\mapsto} W \in H$. c.q.d.

Si un canal *ch* aparece en un *heap* etiquetado alcanzable H, entonces aparece exactamente dos veces: una en la parte izquierda de una ligadura y otra en la parte derecha de otra ligadura.

PROPOSICIÓN 3.1.14. *Sea $H \in EHeap$ un heap etiquetado alcanzable. Se cumple que*

$$\forall ch \in Chan(H) : (\exists! E.ch \overset{\alpha}{\mapsto} E \in H) \wedge (\exists! \theta.\theta \overset{\alpha}{\mapsto} ch \in H).$$

Demostración
Se demostrará por inducción sobre la obtención del *heap* etiquetado teniendo en cuenta la aplicación de las reglas.

CASO BASE:
Sea $H = \{main \overset{A}{\mapsto} E\}$. Trivial pues $Chan(H) = \emptyset$.

PASO DE INDUCCIÓN:
Supongamos que en un *heap* alcanzable se cumple la propiedad. Se verá que al aplicar cualquier regla se sigue conservando.

- CREACIÓN DE PROCESO:

$$H : \theta \overset{A}{\mapsto} x\#y \longrightarrow_1 H + \{\theta \overset{A}{\mapsto} o,\ i \overset{I}{\mapsto} y,\ \eta(\mathsf{nh}(x, H)),\ o \overset{I}{\mapsto} \eta(x)\ z,\ z \overset{I}{\mapsto} i\}$$

  Sólo podrá ejecutarse si $\neg\mathsf{d}(x, H + \{\theta \overset{A}{\mapsto} x\#y\})$. En esta regla se añaden al *heap* etiquetado dos nuevos canales. Pero cada uno de ellos aparece en la parte derecha de una ligadura y en la parte izquierda de otra ligadura; así se conserva la propiedad. Además se añade al *heap* etiquetado un conjunto de ligaduras dado por $\eta(\mathsf{nh}(x, H))$ pero, puesto que se exige que no haya dependencias de canales, en este conjunto no habrá ligaduras con canales.

- COMUNICACIÓN:

$$H + \{ch \overset{I}{\mapsto} W\} \ :\ \theta \overset{A}{\mapsto} ch \ \longrightarrow_1 \ H + \{\theta \overset{A}{\mapsto} \eta(W)\} + \eta(\mathsf{nh}(W, H))$$

Una vez aplicada la regla la ligadura $ch \overset{I}{\mapsto} W$ desaparece, por lo tanto no hay ninguna ligadura en la que aparezca ch en el lado izquierdo, pero a su vez la ligadura $\theta \overset{A}{\mapsto} ch$ donde aparece el canal en el lado derecho se modifica pasando a $\theta \overset{A}{\mapsto} \eta(W)$, por lo tanto el canal también desaparece del lado derecho. Esta regla sólo puede ser aplicada si se verifica que $\neg\mathsf{d}(W, H)$, es decir, W no depende de ningún canal, por lo tanto en $\eta(\mathsf{nh}(W, H))$, parte que se añade al *heap* , no aparece ningún canal nuevo. Consecuentemente la propiedad se conserva, aunque con un canal menos.

- CREACIÓN DE PROCESO IMPACIENTE:

$$H : \theta \overset{I}{\mapsto} x\#y \xrightarrow{CPI}_1 H + \{\theta \overset{I}{\mapsto} o,\ i \overset{I}{\mapsto} y,\ \eta(\mathsf{nh}(x, H)),\ o \overset{I}{\mapsto} \eta(x)\ z,\ z \overset{I}{\mapsto} i\}$$

Análogo a la regla *creación de proceso.*

En el resto de reglas no se modifica el conjunto de canales del *heap.*

c.q.d.

3.2. Semántica natural extendida

La semántica natural dada en [21], que se ha explicado también en la Sección 2.1, está definida solamente para el núcleo funcional ampliado con la expresión `let`, es decir, el conjunto *Exp*. Es necesario ampliar la semántica natural para dar significado a las creaciones de procesos y de esta forma cubrir todo el conjunto de expresiones extendidas (*EExp*).

Si se quiere calcular el valor asociado a una expresión E en la semántica natural, se comienza evaluando dicha expresión en un *heap* vacío $(\emptyset : E)$ y aplicando las reglas dadas se obtiene una derivación. Al aplicar algunas de las reglas desaparecen del *heap* ciertas ligaduras que pueden considerarse bloqueadas y que están a la espera de que otra variable sea evaluada. Esta pérdida de información dificulta la creación impaciente de procesos. Por ello es necesario ampliar el concepto de *heap* y considerarlo como un par $\langle \Gamma, \Gamma^B \rangle$ donde Γ^B es el conjunto formado por las ligaduras bloqueadas a lo largo de una derivación, es decir, indica las ligaduras que ya han sido demandadas y que están a la espera de algún valor para continuar con su evaluación. De esta forma si se quiere evaluar la expresión E, el *heap* inicial será $\langle \emptyset, \emptyset \rangle$. Además, como en la sección anterior, las variables que están bloqueadas llevarán una anotación indicando la variable o canal en la que se han bloqueado. En realidad, estas anotaciones no son relevantes en esta semántica extendida y podrían omitirse, sin embargo, es necesario conocerlas para poder relacionar los *heaps* de la semántica para un procesador descrita en la sección anterior con los *heaps* de la semántica natural. De esta forma se obtienen las siguientes definiciones:

DEFINICIÓN 3.2.1. Un *heap de bloqueos* $\Gamma^B \in BHeap$ es un conjunto finito de ligaduras

bloqueadas del tipo $\theta \overset{B(\tau)}{\mapsto} E$.

□

DEFINICIÓN 3.2.2. Sea $\Gamma \in \mathit{Heap}$ un *heap* de la semántica natural. Se definen los siguientes conjuntos:

- *variables que aparecen en* Γ: $\mathit{Var}(\Gamma) = \{x \in \mathit{Var} \mid x \mapsto E \in \Gamma\}$;
- *canales que aparecen en* Γ: $\mathit{Chan}(\Gamma) = \{ch \in \mathcal{C}\mathit{han} \mid ch \mapsto E \in \Gamma\}$;
- *variables y canales que aparecen en* Γ: $\mathit{VCh}(\Gamma) = \mathit{Var}(\Gamma) \cup \mathit{Chan}(\Gamma)$.

Análogamente, sea $\Gamma^B \in \mathit{BHeap}$ un *heap* de bloqueos. Se definen los siguientes conjuntos:

- *variables que aparecen en* Γ^B: $\mathit{Var}(\Gamma^B) = \{x \in \mathit{Var} \mid x \overset{B(\tau)}{\mapsto} E \in \Gamma^B\}$;
- *canales que aparecen en* Γ^B: $\mathit{Chan}(\Gamma^B) = \{ch \in \mathcal{C}\mathit{han} \mid ch \overset{B(\tau)}{\mapsto} E \in \Gamma^B\}$;
- *variables y canales que aparecen en* Γ^B: $\mathit{VCh}(\Gamma^B) = \mathit{Var}(\Gamma^B) \cup \mathit{Chan}(\Gamma^B)$.

□

DEFINICIÓN 3.2.3. Se define un *heap ampliado* como un par $\langle \Gamma, \Gamma^B \rangle \in \mathit{AHeap}$, y se denota por $\overline{\Gamma}$, siendo $\Gamma \in \mathit{Heap}$ y $\Gamma^B \in \mathit{BHeap}$ tales que $\mathit{Var}(\Gamma) \cap \mathit{Var}(\Gamma^B) = \emptyset$. Γ^B representa el conjunto de ligaduras bloqueadas asociado a Γ.

□

DEFINICIÓN 3.2.4. Sea $\overline{\Gamma} = \langle \Gamma, \Gamma^B \rangle \in \mathit{AHeap}$ un *heap* ampliado . Se define los siguientes conjuntos:

- *variables que aparecen en* $\overline{\Gamma}$: $\mathit{Var}(\overline{\Gamma}) = \mathit{Var}(\Gamma) \cup \mathit{Var}(\Gamma^B)$;
- *canales que aparecen en* $\overline{\Gamma}$: $\mathit{Chan}(\overline{\Gamma}) = \mathit{Chan}(\Gamma) \cup \mathit{Chan}(\Gamma^B)$;
- *variables y canales que aparecen en* $\overline{\Gamma}$: $\mathit{VCh}(\overline{\Gamma}) = \mathit{VCh}(\Gamma) \cup \mathit{VCh}(\Gamma^B)$;

□

Los *heaps* ampliados de la forma $\langle \Gamma, \Gamma^B \rangle$ se modifican a lo largo de la derivación según indican las reglas expresadas en las Figuras 3.5 y 3.6. Nótese que ahora las expresiones a evaluar estarán ligadas a alguna variable, y las aserciones serán de la siguiente forma:

$$\overline{\Gamma} : \theta \mapsto E \;\Downarrow\; \overline{\Delta} : \theta \mapsto W,$$

donde $\theta \notin \mathit{VCh}(\overline{\Gamma}) \cup \mathit{VCh}(\overline{\Delta})$.

En las reglas se ha utilizado la notación $\overline{\Gamma} + \{\theta \overset{\alpha}{\mapsto} E\}$ (siendo α del tipo $B(\tau)$ o nada) para indicar que el *heap* ampliado $\overline{\Gamma}$ se extiende con una nueva ligadura. Si se trata de una ligadura bloqueada, ésta se incorporará a Γ^B, en otro caso se incorporará a Γ.

Antes de explicar las reglas se definirá el concepto de *heap saturado*, donde la noción de creación de proceso factible es la misma que la expuesta en la Sección 3.1, es decir, una creación de proceso es factible si no tiene dependencias.

$$\text{Sea } \overline{\Gamma} = \langle \Gamma, \Gamma^B \rangle,\ \theta \notin \mathit{VCh}(\overline{\Gamma})$$
$$\mathsf{sat}(\overline{\Gamma} : \theta \mapsto E) = \langle \Delta, \Delta^B \rangle \models (\Delta^B = \Gamma^B) \wedge \{\text{(i)-(iii)}\} \wedge \forall A.(A \models \{\text{(i)-(iii)}\} \Rightarrow \mathsf{sat}(\overline{\Gamma} : \theta \mapsto E) \subseteq A)$$

(i) $\tau \mapsto x\#y \in \Gamma \wedge \mathsf{d}(x, \Gamma \cup \Gamma^B \cup \{\theta \mapsto E\}) \Rightarrow \tau \mapsto x\#y \in \Delta$

(ii) $\tau \mapsto x\#y \in \Gamma \wedge \neg\mathsf{d}(x, \Gamma \cup \Gamma^B \cup \{\theta \mapsto E\}) \Rightarrow \{\tau \mapsto o, i \mapsto y, o \mapsto \eta(x)\ z, z \mapsto i\} \cup \eta(\mathsf{nh}(x, \overline{\Gamma} \cup \{\theta \mapsto E\})) \subseteq \Delta$
donde o, i, z son frescas y η es un renombramiento fresco

(iii) $\tau \mapsto E \in \Gamma$ con $(E \not\equiv x\#y) \Rightarrow \tau \mapsto E \in \Delta$

Figura 3.4: Función de saturación

Los *heaps* ampliados se saturan aplicando la función sat que se define en la Figura 3.4. Esta función refleja la impaciencia asociada a las creaciones de procesos. Una creación de proceso podrá realizarse siempre y cuando no tenga dependencias ni en el *heap* Γ, ni en el *heap* de bloqueos Γ^B, ni dependa de la variable o canal que se está evaluando en el momento en el que se aplica la saturación. Sólo entonces se puede asegurar que todos los canales o creaciones de procesos de los que dependa habrán sido resueltos. De forma análoga a como se hizo en la Sección 3.1, se puede demostrar que al aplicar la función de saturación a un *heap* ampliado no importa el orden en el que se vayan realizando las creaciones de proceso, pues las unas no interfieren con las otras.

DEFINICIÓN 3.2.5. Se dice que un *heap* ampliado $\overline{\Gamma} \in \mathit{AHeap}$ está *saturado* en $\theta \mapsto E$ si todas las creaciones de procesos factibles se han realizado. Es decir, $\mathsf{sat}(\overline{\Gamma} : \theta \mapsto E) = \overline{\Gamma}$. □

En las aserciones $\overline{\Gamma} : \theta \mapsto E \Downarrow \overline{\Delta} : \theta \mapsto W$ de las Figuras 3.5 y 3.6 se considera que los *heaps* $\overline{\Gamma}$ y $\overline{\Delta}$ están saturados, la conservación de esta propiedad se demuestra por inducción sobre la derivación. En algunas de estas nuevas reglas se hace referencia a las funciones d, nh y pld. Aunque sólamente se han definido para los *heaps* etiquetados (Sección 2.3.4) son trivialmente adaptables a los *heaps* ampliados.

Las nuevas reglas *Lambda* y *Variable* son muy similares a las expuestas en la Sección 2.1, salvo por la introducción de la función de saturación sat y porque cada *heap* va acompañado ahora de su correspondiente conjunto de ligaduras bloqueadas. Debido a que el proceso de normalización de las expresiones extendidas es diferente al indicado en la Sección 2.1 las reglas *Let* y *Aplicación* se han modificado. En el primer caso es necesario incorporar renombramientos de las variables ligadas a la expresión *let* (como ocurre en las reglas de Jauja básico para garantizar que todas las variables en el *heap* son distintas). En el segundo caso la modificación es mayor. En la semántica natural de Launchbury se permiten expresiones $E\ x$, en cambio, en Jauja básico se exige que tanto el argumento como la abstracción funcional en una aplicación sean variables, de forma que sólo se admiten expresiones de la forma $(x\ y)$. El objetivo de esta restricción es evitar que el cómputo operacional se bloquee en el caso de

(Lambda)

$$\mathsf{sat}(\overline{\Gamma} : \theta \mapsto \lambda x.E) : \theta \mapsto \lambda x.E \;\Downarrow\; \mathsf{sat}(\overline{\Gamma} : \theta \mapsto \lambda x.E) : \theta \mapsto \lambda x.E \text{ si } \theta \notin \mathit{VCh}(\overline{\Gamma})$$

(Aplicación)

$$\frac{\overline{\Gamma} + \{\theta \overset{B(x)}{\mapsto} x\ y\} : x \mapsto E \;\Downarrow\; \overline{\Delta} + \{\theta \overset{B(x)}{\mapsto} x\ y\} : x \mapsto \lambda z.E' \qquad \overline{\Delta} + \{x \mapsto \lambda z.E'\} : \theta \mapsto E'[y/z] \;\Downarrow\; \overline{\Theta} : \theta \mapsto W}{\overline{\Gamma} + \{x \mapsto E\} : \theta \mapsto x\ y \;\Downarrow\; \overline{\Theta} : \theta \mapsto W}$$

(Variable)

$$\frac{\overline{\Gamma} + \{\theta \overset{B(x)}{\mapsto} x\} : x \mapsto E \;\Downarrow\; \overline{\Delta} + \{\theta \overset{B(x)}{\mapsto} x\} : x \mapsto W}{\overline{\Gamma} + \{x \mapsto E\} : \theta \mapsto x \;\Downarrow\; \overline{\Delta} + \{x \mapsto W\} : \theta \mapsto \widehat{W}}$$

(Let)

$$\frac{\mathsf{sat}(\overline{\Gamma} + \{y_i \mapsto E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n : \theta \mapsto x[y_j/x_j]_{j=1}^n) : \theta \mapsto x[y_j/x_j]_{j=1}^n \;\Downarrow\; \overline{\Delta} : \theta \mapsto W}{\overline{\Gamma} : \theta \mapsto \mathtt{let}\ \{x_i = E_i\}_{i=1}^n\ \mathtt{in}\ x \;\Downarrow\; \overline{\Delta} : \theta \mapsto W}$$

donde $y_i \in \mathit{Var}$ frescas

Figura 3.5: Semántica natural extendida: núcleo funcional

tener expresiones de aplicación anidada [1]. Por este motivo, en la primera parte de la regla nueva se evalúa la expresión ligada a la variable x y luego se realiza la β-reducción.

Además el conjunto de reglas se ha ampliado para incorporar la creación de procesos y la comunicación de canales. Por un lado la regla *creación de procesos* corresponde a la evaluación de una expresión de creación de proceso que no tiene dependencias. Por otro lado, la regla *demanda de creación de proceso* refleja una situación en la que la creación de proceso a evaluar tiene dependencias. En este caso se procede a evaluar la dependencia dada por la función pld (explicada en la Sección 3.1). Una vez evaluada dicha expresión, la correspondiente ligadura se reincorpora al *heap* y se intenta de nuevo la creación de proceso de partida.

También se han introducido dos reglas para la comunicación mediante canales. La primera de ellas, *comunicación*, refleja el caso más simple en el que se calcula el valor asociado a un canal y el valor obtenido no tiene dependencias, así que el valor se transmite sin ningún problema. La segunda regla, *demanda de comunicación*, expresa el caso en el que, una vez obtenido el valor asociado al canal, éste depende de una comunicación de proceso u otro canal. En este caso, antes de transmitir el valor obtenido, se procede a la evaluación de la expresión de la que depende.

La función de saturación se aplica cuando nuevas ligaduras se añaden al *heap* (regla *Let*)

[1] De hecho, este bloqueo se puede producir en la semántica operacional de GpH dado en [4].

(Creación de proceso)

$$\frac{\overline{\Gamma} + \{i \mapsto y, o \mapsto \eta(x)\ z, z \mapsto i\} + \eta(\mathsf{nh}(x, \Gamma^*)) : \theta \mapsto o \ \Downarrow\ \overline{\Delta} : \theta \mapsto W}{\overline{\Gamma} : \theta \mapsto x\#y \ \Downarrow\ \overline{\Delta} : \theta \mapsto W}$$

si $\neg\mathsf{d}(x, \overline{\Gamma}^*)$, donde $\Gamma^* = \Gamma \cup \Gamma^B \cup \{\theta \mapsto x\#y\}$

(Demanda creación de proceso)

$$\frac{\overline{\Gamma} + \{\theta \overset{B(z)}{\mapsto} x\#y\} : z \mapsto E' \ \Downarrow\ \overline{\Delta} + \{\theta \overset{B(z)}{\mapsto} x\#y\} : z \mapsto W' \qquad \overline{\Delta} + \{z \mapsto W'\} : \theta \mapsto x\#y \ \Downarrow\ \overline{\Theta} : \theta \mapsto W}{\overline{\Gamma} + \{z \mapsto E'\} : \theta \mapsto x\#y \ \Downarrow\ \overline{\Theta} : \theta \mapsto W}$$

si $\mathsf{d}(x, \overline{\Gamma}^*) \ \wedge\ \mathsf{pld}(x, \overline{\Gamma}^*) = z \mapsto E' \ \wedge z \neq \theta$, donde $\overline{\Gamma}^* = \Gamma \cup \Gamma^B \cup \{z \mapsto E', \theta \mapsto x\#y\}$

(Comunicación)

$$\frac{\overline{\Gamma} + \{\theta \overset{B(ch)}{\mapsto} ch\} : ch \mapsto E \ \Downarrow\ \overline{\Delta} + \{\theta \overset{B(ch)}{\mapsto} ch\} : ch \mapsto W}{\overline{\Gamma} + \{ch \mapsto E\} : \theta \mapsto ch \ \Downarrow\ \mathsf{sat}(\overline{\Delta} \cup \eta(\mathsf{nh}(W, \overline{\Delta})) : \theta \mapsto \eta(W)) : \theta \mapsto \eta(W)}$$

si $\neg\mathsf{d}(W, \Delta \cup \Delta^B)$

(Demanda comunicación)

$$\frac{\begin{array}{c}\overline{\Gamma} + \{\theta \overset{B(ch)}{\mapsto} ch\} : ch \mapsto E \ \Downarrow\ \overline{\Delta} + \{z \mapsto E', \theta \overset{B(ch)}{\mapsto} ch\} : ch \mapsto W \\ \overline{\Delta} + \{ch \mapsto W, \theta \overset{B(z)}{\mapsto} ch\} : z \mapsto E' \ \Downarrow\ \overline{\Theta} + \{ch \mapsto W, \theta \overset{B(z)}{\mapsto} ch\} : z \mapsto W' \\ \overline{\Theta} + \{z \mapsto W', ch \mapsto W\} : \theta \mapsto ch \ \Downarrow\ \overline{\Lambda} : \theta \mapsto W''\end{array}}{\overline{\Gamma} + \{ch \mapsto E\} : \theta \mapsto ch \ \Downarrow\ \overline{\Lambda} : \theta \mapsto W''}$$

si $\mathsf{d}(W, \Delta^*) \ \wedge\ \mathsf{pld}(W, \Delta^*) = z \mapsto E'$, donde $\Delta^* = \Delta \cup \Delta^B \cup \{z \mapsto E'\}$;

donde $z \in \mathit{Var}$ frescas, $o, i, \in \mathcal{C}\mathit{han}$ frescos y η es un renombramiento fresco

Figura 3.6: Semántica natural extendida: creación de proceso y comunicación

o alguna variable o canal pasa de estar ligada a un canal a estar ligada a un valor (regla *Comunicación*). Al aplicar la regla *creación de proceso* no se efectúa la saturación porque todas las ligaduras que se introducen en el *heap* ampliado corresponden a variables y canales frescos.

Se puede demostrar fácilmente que las reglas están *bien definidas*; es decir, que en todas las aserciones los *heaps* ampliados están saturados, que sus conjuntos de variables son disjuntos y que la variable ligada a la expresión a evaluar no aparece ligada ni en el *heap* ni en el *heap* de bloqueos que forman el correspondiente *heap* ampliado.

La siguiente proposición establece que en una derivación el *heap* de bloqueos inicial coincide siempre con el final.

PROPOSICIÓN 3.2.6. *Para cualquier aserción derivable*

$$\langle \Gamma, \Gamma^B \rangle : \theta \mapsto E \ \Downarrow \ \langle \Delta, \Delta^B \rangle : \theta \mapsto W$$

se tiene que $\Gamma^B = \Delta^B$.

Demostración
Se demuestra fácilmente por inducción sobre las reglas que se aplican. c.q.d.

3.2.1. Consistencia entre la semántica natural y su extensión

A continuación se demostrará que la extensión dada para la semántica natural es consistente con la semántica expuesta en la Sección 2.1. Esta consistencia se basa en ver que para toda expresión $E \in Exp$ el valor obtenido en la semántica extendida es el mismo que el producido en la semántica original, así como que el valor producido en la semántica original coincide con el obtenido en la semántica extendida. Para poder demostrarlo es necesario especificar cómo pasar de los *heaps* de Launchbury a los *heaps* ampliados y viceversa.

Pasar de los *heaps* ampliados a los *heaps* de Launchbury es sencillo, basta tomar la proyección del *heap* ampliado $\overline{\Gamma}$. Además la expresión a evaluar será aquella que está ligada a la ligadura que se está evaluando.

DEFINICIÓN 3.2.7. Sea $\overline{\Gamma} : x \mapsto E$ un *heap* ampliado en el que se está evaluando la ligadura $x \mapsto E$ y donde $\overline{\Gamma} = \langle \Gamma, \Gamma^B \rangle$. Su *correspondiente heap de Launchbury* y la expresión a evaluar será: $\Gamma : E$.

□

Sin embargo, pasar de un *heap* de Launchbury a un *heap* ampliado no es tan fácil. Para empezar no se dispone de la información de las ligaduras que se han ido bloqueando a lo largo de la derivación, ni tampoco de la variable ligada a la expresión que se está evaluando. Se recuerda que dicha información era necesaria para calcular las dependencias de las creaciones de proceso y de los canales. Si se asegura que no hay expresiones de dicho tipo entonces podrá tomarse como *heap* ampliado aquel formado por el *heap* de Launchbury y considerar que el *heap* de bloqueos es cualquier conjunto de ligaduras bloqueadas cuyas variables sean disjuntas a las del *heap* de partida, la expresión a evaluar se ligará a una variable que no esté en el *heap* ampliado que se acaba de construir.

DEFINICIÓN 3.2.8. Sea $\Gamma : E$ un *heap* de Launchbury en el que se está evaluando la expresión E. Su correspondiente *heap* ampliado con la ligadura a evaluar vendrá dado por:

$$\langle \Gamma, \Gamma^B \rangle : x \mapsto E$$

para cualquier Γ^B tal que $Var(\Gamma) \cap Var(\Gamma^B) = \emptyset$ y x es cualquier variable que verifica que $x \notin Var(\Gamma) \cup Var(\Gamma^B)$.

□

TEOREMA 3.2.9. Sea $E \in Exp$. Si $\langle \Gamma, \Gamma^B \rangle : y \mapsto E \ \Downarrow \ \langle \Delta, \Delta^B \rangle : y \mapsto W$[2] entonces $\Gamma : E \ \Downarrow \ \Delta : W$.

Demostración
Se demostrará por inducción sobre la derivación. Como $x\#y \notin Exp$ se tiene que todos los *heaps* ampliados están saturados.

- LAMBDA:
$$\overline{\Gamma} : y \mapsto \lambda x.E \ \Downarrow \ \overline{\Gamma} : y \mapsto \lambda x.E$$
Es trivial ver que
$$\Gamma : \lambda x.E \ \Downarrow \ \Gamma : \lambda x.E.$$

- APLICACIÓN:
$$\overline{\Gamma} + \{\theta \overset{B(x_1)}{\mapsto} x_1 \ x_2\} : x_1 \mapsto E \ \Downarrow \ \overline{\Delta} + \{\theta \overset{B(x_1)}{\mapsto} x_1 \ x_2\} : x_1 \mapsto \lambda z.E'$$
$$\frac{\overline{\Delta} + \{x_1 \mapsto \lambda z.E'\} : \theta \mapsto E'[x_2/z] \ \Downarrow \ \overline{\Theta} : \theta \mapsto W}{\overline{\Gamma} + \{x_1 \mapsto E\} : y \mapsto x_1 \ x_2 \ \Downarrow \ \overline{\Theta} : y \mapsto W}$$
Sean $\overline{\Gamma} = \langle \Gamma, \Gamma^B \rangle$, $\overline{\Delta} = \langle \Delta, \Delta^B \rangle$ y $\overline{\Theta} = \langle \Theta, \Theta^B \rangle$.
Por hipótesis de inducción se tiene que
(1) $\Gamma : E \ \Downarrow \ \Delta : \lambda z.E'$ y
(2) $(\Delta, x_1 \mapsto \lambda z.E') : E'[x_2/z] \ \Downarrow \ \Theta : W$.
Utilizando la regla *Aplicación* de la semántica natural original se puede derivar
$(\Gamma, x_1 \mapsto E) : x_1 \ x_2 \ \Downarrow \ \Theta : W$.
$$\frac{\frac{(1) \ \Gamma:E \Downarrow \Delta:\lambda z.E'}{(\Gamma,x_1\mapsto E):x_1 \Downarrow (\Delta,x_1\mapsto\lambda z.E'):\lambda t.E'[t/z]} \qquad (2) \ (\Delta, x_1 \mapsto \lambda z.E') : E'[t/z][x_2/t] \ \Downarrow \ \Theta : W}{(\Gamma, x_1 \mapsto E) : x_1 \ x_2 \ \Downarrow \ \Theta : W}$$
donde se ha aplicado que $E'[t/z][x_2/t] = E'[x_2/z]$.

- VARIABLE: Se demuestra con un razonamiento análogo al anterior.

- LET: La demostración es trivial teniendo en cuenta que no es necesario renombrar las variables x_i, pues el proceso de normalización asegura que estas variables no aparecen en el *heap* y por lo tanto no puede haber conflictos.

El resto de reglas no son aplicables, puesto que involucran creaciones de proceso o canales, que no pertenecen al conjunto de expresiones *Exp*. c.q.d.

Antes de demostrar la consistencia de la semántica extendida en la dirección contraria se demostrará una propiedad que se utilizará para la demostración del teorema.

PROPOSICIÓN 3.2.10. Si $\Gamma : E \ \Downarrow \ \Delta : W$ entonces $Var(\Gamma) \subseteq Var(\Delta)$.

[2]Por supuesto, los *heaps* Γ, Γ^B, Δ y Δ^B sólo involucran expresiones $E \in Exp$ y no contienen canales.

Demostración
Se demostrará por inducción sobre la derivación:

- LAMBDA: Trivial que $Var(\Gamma) \subseteq Var(\Gamma)$.
- APLICACIÓN:
$$\frac{\Gamma : x \ \Downarrow \ \Delta : \lambda z.E' \qquad \Delta : E'[y/z] \ \Downarrow \ \Theta : W}{\Gamma : x\ y \ \Downarrow \ \Theta : W}$$
Hay que ver que $Var(\Gamma) \subseteq Var(\Theta)$. Por hipótesis de inducción se tiene que $Var(\Gamma) \subseteq Var(\Delta)$ y $Var(\Delta) \subseteq Var(\Theta)$. Por la transitividad de $\subseteq$ se tiene que $Var(\Gamma) \subseteq Var(\Theta)$.
- VARIABLE:
$$\frac{\Gamma : E \ \Downarrow \ \Delta : W}{(\Gamma, x \mapsto E) : x \ \Downarrow \ (\Delta, x \mapsto W) : \widehat{W}}$$
Hay que ver que $Var((\Gamma, x \mapsto E)) \subseteq Var((\Delta, x \mapsto W))$. Por hipótesis de inducción se tiene que $Var(\Gamma) \subseteq Var(\Delta)$, luego $Var((\Gamma, x \mapsto E)) = Var(\Gamma) \cup \{x\} \overset{HI}{\subseteq} Var(\Delta) \cup \{x\} = Var((\Delta, x \mapsto W))$.
- LET:
$$\frac{(\Gamma, x_1 \mapsto E_1, ..., x_n \mapsto E_n) : E \ \Downarrow \ \Delta : W}{\Gamma : \texttt{let}\ \{x_i = E_i\}_{i=1}^n\ \texttt{in}\ E \ \Downarrow \ \Delta : W}$$
Hay que demostrar que $Var(\Gamma) \subseteq Var(\Delta)$. Por hipótesis de inducción se tiene que $Var((\Gamma, x_1 \mapsto E_1, ..., x_n \mapsto E_n)) \overset{HI}{\subseteq} Var(\Delta)$. Ahora bien, $Var(\Gamma) \subseteq Var(\Gamma) \cup \{x_i\}_{i=1}^n = Var((\Gamma, x_1 \mapsto E_1, ..., x_n \mapsto E_n)) \overset{HI}{\subseteq} Var(\Delta)$.

c.q.d.

TEOREMA 3.2.11.
Sea $E \in Exp$ una expresión. Si $\Gamma : E \ \Downarrow \ \Delta : W$ entonces para todo Γ^B tal que $Var(\Gamma^B) \cap (Var(\Gamma) \cup Var(\Delta)) = \emptyset$ y para todo x tal que $x \notin Var(\Gamma) \cup Var(\Gamma^B) \cup Var(\Delta)$ se tiene que $\langle \Gamma, \Gamma^B \rangle : x \mapsto E \ \Downarrow \ \langle \Delta, \Gamma^B \rangle : x \mapsto W$.

Demostración
Antes de empezar con la demostración, hay que recordar que en *Exp* se admiten aplicaciones del tipo $(E\ x)$ mientras que en *EExp* se exige que sean de la forma $(x\ y)$. Cualquier expresión $(E\ x)$ se puede transformar a la sintaxis restringida mediante el uso de declaraciones locales, así que se supondrá que las expresiones E en el enunciado del teorema ya se han transformado.

Se demostrará por inducción sobre la derivación. Aunque no se indique explícitamente se aplicará la Proposición 3.2.6 en numerosas ocasiones.

- LAMBDA:
$$\Gamma : \lambda x.E \ \Downarrow \ \Gamma : \lambda x.E$$
Al no haber creaciones de proceso, todo *heap* ampliado está ya saturado, así que es trivial ver que
$$\langle \Gamma, \Gamma^B \rangle : y \mapsto \lambda x.E \ \Downarrow \ \langle \Gamma, \Gamma^B \rangle : y \mapsto \lambda x.E,$$

para cualesquiera Γ^B e y que verifiquen las hipótesis del teorema.

- APLICACIÓN:

$$\frac{\Gamma : x \ \Downarrow \ \Delta : \lambda z.E' \qquad \Delta : E'[y/z] \ \Downarrow \ \Theta : W}{\Gamma : x\ y \ \Downarrow \ \Theta : W}$$

Por hipótesis de inducción se tiene que:
(1) $\langle \Gamma, \Gamma^B \rangle : x_1 \mapsto x \ \Downarrow \ \langle \Delta, \Gamma^B \rangle : x_1 \mapsto \lambda z.E'$ y
(2) $\langle \Delta, \Delta^B \rangle : x_2 \mapsto E'[y/z] \ \Downarrow \ \langle \Theta, \Delta^B \rangle : x_2 \mapsto W$,
para cualesquiera Γ^B, Δ^B, x_1 y x_2 que verifiquen las hipótesis del teorema.

Como $\Gamma : x \ \Downarrow \ \Delta : \lambda z.E'$ entonces existe $E \in Exp$ tal que $\Gamma = (\Gamma^*, x \mapsto E)$ y $\Gamma^* : E \ \Downarrow \ \Delta^* : \lambda z.E'$ con $\Delta = (\Delta^*, x \mapsto \lambda z.E')$; si no fuera así x no obtendría ningún valor. Por lo tanto también se tiene como hipótesis de inducción:
(3) $\langle \Gamma^*, \Gamma^{*B} \rangle : x_3 \mapsto E \ \Downarrow \ \langle \Delta^*, \Gamma^{*B} \rangle : x_3 \mapsto \lambda z.E'$.

Hay que ver que $\langle \Gamma^*, \Gamma^B \rangle + \{x \mapsto E\} : t \mapsto x\ y \ \Downarrow \ \langle \Theta, \Gamma^B \rangle : t \mapsto W$ para cualesquiera Γ^B y t tales que $Var(\Gamma^B) \cap (Var(\Gamma^*) \cup \{x\} \cup Var(\Theta)) = \emptyset$ y $t \notin Var(\Gamma^B) \cup Var(\Gamma^*) \cup \{x\} \cup Var(\Theta)$. Utilizando la regla *Aplicación* de la figura 3.5:

$$\frac{\langle \Gamma^*, \Gamma^B \rangle + \{t \overset{B(x)}{\mapsto} x\ y\} : x \mapsto E \overset{(3)}{\Downarrow} \langle \Delta^*, \Gamma^B \rangle + \{t \overset{B(x)}{\mapsto} x\ y\} : x \mapsto \lambda z.E' \qquad \langle \Delta^*, \Gamma^B \rangle + \{x \mapsto \lambda z.E'\} : t \mapsto E'[y/z] \overset{(2)}{\Downarrow} \langle \Theta, \Gamma^B \rangle : t \mapsto W}{\langle \Gamma^*, \Gamma^B \rangle + \{x \mapsto E\} : t \mapsto x\ y \ \Downarrow \ \langle \Theta, \Gamma^B \rangle : t \mapsto W}$$

Para poder utilizar las hipótesis de inducción (3) y (2) hay que asegurar que los conjuntos de bloqueo y las variables elegidas en la instanciación de la regla de *Aplicación* verifican las hipótesis del teorema. En el caso de (3), si se tiene que $Var(\Gamma^B) \cap (Var(\Gamma^*) \cup \{x\} \cup Var(\Theta)) = \emptyset$ y $t \notin Var(\Gamma^B) \cup Var(\Gamma^*) \cup Var(\Theta)$ entonces también se cumple que $(Var(\Gamma^B) \cup \{t\}) \cap (Var(\Gamma^*) \cup Var(\Delta^*))$ y $x \notin Var(\Gamma^B) \cup Var(\Gamma^*) \cup Var(\Delta^*)$ ya que por la Proposición 3.2.10 $Var(\Delta^*) \subseteq Var(\Theta)$. Además, con este resultado se comprueba fácilmente que $Var(\Gamma^B) \cap (Var(\Delta^*) \cup \{x\} \cup Var(\Theta)) = \emptyset$ y que $t \notin Var(\Gamma^B) \cup Var(\Delta^*) \cup Var(\Theta)$, que justifica el uso de (2).

- VARIABLE:

$$\frac{\Gamma : E \ \Downarrow \ \Delta : W}{(\Gamma, x \mapsto E) : x \ \Downarrow \ (\Delta, x \mapsto W) : \widehat{W}}$$

Por hipótesis de inducción se tiene que: $\langle \Gamma, \Gamma^B \rangle : y \mapsto E \ \Downarrow \ \langle \Delta, \Gamma^B \rangle : y \mapsto W$, para todo Γ^B e y que verifiquen las hipótesis.
Hay que demostrar que

$$\langle \Gamma, \Gamma^B \rangle + \{x \mapsto E\} : y \mapsto x \ \Downarrow \ \langle \Delta, \Gamma^B \rangle + \{x \mapsto W\} : y \mapsto \widehat{W}$$

siendo Γ^B e y como indican las hipótesis, es decir, $Var(\Gamma^B) \cap (Var(\Gamma) \cup \{x\} \cup Var(\Delta) = \emptyset$ e $y \notin Var(\Gamma^B) \cup Var(\Gamma) \cup \{x\} \cup Var(\Delta)$. Además $x \notin Var(\Gamma) \cup Var(\Delta)$, ya que si no fuera

así habría conflicto de variables y no podría aplicarse la regla. Por lo tanto, se tiene que $(Var(\Gamma^B) \cup \{y\}) \cap (Var(\Gamma) \cup Var(\Delta)) = \emptyset$ y que $x \notin Var(\Gamma^B) \cup \{y\} \cup Var(\Gamma) \cup Var(\Delta)$, lo que permite instanciar la hipótesis de inducción de forma que se puede aplicar la regla *Variable* de la semántica natural extendida:

$$\frac{\langle \Gamma, \Gamma^B \rangle + y \overset{B(x)}{\mapsto} E : x \mapsto E \overset{HI}{\Downarrow} \langle \Delta, \Gamma^B \rangle + \{y \overset{B(x)}{\mapsto} x\} : x \mapsto W}{\langle \Gamma, \Gamma^B \rangle + \{x \mapsto E\} : y \mapsto x \Downarrow \langle \Delta, \Gamma^B \rangle + \{x \mapsto W\} : y \mapsto \widehat{W}}$$

- LET:
  En la regla *Let* expuesta en la Figura 2.3 debido a la normalización que realiza Launchbury (2.2) no se realiza ningún renombramiento de las variables locales. Sin embargo se están tomando las expresiones expuestas en la Figura 2.5, y el renombramiento de las variables locales es necesario, por lo que la regla *Let* queda ligeramente modificada:

$$\frac{(\Gamma, \{y_i \mapsto E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n) : x[y_j/x_j]_{j=1}^n \Downarrow \Delta : W}{\Gamma : \texttt{let } \{x_i = E_i\}_{i=1}^n \texttt{ in } x \Downarrow \Delta : W}$$

  Por hipótesis de inducción se sabe que:

$$\langle \Gamma, \Gamma^B \rangle + \{y_i \mapsto E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n : y \mapsto x[y_j/x_j]_{j=1}^n \Downarrow \langle \Delta, \Gamma^B \rangle : W,$$

  donde Γ^B e y cumplen las hipótesis del teorema.
  Hay que demostrar: $\langle \Gamma, \Gamma^B \rangle : y \mapsto \texttt{let } \{x_i = E_i\}_{i=1}^n \texttt{ in } x \Downarrow \langle \Delta, \Gamma^B \rangle : W$, donde Γ^B e y verifican las hipótesis. Aplicando la regla *Let* de la semántica extendida se tiene:

$$\frac{\langle \Gamma, \Gamma^B \rangle + \{y_i \mapsto E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n : y \mapsto x[y_j/x_j]_{j=1}^n \overset{HI}{\Downarrow} \langle \Delta, \Gamma^B \rangle : W}{\langle \Gamma, \Gamma^B \rangle : y \mapsto \texttt{let } \{x_i = E_i\}_{i=1}^n \texttt{ in } x \Downarrow \langle \Delta, \Gamma^B \rangle : W}$$

  No es necesario saturar el *heap* en las premisas ya que no hay creaciones de proceso. Con un razonamiento similar al de los casos anteriores y teniendo en cuenta que las variables y_i son frescas, es decir, $y_i \neq y$ y $y_i \notin Var(\Gamma^B)$ se puede aplicar la hipótesis de inducción obteniendo el resultado buscado.

  c.q.d.

Capítulo 4

Resultados

Siguiendo los pasos de los autores de [4] que ampliaron la demostración de Launchbury en [21] adecuándola a la semántica operacional que definieron para GpH, en este capítulo se demuestran la corrección, adecuación y determinación de la semántica de Jauja básico. Antes de empezar a demostrar estas propiedades se probará que la semántica operacional de Jauja básico para un único procesador expuesta en la Sección 3.1 es equivalente a la semántica natural extendida expuesta en la Sección 3.2.

4.1. Equivalencia entre semánticas

En esta sección se demostrará que, para cualquier expresión extendida $E \in EExp$, el valor obtenido con la semántica operacional de Jauja básico para un único procesador (Sección 3.1) es el mismo que el obtenido mediante las reglas de la semántica natural extendida (Sección 3.2).

Para poder estudiar la equivalencia entre las semánticas es necesario poder relacionar los *heaps* etiquetados de la semántica de Jauja básico, formados por ligaduras etiquetadas (definición 2.3.2), con los *heaps* ampliados de la semántica natural (definición 3.2.3).

Definición 4.1.1. Sea $H \in EHeap$ un *heap* etiquetado alcanzable. Se define el *heap ampliado asociado a* H, denotado por $\overline{\Gamma}_H$ como el *heap* ampliado $\langle \Gamma, \Gamma^B \rangle$ dado por:

- $\Gamma = \{\theta \mapsto E \mid \theta \overset{I}{\mapsto} E \in \mathcal{LI}(H)\}$,
- $\Gamma^B = \mathcal{LB}(H)$.

Análogamente, sea $\overline{\Gamma} = \langle \Gamma, \Gamma^B \rangle \in AHeap$ un *heap* ampliado. Se define el *heap etiquetado asociado a* $\overline{\Gamma}$ y se denota por $H_{\overline{\Gamma}}$ como el *heap* etiquetado tal que:

- $\mathcal{LI}(H_{\overline{\Gamma}}) = \{\theta \overset{I}{\mapsto} E \mid \theta \mapsto E \in \Gamma\}$,
- $\mathcal{LB}(H_{\overline{\Gamma}}) = \Gamma^B$.

□

A continuación se enuncian unas propiedades sencillas que resultarán útiles para las demostraciones del resto de esta sección:

PROPOSICIÓN 4.1.2. *Sean* $E \in EExp \cup \mathcal{C}han$, $\overline{\Gamma} \in AHeap$, $\theta, \tau \in Var \cup \mathcal{C}han$ *tal que* $\theta \notin VCh(\overline{\Gamma})$ *y* $H_{\overline{\Gamma}} \in EHeap$ *el heap etiquetado asociado a* $\overline{\Gamma}$. *Si* $\overline{\Gamma}' = \overline{\Gamma} + \{\theta \overset{\alpha}{\mapsto} E\}$ *entonces el heap etiquetado asociado a* $\overline{\Gamma}'$ *es* $H_{\overline{\Gamma}'} = H_{\overline{\Gamma}} + \{\theta \overset{\alpha'}{\mapsto} E\}$ *siendo* $\begin{cases} \alpha' = I & si\ \alpha =' \ ' \\ \alpha' = \alpha & si\ \alpha = B(\tau) \end{cases}$.

Demostración
Trivial por la definición de *heap* etiquetado asociado.

c.q.d.

PROPOSICIÓN 4.1.3. *Sean* $E \in EExp$, $H \in EHeap$, $\theta, \tau \in Var \cup \mathcal{C}han$ *tal que* $\theta \notin VCh(H)$ *y* $\overline{\Gamma}_H \in AHeap$ *el heap ampliado asociado a* H. *Si* $H' = H + \{\theta \overset{\alpha}{\mapsto} E\}$ *entonces el heap ampliado asociado a* H' *es* $\overline{\Gamma}_{H'} = \overline{\Gamma}_H + \{\theta \overset{\alpha'}{\mapsto} E\}$ *donde* $\begin{cases} \alpha' =' \ ' & si\ \alpha = I \\ \alpha' = \alpha & si\ \alpha = B(\tau) \end{cases}$.

Demostración
Trivial por la definición de *heap* ampliado asociado.

c.q.d.

Antes de enunciar los dos teoremas que establecen la equivalencia se enunciarán algunos lemas que serán necesarios para su demostración.

LEMA 4.1.4. *Sean* $\overline{\Gamma} \in AHeap$, $E \in EExp$ *y* $\theta \notin VCh(\overline{\Gamma})$ *tal que* $\overline{\Gamma}$ *está saturado en* $\theta \mapsto E$, *entonces si* $H_{\overline{\Gamma}}$ *es el heap etiquetado asociado a* $\overline{\Gamma}$ *se tiene que* $H_{\overline{\Gamma}} + \{\theta \overset{A}{\mapsto} E\} \overset{CPI}{\not\longrightarrow}_1$.

Demostración
Se demostrará por reducción al absurdo:
Supongamos que se puede realizar un paso $\overset{CPI}{\longrightarrow}_1$, eso significa que existe una creación de proceso factible $\tau \overset{I}{\mapsto} x\#y \in H_{\overline{\Gamma}}$, es decir, $\neg\mathsf{d}(x, H_{\overline{\Gamma}} + \{\theta \overset{A}{\mapsto} E\})$. Entonces, por la definición de $H_{\overline{\Gamma}}$ se tiene que $\tau \mapsto x\#y \in \overline{\Gamma}$ y $\neg\mathsf{d}(x, \overline{\Gamma} \cup \{\theta \mapsto E\})$ y, por tanto, $\overline{\Gamma}$ no estaría todavía saturado.

c.q.d.

LEMA 4.1.5. *Sean* $\overline{\Gamma} \in AHeap$ *un heap ampliado,* $E \in EExp$ *y* $\theta \notin VCh(\overline{\Gamma})$. *Entonces se cumple que* $H_{\overline{\Gamma}} + \{\theta \overset{A}{\mapsto} E\} \overset{CPI}{\Longrightarrow}_1 H_{\mathsf{sat}(\overline{\Gamma}:\theta\mapsto E)} + \{\theta \overset{A}{\mapsto} E\}$.

Demostración
Sean $\overline{\Gamma} = \langle \Gamma, \Gamma^B \rangle$ y $\mathsf{sat}(\overline{\Gamma} : \theta \mapsto E) = \langle \Delta, \Gamma^B \rangle$.

Por el Corolario 3.1.5 se sabe que no importa el orden en el que se realicen las creaciones de proceso. Sea $\tau \mapsto x\#y \in \Gamma$ tal que $\neg\mathsf{d}(x, \overline{\Gamma} \cup \{\theta \mapsto E\})$, entonces por la definición de la función de saturación se tiene que

$$\{\tau \mapsto o, i \mapsto y, o \mapsto \eta(x)\ z, z \mapsto i\} \cup \eta(\mathsf{nh}(x, \overline{\Gamma} + \{\theta \mapsto E\})) \subseteq \Delta$$

Además, por la definición de *heap* etiquetado asociado se sabe que $\tau \overset{I}{\mapsto} x\#y \in H_{\overline{\Gamma}}$ y $\neg\mathsf{d}(x, H_{\overline{\Gamma}} + \{\theta \overset{A}{\mapsto} E\})$, eligiendo las mismas variables y canales frescos y el mismo renombramiento fresco η que se escogen al hacer la saturación, se tiene que

$$H_{\overline{\Gamma}} + \{\theta \overset{A}{\mapsto} E\} \overset{CPI}{\longrightarrow}_1 H_{\overline{\Gamma}'} + \{\theta \overset{A}{\mapsto} E'\} + \{\tau \overset{I}{\mapsto} o, i \overset{I}{\mapsto} y, o \overset{I}{\mapsto} \eta(x)\ z, z \overset{I}{\mapsto} i\} + \eta(\mathsf{nh}(x, H_{\overline{\Gamma}} + \{\theta \overset{A}{\mapsto} E\}))$$

De esta forma se demuestra que para cada "paso de saturación"de $\overline{\Gamma}$ en $\theta \mapsto E$ se puede realizar una transición $\overset{CPI}{\longrightarrow}_1$ de forma que las modificaciones en el *heap* ampliado $\overline{\Gamma}$ tengan su reflejo exacto en las modificaciones de $H_{\overline{\Gamma}} + \{\theta \mapsto E\}$. c.q.d.

Lema 4.1.6. *Sean $H, H' \in EHeap$ heaps etiquetados, $E \in EExp$ y $\theta \notin VCh(H) \cup VCh(H')$ tal que $H + \{\theta \overset{A}{\mapsto} E\} \overset{CPI}{\Longrightarrow}_1 H' + \{\theta \overset{A}{\mapsto} E'\} \overset{CPI}{\not\longrightarrow}_1$ Entonces, eligiendo de forma adecuada los renombramientos, variables y canales frescos durante la saturación, se tiene que $\overline{\Gamma}_{H'} = \mathsf{sat}(\overline{\Gamma}_H : \theta \mapsto E)$.*

Demostración
El razonamiento es análogo al anterior; para cada transición $\overset{CPI}{\longrightarrow}_1$ se demuestra que se puede realizar un paso de saturación de $\overline{\Gamma}_H$ en $\theta \mapsto E$. Si no se pueden realizar más transiciones $\overset{CPI}{\longrightarrow}_1$ es que se ha logrado la saturar $\overline{\Gamma}_H$. c.q.d.

Corolario 4.1.7. *Si $H + \{\theta \overset{A}{\mapsto} E\}$ es un heap alcanzable, entonces $\overline{\Gamma}_H$ está saturado en $\theta \mapsto E$.*

Demostración
Por el Lema 3.1.7 se sabe que si $H + \{\theta \overset{A}{\mapsto} E\}$ es alcanzable entonces todas las creaciones de proceso factibles se han realizado, es decir, $H + \{\theta \overset{A}{\mapsto} E\} \overset{CPI}{\Longrightarrow}_1 H + \{\theta \overset{A}{\mapsto} E\}$ y por el Lema 4.1.6 se tiene que $\overline{\Gamma}_H = \mathsf{sat}(\overline{\Gamma}_H : \{\theta \mapsto E\})$, luego $\overline{\Gamma}_H$ está saturado. c.q.d.

Se recuerda que cada transición $\Longrightarrow_1$ en la semántica de Jauja básico para un procesador se descompone en $\longrightarrow_1; \overset{CPI}{\Longrightarrow}_1$. A lo largo de la demostración se utilizará la siguiente notación:

- $\Longrightarrow_1$ para indicar que se da un solo paso. Junto al símbolo aparecerá el nombre de la regla de tipo $\longrightarrow_1$ que se aplica en la primera parte de la transición.
- $\Longrightarrow_1^*$ para indicar que se dan varios pasos (o ninguno).
- Cuando se considere oportuno se descompondrá la transición aplicando primero una regla de tipo $\longrightarrow_1$, seguida de $\overset{CPI}{\Longrightarrow}_1$.

Si se utiliza alguno de los lemas expuestos anteriormente se indicará explícitamente junto al símbolo de regla o transición que se está aplicando en ese paso.

TEOREMA 4.1.8. Equivalencia 1.
Sean $E \in EExp \cup \mathcal{C}han$, $\overline{\Gamma}$, $\overline{\Delta} \in AHeap$ *y* $\theta \notin VCh(\overline{\Gamma})$. *Si* $\overline{\Gamma} : \theta \mapsto E \Downarrow \overline{\Delta} : \theta \mapsto W$, *entonces existe un cómputo tal que* $H_{\overline{\Gamma}} + \{\theta \overset{A}{\mapsto} E\} \Longrightarrow_1^* H_{\overline{\Delta}} + \{\theta \overset{A}{\mapsto} W\}$.

Demostración

Por inducción sobre las derivaciones teniendo en cuenta que, cada vez que se introduzcan variables, canales y renombramientos frescos, se tomarán los mismos en ambas semánticas.

LAMBDA:

$$\mathsf{sat}(\overline{\Gamma} : \theta \mapsto \lambda x.E) \Downarrow \mathsf{sat}(\overline{\Gamma} : \theta \mapsto \lambda x.E)$$

Sea $\Delta = \mathsf{sat}(\overline{\Gamma} : \theta \mapsto \lambda x.E)$. La demostración es trivial por la reflexividad de $\Longrightarrow_1^*$:

$$H_{\overline{\Delta}} + \{\theta \overset{A}{\mapsto} \lambda x.E\} \Longrightarrow_1^* H_{\overline{\Delta}} + \{\theta \overset{A}{\mapsto} \lambda x.E\}$$

.

APLICACIÓN:

$$\frac{\overline{\Gamma} + \{\theta \overset{B(x)}{\mapsto} x\ y\} : x \mapsto E \Downarrow \overline{\Delta} + \{\theta \overset{B(x)}{\mapsto} x\ y\} : x \mapsto \lambda z.E' \qquad \overline{\Delta} + \{x \mapsto \lambda z.E'\} : \theta \mapsto E'[y/z] \Downarrow \overline{\Theta} : \theta \mapsto W}{\overline{\Gamma} + \{x \mapsto E\} : \theta \mapsto x\ y \Downarrow \overline{\Theta} : \theta \mapsto W}$$

Por hipótesis de inducción:
(1) $H_{\overline{\Gamma}} + \{\theta \overset{B(x)}{\mapsto} x\ y, x \overset{A}{\mapsto} E\} \Longrightarrow_1^* H_{\overline{\Delta}} + \{\theta \overset{B(x)}{\mapsto} x\ y, x \overset{A}{\mapsto} \lambda z.E'\}$ y
(2) $H_{\overline{\Delta}} + \{x \overset{I}{\mapsto} \lambda z.E', \theta \overset{A}{\mapsto} E'[y/z]\} \Longrightarrow_1^* H_{\overline{\Theta}} + \{\theta \overset{A}{\mapsto} W\}$

Hay que demostrar que $H_{\overline{\Gamma}} + \{x \overset{I}{\mapsto} E, \theta \overset{A}{\mapsto} x\ y\} \Longrightarrow_1^* H_{\overline{\Theta}} + \{\theta \overset{A}{\mapsto} W\}$

- CASO 1: $E \equiv \lambda z.E'$. En este caso se tiene que $\overline{\Delta} = \overline{\Gamma}$ ya que al estar E evaluado se aplica la regla *Lambda*.

$$\begin{aligned} H_{\overline{\Gamma}} + \{x \overset{I}{\mapsto} \lambda z.E', \theta \overset{A}{\mapsto} x\ y\} &= H_{\overline{\Delta}} + \{x \overset{I}{\mapsto} \lambda z.E', \theta \overset{A}{\mapsto} x\ y\} \\ \Longrightarrow_1 \begin{cases} \beta\text{-red.} \longrightarrow_1 & H_{\overline{\Delta}} + \{x \overset{I}{\mapsto} \lambda z.E', \theta \overset{A}{\mapsto} E'[y/z]\} \\ \text{lema 4.1.5, } \overline{\Delta} \text{ saturado} \overset{CPI}{\Longrightarrow}_1 & H_{\overline{\Delta}} + \{x \overset{I}{\mapsto} \lambda z.E', \theta \overset{A}{\mapsto} E'[y/z]\} \\ \text{HI (2)} \Longrightarrow_1^* & H_{\overline{\Theta}} + \{\theta \overset{A}{\mapsto} W\} \end{cases} \end{aligned}$$

- CASO 2: $E \notin Val$

$$\begin{aligned} & H_{\overline{\Gamma}} + \{x \overset{I}{\mapsto} E, \theta \overset{A}{\mapsto} x\ y\} \\ \Longrightarrow_1 \begin{cases} \text{dem. ap.} \longrightarrow_1 & H_{\overline{\Gamma}} + \{\theta \overset{B(x)}{\mapsto} x\ y, x \overset{A}{\mapsto} E\} \\ \text{lema 4.1.5, } \overline{\Gamma} \text{ saturado} \overset{CPI}{\Longrightarrow}_1 & H_{\overline{\Gamma}} + \{\theta \overset{B(x)}{\mapsto} x\ y, x \overset{A}{\mapsto} E\} \\ \text{HI (1)} \Longrightarrow_1^* & H_{\overline{\Delta}} + \{\theta \overset{B(x)}{\mapsto} x\ y, x \overset{A}{\mapsto} \lambda z.E'\} \\ \text{desb-desact} \Longrightarrow_1 & H_{\overline{\Delta}} + \{x \overset{I}{\mapsto} \lambda z.E' + \theta \overset{A}{\mapsto} x\ y\} \\ \beta\text{-red.} \Longrightarrow_1 & H_{\overline{\Delta}} + \{x \overset{I}{\mapsto} \lambda z.E' + \theta \overset{A}{\mapsto} E'[y/z]\} \\ \text{HI (2)} \Longrightarrow_1^* & H_{\overline{\Theta}} : \theta \overset{A}{\mapsto} W \end{cases} \end{aligned}$$

VARIABLE:

$$\frac{\overline{\Gamma} + \{\theta \overset{B(x)}{\mapsto} x\} : x \mapsto E \ \Downarrow \ \overline{\Delta} + \{\theta \overset{B(x)}{\mapsto} x\} : x \mapsto W}{\overline{\Gamma} + \{x \mapsto E\} : \theta \mapsto x \ \Downarrow \ \overline{\Delta} + \{x \mapsto W\} : \theta \mapsto \widehat{W}}$$

Por hipótesis de inducción $H_{\overline{\Gamma}} + \{\theta \overset{B(x)}{\mapsto} x, x \overset{A}{\mapsto} E\} \implies_1^* H_{\overline{\Delta}} + \{\theta \overset{B(x)}{\mapsto} x, x \overset{A}{\mapsto} W\}$.
Hay que demostrar que $H_{\overline{\Gamma}} + \{x \overset{I}{\mapsto} E, \theta \overset{A}{\mapsto} x\} \implies_1^* H_{\overline{\Delta}} + \{x \overset{I}{\mapsto} W, \theta \overset{A}{\mapsto} \widehat{W}\}$

$$\begin{aligned} & H_{\overline{\Gamma}} + \{x \overset{I}{\mapsto} E, \theta \overset{A}{\mapsto} x\} \\ \text{demanda} \implies_1 \quad & H_{\overline{\Gamma}} + \{\theta \overset{B(x)}{\mapsto} x, x \overset{A}{\mapsto} E\} \\ \text{HI} \implies_1^* \quad & H_{\overline{\Delta}} + \{\theta \overset{B(x)}{\mapsto} x, x \overset{A}{\mapsto} W\} \\ \text{desb-desact} \implies_1 \quad & H_{\overline{\Delta}} + \{x \overset{I}{\mapsto} W, \theta \overset{A}{\mapsto} x\} \\ \text{valor} \longrightarrow_1 \quad & H_{\overline{\Delta}} + \{x \overset{I}{\mapsto} W, \theta \overset{A}{\mapsto} W\} \end{aligned}$$

LET:

$$\frac{\mathsf{sat}(\overline{\Gamma} + \{y_i \mapsto E_1[y_j/x_j]_{j=1}^n\}_{i=1}^n : \theta \mapsto x[y_j/x_j]_{j=1}^n) : \theta \mapsto x[y_j/x_j]_{j=1}^n \ \Downarrow \ \overline{\Delta} : \theta \mapsto W}{\overline{\Gamma} : \theta \mapsto \mathtt{let}\ \{x_i = E_i\}_{i=1}^n\ \mathtt{in}\ x \ \Downarrow \ \overline{\Delta} : \theta \mapsto W}$$

Sea $\overline{\Gamma}_1 = \overline{\Gamma} + \{y_i \mapsto E_1[y_j/x_j]_{j=1}^n\}_{i=1}^n$.
Por hipótesis de inducción: $H_{\mathsf{sat}(\overline{\Gamma}_1 : \theta \mapsto x[y_j/x_j]_{j=1}^n)} + \{\theta \overset{A}{\mapsto} x[y_j/x_j]_{j=1}^n\} \implies_1^* H_{\overline{\Delta}} : \theta \overset{A}{\mapsto} W$.

Hay que demostrar que $H_{\overline{\Gamma}} + \{\theta \overset{A}{\mapsto} \mathtt{let}\ \{x_i = E_i\}_{i=1}^n\ \mathtt{in}\ x\} \implies_1^* \overline{\Delta} : \theta \overset{A}{\mapsto} W$.

$$H_{\overline{\Gamma}} : \theta \overset{A}{\mapsto} \mathtt{let}\ \{x_i = E_i\}_{i=1}^n\ \mathtt{in}\ x$$
$$\implies_1 \begin{cases} \text{let} \longrightarrow_1 & H_{\overline{\Gamma}} + \{y_i \overset{I}{\mapsto} E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n + \{\theta \overset{A}{\mapsto} x[y_j/x_j]_{j=1}^n\} \\ \text{lema 4.1.5} \overset{CPI}{\implies}_1 & H_{\mathsf{sat}(\overline{\Gamma}_1 : \theta \mapsto x[y_j/x_j]_{j=1}^n)} + \{\theta \overset{A}{\mapsto} x[y_j/x_j]_{j=1}^n\} \\ \text{HI} \implies_1^* & H_{\overline{\Delta}} : \theta \overset{A}{\mapsto} W \end{cases}$$

CREACIÓN DE PROCESO:

$$\frac{\overline{\Gamma} + \{i \mapsto y, o \mapsto \eta(x)\ z, z \mapsto i\} + \eta(\mathsf{nh}(x, \Gamma^*)) : \theta \mapsto o \ \Downarrow \ \overline{\Delta} : \theta \mapsto W}{\overline{\Gamma} : \theta \mapsto x\#y \ \Downarrow \ \overline{\Delta} : \theta \mapsto W}$$

si $\neg\mathsf{d}(x, \overline{\Gamma}^*)$, donde $\Gamma^* = \Gamma \cup \Gamma^B \cup \{\theta \mapsto x\#y\}$

Por hipótesis de inducción
$H_{\overline{\Gamma}} + \{i \mapsto y, o \mapsto \eta(x)\ z, z \mapsto i\} + \eta(\mathsf{nh}(x, \Gamma^*)) + \{\theta \overset{A}{\mapsto} o\} \implies_1^* H_{\overline{\Delta}} + \{\theta \overset{A}{\mapsto} W\}$.
Además como $\neg\mathsf{d}(x, \overline{\Gamma}^*)$ entonces la creación de proceso ligada a θ es factible.
Al realizar la creación de proceso se escogerán las mismas variables y canales frescos y el mismo renombramiento η.
Hay que demostrar que $H_{\overline{\Gamma}} + \{\theta \overset{A}{\mapsto} x\#y\} \implies_1^* H_{\overline{\Delta}} + \{\theta \overset{A}{\mapsto} W\}$.

$$
\begin{array}{l}
H_{\overline{\Gamma}} : \theta \overset{A}{\mapsto} x\#y \\
\quad \Longrightarrow_1 \left\{ \begin{array}{rl}
\text{creac. proc.} \longrightarrow_1 & H_{\overline{\Gamma}} + \{i \overset{I}{\mapsto} y, o \overset{I}{\mapsto} \eta(x)\ z, z \overset{I}{\mapsto} i\} + \eta(\mathsf{nh}(x, \Gamma^*)) + \{\theta \overset{A}{\mapsto} o\} \\
\text{lema. 4.1.5} \overset{CPI}{\Longrightarrow}_1 & H_{\overline{\Gamma}} + \{i \overset{I}{\mapsto} y, o \overset{I}{\mapsto} \eta(x)\ z, z \overset{I}{\mapsto} i\} + \eta(\mathsf{nh}(x, \Gamma^*)) + \{\theta \overset{A}{\mapsto} o\} \\
\text{HI} \Longrightarrow_1^* & H_{\overline{\Delta}} + \{\theta \overset{A}{\mapsto} W\}
\end{array} \right.
\end{array}
$$

<u>DEMANDA DE CREACIÓN DE PROCESO</u>:

$$
\frac{\overline{\Gamma} + \{\theta \overset{B(z)}{\mapsto} x\#y\} : z \mapsto E' \ \Downarrow\ \overline{\Delta} + \{\theta \overset{B(z)}{\mapsto} x\#y\} : z \mapsto W' \qquad \overline{\Delta} + \{z \mapsto W'\} : \theta \mapsto x\#y \ \Downarrow\ \overline{\Theta} : \theta \mapsto W}{\overline{\Gamma} + \{z \mapsto E'\} : \theta \mapsto x\#y \ \Downarrow\ \overline{\Theta} : \theta \mapsto W}
$$

si $\mathsf{d}(x, \overline{\Gamma}^*) \ \wedge\ \mathsf{pld}(x, \overline{\Gamma}^*) = z \mapsto E' \ \wedge z \neq \theta$, donde $\overline{\Gamma}^* = \Gamma \cup \Gamma^B \cup \{z \mapsto E', \theta \mapsto x\#y\}$

Por hipótesis de inducción,
(1) $H_{\overline{\Gamma}} + \{\theta \overset{B(z)}{\mapsto} x\#y, z \overset{A}{\mapsto} E'\} \implies_1^* H_{\overline{\Delta}} + \{\theta \overset{B(z)}{\mapsto} x\#y, z \overset{A}{\mapsto} W'\}$,
(2) $H_{\overline{\Delta}} + \{z \overset{I}{\mapsto} W', \theta \overset{A}{\mapsto} x\#y\} \implies_1^* H_{\overline{\Theta}} : \theta \overset{A}{\mapsto} W$.
Además se tiene que $\mathsf{d}(x, H_{\overline{\Gamma}} + \{\theta \overset{B(z)}{\mapsto} x\#y, z \overset{A}{\mapsto} E'\})$ y $\mathsf{pld}(x, H_{\overline{\Gamma}} + \{\theta \overset{B(z)}{\mapsto} x\#y, z \overset{A}{\mapsto} E'\}) = z \overset{I}{\mapsto} E'$

Hay que demostrar, $H_{\overline{\Gamma}} + \{z \overset{I}{\mapsto} E', \theta \overset{A}{\mapsto} x\#y\} \implies_1^* H_{\overline{\Theta}} + \{\theta \overset{A}{\mapsto} W\}$.

$$
\begin{array}{l}
H_{\overline{\Gamma}} + \{z \overset{I}{\mapsto} E', \theta \overset{A}{\mapsto} x\#y\} \\
\qquad \begin{array}{rl}
\text{dem. cp.} \Longrightarrow_1 & H_{\overline{\Gamma}} + \{\theta \overset{B(z)}{\mapsto} x\#y, z \overset{A}{\mapsto} E'\} \\
\text{HI (1)} \Longrightarrow_1^* & H_{\overline{\Delta}} + \{\theta \overset{B(z)}{\mapsto} x\#y, z \overset{A}{\mapsto} W'\} \\
\text{desb-desact} \Longrightarrow_1 & H_{\overline{\Delta}} + \{z \overset{I}{\mapsto} W', \theta \overset{A}{\mapsto} x\#y\} \\
\text{HI (2)} \Longrightarrow_1^* & H_{\overline{\Theta}} : \theta \overset{A}{\mapsto} W
\end{array}
\end{array}
$$

<u>COMUNICACIÓN</u>:
La demostración es similar a las anteriores.

$$
\frac{\overline{\Gamma} + \{\theta \overset{B(ch)}{\mapsto} ch\} : ch \mapsto E \ \Downarrow\ \overline{\Delta} + \{\theta \overset{B(ch)}{\mapsto} ch\} : ch \mapsto W}{\overline{\Gamma} + \{ch \mapsto E\} : \theta \mapsto ch \ \Downarrow\ \mathsf{sat}(\overline{\Delta} \cup \eta(\mathsf{nh}(W, \overline{\Delta})) : \theta \mapsto \eta(W)) : \theta \mapsto \eta(W)}
$$

si $\neg \mathsf{d}(W, \Delta \cup \Delta^B)$.
Por hipótesis de inducción, $H_{\overline{\Gamma}} + \{\theta \overset{B(ch)}{\mapsto} ch, ch \overset{A}{\mapsto} E\} \implies_1^* H_{\overline{\Delta}} + \{\theta \overset{B(ch)}{\mapsto} ch, ch \mapsto W\}$.
Hay que demostrar, $H_{\overline{\Gamma}} + \{ch \overset{I}{\mapsto} E, \theta \overset{A}{\mapsto} ch\} \implies_1^* H_{\mathsf{sat}(\overline{\Delta} \cup \eta(\mathsf{nh}(W, \overline{\Delta})):\theta \mapsto \eta(W))} + \{\theta \overset{A}{\mapsto} \eta(W)\}$.

$$
\begin{array}{l}
H_{\overline{\Gamma}} + \{ch \overset{I}{\mapsto} E, \theta \overset{A}{\mapsto} ch\} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \begin{array}{rl}
\text{demanda} \implies_1 & H_{\overline{\Gamma}} + \{\theta \overset{B(ch)}{\mapsto} ch, ch \overset{A}{\mapsto} E\} \\
\text{HI} \implies_1^* & H_{\overline{\Delta}} + \{\theta \overset{B(ch)}{\mapsto} ch, ch \mapsto W\} \\
\text{desb-desact} \implies_1 & H_{\overline{\Delta}} + \{ch \overset{I}{\mapsto} W, \theta \overset{A}{\mapsto} ch\}
\end{array} \\
\qquad\qquad \implies_1 \left\{ \begin{array}{rl}
\text{comunicación} \longrightarrow_1 & H_{\overline{\Delta}} + \eta(\mathsf{nh}(W, \overline{\Delta})) + \{\theta \overset{A}{\mapsto} \eta(W)\} \\
\text{lema. 4.1.5} \overset{CPI}{\Longrightarrow}_1 & H_{\mathsf{sat}(\overline{\Delta} \cup \eta(\mathsf{nh}(W, \overline{\Delta})):\theta \mapsto \eta(W))} + \{\theta \overset{A}{\mapsto} \eta(W)\}
\end{array} \right.
\end{array}
$$

DEMANDA DE COMUNICACIÓN:
La demostración es similar a las anteriores. c.q.d.

TEOREMA 4.1.9. Equivalencia 2. Sean $E \in EExp \cup \mathcal{C}han$ y $H + \{\theta \overset{A}{\mapsto} E\}$ un *heap* etiquetado alcanzable.
Si $H + \{\theta \overset{A}{\mapsto} E\} \implies_1^* H' + \{\theta \overset{A}{\mapsto} W\}$, entonces $\overline{\Gamma}_H : \theta \mapsto E \Downarrow \overline{\Gamma}_{H'} : \theta \mapsto W$.

Demostración
Se hará por inducción sobre el cómputo para $H + \{\theta \overset{A}{\mapsto} E\}$.

CASO BASE: No se aplica ninguna regla; por tanto $H' = H$ y E ya está en *whnf*, es decir, $E \equiv \lambda x.E'$.
Por ser $H + \{\theta \overset{A}{\mapsto} \lambda x.E'\}$ un *heap* etiquetado alcanzable, entonces por el Corolario 4.1.7 $\overline{\Gamma}_H$ está saturado en $\theta \mapsto \lambda x.E'$ y se tiene

$$\overline{\Gamma}_H : \theta \mapsto \lambda x.E' \overset{\text{Lambda}}{\Downarrow} \overline{\Gamma}_H : \theta \mapsto \lambda x.E'.$$

REGLA VALOR: Si la primera regla del cómputo es la regla valor, se tiene que $E \equiv x$, y

$$
\begin{array}{l}
H + \{\theta \overset{A}{\mapsto} x\} = H^* + \{x \overset{I}{\mapsto} W, \theta \overset{A}{\mapsto} x\} \\
\qquad\qquad\qquad\qquad \implies_1 \left\{ \begin{array}{rl}
\text{valor} \longrightarrow_1 & H^* + \{x \overset{I}{\mapsto} W, \theta \overset{A}{\mapsto} W\} \\
* \overset{CPI}{\Longrightarrow}_1 & H^* + \{x \overset{I}{\mapsto} W, \theta \overset{A}{\mapsto} W\} = H + \{\theta \overset{A}{\mapsto} W\}
\end{array} \right.
\end{array}
$$

$*$ Como $H + \{\theta \overset{A}{\mapsto} E\}$ es un *heap* etiquetado alcanzable la creación de proceso impaciente no produce ningún cambio ya que el *heap* original H no tenía creaciones de proceso pendientes, y el cambio de $\theta \overset{A}{\mapsto} x$ (con $x \overset{I}{\mapsto} W \in H$) a $\theta \overset{A}{\mapsto} W$ no puede hacer factible ninguna creación de proceso en H.

De esta forma, $\overline{\Gamma}_H = \overline{\Gamma}_{H^*} + \{x \mapsto W\}$ está saturado en $\theta \mapsto x$ por el Corolario 4.1.7, igualmente $\overline{\Gamma}' = \overline{\Gamma}_{H^*} + \{\theta \overset{B(x)}{\mapsto} x\}$ está saturado en $x \mapsto W$. Aplicando la regla *Variable*,

$$\frac{\overline{\Gamma}_{H^*} + \{\theta \overset{B(x)}{\mapsto} x\} : x \mapsto W \overset{\text{Lambda}}{\Downarrow} \overline{\Gamma}_{H^*} + \{\theta \overset{B(x)}{\mapsto} x\} : x \mapsto W}{\overline{\Gamma}_H : \theta \mapsto x \Downarrow \overline{\Gamma}_H : \theta \mapsto W}$$

REGLA DEMANDA: Se necesita que $E \equiv \tau$ y $H = H^* + \{\tau \overset{I}{\mapsto} E\}$. No puede ser que $\tau \overset{B}{\mapsto} E \in H$, ya que por la proposición 3.1.10 el cómputo se bloquearía y no se alcanzaría ningún valor para θ.

$$\begin{array}{rl} H^* + \{\tau \overset{I}{\mapsto} E, \theta \overset{A}{\mapsto} \tau\} & \\ \xrightarrow{\text{demanda}}_1 & H^* + \{\theta \overset{B(\tau)}{\mapsto} \tau, \tau \overset{A}{\mapsto} E\} \\ \xrightarrow{\text{lema 3.1.13}}_1^* & H'' + \{\theta \overset{B(\tau)}{\mapsto} \tau, \tau \overset{A}{\mapsto} W\} \qquad (1) \\ \xrightarrow{\text{desb-desact}}_1 & H'' + \{\tau \overset{I}{\mapsto} W, \theta \overset{A}{\mapsto} \tau\} \qquad (2) \end{array}$$

- CASO 1: $\tau \in \mathit{Var}$

$$\begin{array}{rl} (2) \xrightarrow{\text{valor}}_1 & H'' + \{\tau \overset{I}{\mapsto} W, \theta \overset{A}{\mapsto} W\} \\ & = H' + \{\theta \overset{A}{\mapsto} W\} \end{array}$$

  A partir de (1), por hipótesis de inducción se tiene que

$$\overline{\Gamma}_{H^*} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto E \Downarrow \overline{\Gamma}_{H''} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto W$$

  Aplicando la regla *Variable*,

$$\frac{\overline{\Gamma}_{H^*} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto E \overset{\text{HI}}{\Downarrow} \overline{\Gamma}_{H''} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto W}{\underbrace{\overline{\Gamma}_{H^*} + \{\tau \mapsto E\}}_{\overline{\Gamma}_H} : \theta \mapsto \tau \ \Downarrow \ \underbrace{\overline{\Gamma}_{H''} + \{\tau \mapsto W\}}_{\overline{\Gamma}_{H'}} : \theta \mapsto W}$$

- CASO 2: $\tau \in \mathcal{C}\mathit{han}$
  - SUBCASO 2.1: $\neg\mathsf{d}(W, H'')$, esto implica que $\neg\mathsf{d}(W, \overline{\Gamma}_{H''})$

$$(2) \Longrightarrow_1 \begin{cases} \xrightarrow{\text{comunic.}}_1 & H'' + \eta(\mathsf{nh}(W, H'')) + \{\theta \overset{A}{\mapsto} \eta(W)\} \\ \overset{CPI}{\Longrightarrow}_1 & H''' + \eta(\mathsf{nh}(W, H'')) + \{\theta \overset{A}{\mapsto} \eta(W)\} \\ & = H' : \theta \overset{A}{\mapsto} \eta(W) \end{cases}$$

    A partir de (1) por hipótesis de inducción se tiene

$$\overline{\Gamma}_{H^*} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto E \Downarrow \overline{\Gamma}_{H''} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto W$$

    y puesto que el valor no tiene dependencias, se puede aplicar la regla *Comunicación* para obtener

$$\overline{\Gamma}_H : \theta \mapsto \tau \Downarrow \overline{\Gamma}_{H'} : \theta \mapsto \eta(W)$$

$$\frac{\overline{\Gamma}_{H^*} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto E \overset{\text{HI}}{\Downarrow} \overline{\Gamma}_{H''} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto W}{\underbrace{\overline{\Gamma}_{H^*} + \{\tau \mapsto E\}}_{\overline{\Gamma}_H} : \theta \mapsto \tau \;\Downarrow\; \underbrace{\mathsf{sat}(\overline{\Gamma}_{H''} + \eta(\mathsf{nh}(W, \overline{\Gamma}_{H''})) : \theta \mapsto \eta(W))}_{\underbrace{\overline{\Gamma}_{H'''} + \eta(\mathsf{nh}(W, \overline{\Gamma}_{H''}))}_{\overline{\Gamma}_{H'}}} : \theta \mapsto \eta(W)}$$

- SUBCASO 2.2: $\mathsf{d}(W, H'') \wedge \mathsf{pld}(W, H'') = z \overset{I}{\mapsto} E'$,
  luego $\mathsf{d}(W, \overline{\Gamma}_{H''}) \wedge \mathsf{pld}(W, \overline{\Gamma}_{H''}) = z \mapsto E'$

$$\begin{aligned}(2) = H''' + \{\tau \overset{I}{\mapsto} W, z \overset{I}{\mapsto} E', \theta \overset{A}{\mapsto} \tau\} \\ \overset{\text{dem. com.}}{\Longrightarrow_1} \quad & H''' + \{\tau \overset{I}{\mapsto} W, \theta \overset{B(z)}{\mapsto} \tau, z \overset{A}{\mapsto} E'\} \\ \overset{\text{lem. 3.1.13}}{\Longrightarrow_1^*} \quad & H^{iv} + \{\tau \overset{I}{\mapsto} W, \theta \overset{B(z)}{\mapsto} \tau, z \overset{A}{\mapsto} W'\} \\ \overset{\text{desb-desact}}{\Longrightarrow_1} \quad & H^{iv} + \{\tau \overset{I}{\mapsto} W, z \overset{I}{\mapsto} W', \theta \overset{A}{\mapsto} \tau\} \qquad (3)\end{aligned}$$

Si W no tiene dependencias se procede a realizar la comunicación (como en el subcaso 2.1). En caso contrario se vuelve a aplicar la demanda de comunicación hasta que todas las dependencias hayan sido resueltas (como los *heaps* son finitos este proceso termina en algún momento). Por lo tanto existe un *heap* etiquetado final en el que θ obtiene un valor que le ha sido comunicado:

$$(3) \Longrightarrow_1^* \quad H' + \{\theta \overset{A}{\mapsto} W''\}$$

Por hipótesis de inducción se tiene
(1) $\overline{\Gamma}_{H^*} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto E \Downarrow \overline{\Gamma}_{H''} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto W$,
(2) $\overline{\Gamma}_{H'''} + \{\tau \mapsto W, \theta \overset{B(z)}{\mapsto} \tau\} : z \mapsto E' \Downarrow \overline{\Gamma}_{H^{iv}} + \{\tau \mapsto W, \theta \overset{B(z)}{\mapsto} \tau\} : z \mapsto W'$,
(3) $\overline{\Gamma}_{H^{iv}} + \{\tau \mapsto W, z \mapsto W'\} : \theta \mapsto \tau \Downarrow \overline{\Gamma}_{H'} : \theta \mapsto W''$.
y aplicando la regla *Demanda de comunicación* y teniendo en cuenta cómo se corresponden los *heaps* etiquetados y los *heaps* ampliados , se obtiene el resultado buscado:

$$\frac{\begin{array}{c}\overline{\Gamma}_{H^*} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto E \overset{\text{HI (1)}}{\Downarrow} \overbrace{\overline{\Gamma}_{H'''} + \{z \mapsto E', \theta \overset{B(\tau)}{\mapsto} \tau\}}^{\overline{\Gamma}_{H'''} + \{z \mapsto E', \theta \overset{B(\tau)}{\mapsto} \tau\}} \overline{\Gamma}_{H''} + \{\theta \overset{B(\tau)}{\mapsto} \tau\} : \tau \mapsto W \\ \overline{\Gamma}_{H'''} + \{\tau \mapsto, \theta \overset{B(z)}{\mapsto} \tau\} : z \mapsto E' \overset{\text{HI (2)}}{\Downarrow} \overline{\Gamma}_{H^{iv}} + \{\tau \mapsto W, \theta \overset{B(z)}{\mapsto} \tau\} : z \mapsto W' \\ \overline{\Gamma}_{H^{iv}} + \{z \mapsto W', \tau \mapsto W\} : \theta \mapsto \tau \overset{\text{HI (3)}}{\Downarrow} \overline{\Gamma}_{H'} : \theta \overset{A}{\mapsto} W''\end{array}}{\underbrace{\overline{\Gamma}_{H^*} + \{\tau \mapsto E\}}_{\overline{\Gamma}_H} : \theta \mapsto \tau \;\Downarrow\; \overline{\Gamma}_{H'} : \theta \mapsto W''}$$

DEMANDA DE APLICACIÓN: En este caso se tiene $E \equiv x\ y$ y $H = H^* + \{x \overset{I}{\mapsto} E'\}$ con $E' \notin Val$.

$$\begin{array}{rl} H^* + \{x \overset{I}{\mapsto} E', \theta \overset{A}{\mapsto} x\ y\} & \\ \text{demanda ap.} \Longrightarrow_1 & H^* + \{\theta \overset{B(x)}{\mapsto} x\ y, x \overset{A}{\mapsto} E'\} \\ \text{lem. 3.1.13} \Longrightarrow_1^* & H'' + \{\theta \overset{B(x)}{\mapsto} x\ y, x \overset{A}{\mapsto} \lambda z.E''\} \\ \text{desb-desact} \Longrightarrow_1 & H'' + \{x \overset{I}{\mapsto} \lambda z.E'', \theta \overset{A}{\mapsto} x\ y\} \\ \beta\text{-red.} \Longrightarrow_1 & H'' + \{x \overset{I}{\mapsto} \lambda z.E'', \theta \overset{A}{\mapsto} E''[y/z]\} \\ \Longrightarrow_1^* & H''' + \{x \overset{I}{\mapsto} \lambda z.E'', \theta \overset{A}{\mapsto} W\} \\ & = H' : \theta \overset{A}{\mapsto} W \end{array}$$

Por hipótesis de inducción se tiene
(1) $\overline{\Gamma}_{H^*} + \{\theta \overset{B(x)}{\mapsto} x\ y\} : x \mapsto E' \Downarrow \overline{\Gamma}_{H''} + \{\theta \overset{B(x)}{\mapsto} x\ y\} : x \mapsto \lambda z.E''$, y
(2) $\overline{\Gamma}_{H''} + \{x \mapsto \lambda z.E''\} : \theta \mapsto E''[y/z] \Downarrow \overline{\Gamma}_{H'''} + \{x \mapsto \lambda z.E''\} : \theta \mapsto W$
Utilizando la regla *Aplicación*,

$$\frac{\overline{\Gamma}_{H^*} + \{\theta \overset{B(x)}{\mapsto} x\ y\} : x \mapsto E' \overset{\text{HI (1)}}{\Downarrow} \overline{\Gamma}_{H''} + \{\theta \overset{B(x)}{\mapsto} x\ y\} x \mapsto \lambda z.E'' \quad \overline{\Gamma}_{H''} + \{x \mapsto \lambda z.E''\} : \theta \mapsto E''[y/z] \overset{\text{HI (2)}}{\Downarrow} \overline{\Gamma}_{H''} + \{x \mapsto \lambda z.E''\} \theta \mapsto W}{\underbrace{\overline{\Gamma}_{H^*} + \{x \mapsto E'\}}_{\overline{\Gamma}_H} : \theta \mapsto x\ y \ \Downarrow\ \underbrace{\overline{\Gamma}_{H'''} + \{x \mapsto \lambda z.E''\}}_{\overline{\Gamma}_{H'}} : \theta \mapsto W}$$

β-REDUCCIÓN:
Análogo a la segunda parte de la regla *demanda de aplicación*.

LET:

$$\begin{array}{rl} H : \theta \overset{A}{\mapsto} \texttt{let}\ \{x_i = E_i\}_{i=1}^n\ \texttt{in}\ x & \\ \Longrightarrow_1 \begin{cases} \text{let} \longrightarrow_1 & H + \{y_i \overset{I}{\mapsto} E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n : \theta \overset{A}{\mapsto} x[y_j/x_j]_{j=1}^n \\ \overset{CPI}{\Longrightarrow}_1 & H'' + \{y_i \overset{I}{\mapsto} E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n : \theta \overset{A}{\mapsto} x[y_j/x_j]_{j=1}^n \\ & = H_1 : \theta \overset{A}{\mapsto} x[y_j/x_j]_{j=1}^n \end{cases} & \\ \Longrightarrow_1^* & H' : \theta \overset{A}{\mapsto} W \end{array}$$

Teniendo en cuenta que por hipótesis de inducción

$$\overline{\Gamma}_{H''} + \{y_i \mapsto E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n : \theta \mapsto x[y_j/x_j]_{j=1}^n \Downarrow \overline{\Gamma}_{H'} : \theta \mapsto W$$

se puede aplicar la regla *let*:

$$\frac{\overbrace{\mathsf{sat}(\overline{\Gamma}_H + \{y_i \mapsto E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n : \theta \mapsto x[y_j/x_j]_{j=1}^n)}^{\overline{\Gamma}_{H''} + \{y_i \mapsto E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n} : \theta \mapsto x[y_j/x_j]_{j=1}^n \overset{\text{HI}}{\Downarrow} \overline{\Gamma}_{H'} : \theta \mapsto W}{\overline{\Gamma}_H : \theta \mapsto \texttt{let}\ \{x_i = E_i\}_{i=1}^n\ \texttt{in}\ x \ \Downarrow\ \overline{\Gamma}_{H'} : \theta \mapsto W}$$

CREACIÓN DE PROCESO: Se tiene que $E \equiv x\#y$ y se supone que $\neg \mathsf{d}(x, H + \{\theta \overset{A}{\mapsto} x\#y\})$

$$
\begin{array}{rl}
H + \{\theta \overset{A}{\mapsto} x\#y\} & \\
\text{creac. proc.} \Longrightarrow_1 & H + \{i \overset{I}{\mapsto} y, o \overset{I}{\mapsto} \eta(x)\ z, z \overset{I}{\mapsto} i\} + \eta(\mathsf{nh}(x, H + \{\theta \overset{A}{\mapsto} x\#y\})) + \{\theta \overset{A}{\mapsto} o\} \\
& = H_1 + \{o \overset{I}{\mapsto} \eta(x)\ z, \theta \overset{A}{\mapsto} o\} \\
\text{demanda} \Longrightarrow_1 & H_1 + \{\theta \overset{B(o)}{\mapsto} o, o \overset{A}{\mapsto} \eta(x)\ z\} \\
\text{lem. 3.1.13} \Longrightarrow_1^* & H'' + \{\theta \overset{B(o)}{\mapsto} o, o \overset{A}{\mapsto} W\} \\
\text{desb-desact} \Longrightarrow_1 & H'' + \{o \overset{I}{\mapsto} W, \theta \overset{A}{\mapsto} o\} \\
(\Diamond) \Longrightarrow_1^* & H' + \{\theta \overset{A}{\mapsto} W\}
\end{array}
$$

($\Diamond$) Con un razonamiento análogo al de la regla demanda se tiene que θ alcanza el valor W. Por hipótesis de inducción se tiene

$$\overline{\Gamma}_{H_1} + \{o \mapsto \eta(x)\ z\} : \theta \overset{B(o)}{\mapsto} o \ \Downarrow\ \overline{\Gamma}_{H'} : \theta \mapsto W$$

y además $\neg \mathsf{d}(x, \Gamma^*)$ donde $\Gamma^* = \Gamma_H \cup \Gamma_H^B \cup \{\theta \mapsto x\#y\}$. Aplicando la regla *Creación de proceso*, se tiene que

$$\frac{\overbrace{\overline{\Gamma}_H + \{i \mapsto y, o \mapsto \eta(x)\ z, z \mapsto i\} + \eta(\mathsf{nh}(x, \Gamma^*))}^{\overline{\Gamma}_{H_1} + \{o \mapsto \eta(x)\ z\}} : \theta \mapsto o \overset{\text{HI}}{\Downarrow} \overline{\Gamma}_{H'} : \theta \overset{A}{\mapsto} W}{\overline{\Gamma}_H : \theta \mapsto x\#y \ \Downarrow\ \overline{\Gamma}_{H'} : \theta \overset{A}{\mapsto} W}$$

COMUNICACIÓN:
Similar al subcaso 2.1 de la regla demanda.

DEMANDA DE CREACIÓN DE PROCESO:
$\mathsf{d}(x, H^*) \wedge \mathsf{pld}(x, H^*) = z \overset{I}{\mapsto} E$ donde $H^* = H + \{z \overset{I}{\mapsto} E, \theta \overset{A}{\mapsto} x\#y\}$

$$
\begin{array}{rl}
H^* + \{z \overset{I}{\mapsto} E, \theta \overset{A}{\mapsto} x\#y\} & \\
\text{dem.creac. proc.} \Longrightarrow_1 & H^* + \{\theta \overset{B(z)}{\mapsto} x\#y, z \overset{A}{\mapsto} E\} \\
\Longrightarrow_1^* & H'' + \{\theta \overset{B(z)}{\mapsto} x\#y, z \overset{A}{\mapsto} W'\} \\
\text{desb-desact} \Longrightarrow_1 & H'' + \{z \overset{I}{\mapsto} W', \theta \overset{A}{\mapsto} x\#y\} \\
\Longrightarrow_1^* & H' + \{\theta \overset{A}{\mapsto} W\}
\end{array}
$$

Teniendo en cuenta que por hipótesis de inducción
(1) $\overline{\Gamma}_{H^*} + \{\theta \overset{B(z)}{\mapsto} x\#y\} : z \mapsto E \Downarrow \overline{\Gamma}_{H''} + \{\theta \overset{B(z)}{\mapsto} x\#y\} : z \mapsto W'$
(2) $\overline{\Gamma}_{H''} + \{z \mapsto W'\} : \theta \mapsto x\#y \Downarrow \overline{\Gamma}_{H'} : \theta \mapsto W$
y, además, $\mathsf{d}(x, \overline{\Gamma}^*)$ y $\mathsf{pld}(x, \overline{\Gamma}^*)$ (siendo $\overline{\Gamma}^* = \overline{\Gamma}_{H^*} \cup \{z \mapsto E, \theta \mapsto x\#y\}$) se puede aplicar la regla *Demanda de creación de proceso* para obtener

$$\overline{\Gamma}_{H^*} + \{\theta \overset{B(z)}{\mapsto} x\#y\} : z \mapsto E \overset{\text{HI (1)}}{\Downarrow} \overline{\Gamma}_{H''} + \{\theta \overset{B(z)}{\mapsto} x\#y\} : z \mapsto W'$$
$$\frac{\overline{\Gamma}_{H''} + \{z \mapsto W'\} : \theta \mapsto x\#y \overset{\text{HI (2)}}{\Downarrow} \overline{\Gamma}_{H'} : \theta \mapsto W}{\overline{\Gamma}_{H^*} + \{z \mapsto E\} : \theta \mapsto x\#y \Downarrow \overline{\Gamma}_{H'} : \theta \mapsto W}$$

DEMANDA DE COMUNICACIÓN:
Similar al subcaso 2.2 de la regla demanda. c.q.d.

4.2. Semántica denotacional estándar

Se comienza dando una ampliación de la semántica denotacional estándar para r-Jauja, es decir, la semántica denotacional de Abramsky ampliada con la creación de procesos. Se hace notar que se está trabajando con la semántica mínima, pues lo único que se quiere estudiar por el momento es el valor final de la variable *main*, por ello, se define la denotación de la creación de proceso, $x\#y$, como la de la aplicación, $x\ y$, ignorando los procesos y comunicaciones en cuanto a lo que al valor final de ambas expresiones se refiere. Sin embargo, hay que tener en cuenta la presencia de canales en los *heaps* que estamos considerando, por lo que es necesario ampliar la definición de entorno.

DEFINICIÓN 4.2.1. Un *entorno*, ρ, es una función que asocia variables y canales a valores,

$$\rho \in Env = Var \cup \mathcal{C}han \mapsto Val_\perp,$$

donde el conjunto de valores, $Val_\perp$, está formado por las expresiones de *EExp* en *whnf* y el valor indefinido ($\perp$).

□

$$\begin{aligned}
[\![\lambda x.E]\!]_\rho &= lift\lambda\epsilon.[\![E]\!]_{\rho[x\mapsto\epsilon]} \\
[\![x\ y]\!]_\rho &= drop([\![x]\!]_\rho)([\![y]\!]_\rho) \\
[\![x\#y]\!]_\rho &= [\![x\ y]\!]_\rho \\
[\![\theta]\!]_\rho &= \rho(\theta) \\
[\![\texttt{let}\ \{x_i = E_i\}_{i=1}^n\ \texttt{in}\ x]\!]_\rho &= [\![x[y_j/x_j]_{j=1}^n]\!]_{\{\!\{y_1\mapsto E_1[y_j/x_j]_{j=1}^n,\ \ldots,\ y_n\mapsto E_n[y_j/x_j]_{j=1}^n\}\!\}\rho}
\end{aligned}$$

Figura 4.1: Semántica denotacional de Jauja básico

Para dotar de significado a las expresiones se utiliza la función semántica:

$$[\![-]\!]_\rho : \ EExp \cup \mathcal{C}han \to Env \to Val_\perp$$

que viene definida en la Figura 4.1.

La recursión que genera la expresión *let* se captura mediante un entorno definido de forma recursiva mediante la siguiente función:

$$\{\!\{\ldots\}\!\} : \ Heap \to Env \to Env$$

que viene definida como:

$$\{\!\{x_1 \mapsto E_1, \ldots, x_n \mapsto E_n\}\!\}\rho = \mu\rho'.\rho \sqcup (x_1 \mapsto [\![E_1]\!]_{\rho'} \ldots x_n \mapsto [\![E_n]\!]_{\rho'})$$

donde μ representa el menor punto fijo del operador. Nótese que esta definición sólo tiene sentido si el entorno ρ y el *heap* $\Gamma = \{x_1 \mapsto E_1, \ldots, x_n \mapsto E_n\}$ son consistentes, es decir, en caso de que liguen una misma variable deben hacerlo a valores para los que exista una cota superior. Esta función puede verse como un modificador de entornos ya que extiende el entorno añadiendo, de forma adecuada, las ligaduras del *heap*. Una definición equivalente viene dada por:

$$\begin{array}{ll} \{\!\{\emptyset\}\!\}_\rho & = \rho \\ \{\!\{\Gamma,\ x \mapsto E\}\!\}_\rho & = \mu\rho'.\{\!\{\Gamma\}\!\}_{\rho'} \sqcup (x \mapsto [\![E]\!]_{\rho'}) \sqcup \rho \end{array}$$

Por lo tanto si ρ y Γ son consistentes se tiene que $\forall x.\rho(x) \sqsubseteq (\{\!\{\Gamma\}\!\}\rho)(x)$.

Se define un orden $\leq$ entre los entornos de forma que $\rho \leq \rho'$ significa que

$$\forall x \in Var.\rho(x) \neq \bot \Rightarrow \rho(x) = \rho'(x).$$

Así, si $\rho \leq \rho'$ entonces ρ' tiene más variables ligadas que ρ, pero una variable ligada en ρ tiene el mismo valor en ρ'. El orden de los entornos se fija con respecto a las variables y no a los canales. Los canales aparecen y desaparecen del heap y son sólo un medio para mantener la idea de que dos procesos interactuan entre sí comunicándose un valor a través de un canal.

En el caso de un *heap* ampliado, $\overline{\Gamma} = \langle \Gamma, \Gamma^B \rangle \in AHeap$, es suficiente con la información que se encuentra en Γ. Por ello se define la función *proyección* dada por:

$$\mathsf{pr} : AHeap \to Heap$$

$$\mathsf{pr}(\overline{\Gamma}) = \mathsf{pr}(\langle \Gamma, \Gamma^B \rangle) = \Gamma$$

.

En el siguiente lema se demuestra que el valor denotacional de una expresión coincide con el de cualquier renombramiento suyo.

Lema 4.2.2. *Sean $E \in EExp \cup \mathcal{C}han$, $\rho \in Env$ y $\overline{\Gamma} \in AHeap$ un heap ampliado consistente con ρ. Entonces,*

$$[\![E]\!]_{\{\!\{\mathsf{pr}(\overline{\Gamma})\}\!\}\rho} = [\![\eta(E)]\!]_{\{\!\{\mathsf{pr}(\eta(\mathsf{nh}(E,\overline{\Gamma})))\}\!\}\rho}$$

donde η es un renombramiento fresco.

Demostración
Sean $\rho, \rho' \in Env$ tales que $\rho' \leq \rho$. Si $[\![E]\!]_\rho \neq \bot$, entonces para obtener la denotación de E es suficiente con conocer ρ' tal que $\forall \tau \in Var \cup \mathcal{C}han.\tau \in VCh(\mathsf{nh}(E, \overline{\Gamma})) \Rightarrow [\![\tau]\!]_{\rho'} \neq \bot$. Por lo tanto,

$$[\![E]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma})|\!\}\rho} = [\![E]\!]_{\{\!|\mathsf{pr}(\mathsf{nh}(E,\overline{\Gamma}))|\!\}} = [\![\eta(E)]\!]_{\{\!|\mathsf{pr}(\eta(\mathsf{nh}(E,\overline{\Gamma})))|\!\}}$$

c.q.d.

PROPOSICIÓN 4.2.3. *Sea $\overline{\Gamma} = \langle \Gamma, \Gamma^B \rangle \in AHeap$ un heap ampliado en el que se quiere evaluar $\theta \mapsto E$ (donde $E \in EExp$ y $\theta \in Var \cup \mathcal{C}han$ tal que $\theta \notin VCh(\overline{\Gamma})$). Entonces $\{\!|\mathsf{pr}(\overline{\Gamma})|\!\}\rho \leq \{\!|\mathsf{pr}(\mathsf{sat}(\overline{\Gamma} : \theta \mapsto E))|\!\}\rho$.*

Demostración
Esto se debe a que la función de saturación sólo modifica $\mathsf{pr}(\overline{\Gamma})$ añadiendo ligaduras o modificando la expresión ligada a alguna variable o canal, pero nunca eliminando ligaduras. Hay que ver que el nuevo valor asociado es semánticamente equivalente. Se demostrará el caso de las creaciones de proceso factibles.
Sea $\tau \mapsto x\#y \in \Gamma$ tal que $\mathsf{d}(x, \overline{\Gamma} \cup \{\theta \mapsto E\})$ entonces $\tau \mapsto o \in \mathsf{pr}(\mathsf{sat}(\overline{\Gamma} : \theta \mapsto E))$. Sea $\rho' = \{\!|\mathsf{pr}(\overline{\Gamma})|\!\}$ y $\rho'' = \{\!|\mathsf{pr}(\mathsf{sat}(\overline{\Gamma} : \theta \mapsto E))|\!\}$, hay que ver que que $\rho'(\tau) = \rho''(tau)$ si $\rho'(\tau) \neq \bot$, es decir, $[\![x\#y]\!]_{\rho'} = [\![o]\!]_{\rho''}$.
Por un lado se tiene que $[\![x\#y]\!]_{\rho'} = [\![x\ y]\!]_{\rho''}$, por otro

$$\begin{array}{rcl} [\![o]\!]_{\rho''} & = & [\![\eta(x)\ z]\!]_{\rho''} \\ & = & drop([\![\eta(x)]\!]_{\rho''})([\![z]\!]_{\rho''}) \\ \text{Lema 4.2.2} = & & drop([\![x]\!]_{\rho'})([\![i]\!]_{\rho''}) \\ & = & drop([\![x]\!]_{\rho'})([\![y]\!]_{\rho''}) \\ & = & drop([\![x]\!]_{\rho'})([\![y]\!]_{\rho'}) \\ & = & [\![x\ y]\!]_{\rho'} \end{array}$$

c.q.d.

4.3. Corrección

En [21], Launchbury demostró que para las expresiones en *Exp* la semántica operacional natural es correcta con respecto a la semántica denotacional estándar, es decir, que el significado de los términos se conserva a lo largo del cómputo. En esta sección se va a extender este resultado para las expresiones extendidas en *EExp*. En [5] fue suficiente con demostrar la corrección para las nuevas expresiones con las que se extendió la semántica natural. Sin embargo, en este caso se considera oportuno reescribir toda la demostración dada por Launchbury, pues las reglas de la semántica natural extendida son considerablemente distintas a las de la semántica natural.

TEOREMA 4.3.1. *Sean* $E \in EExp \cup \mathcal{C}han$, $\overline{\Gamma} = \langle \Gamma, \Gamma^B \rangle, \overline{\Delta} = \langle \Delta, \Delta^B \rangle \in AHeap$ *y* $\theta \notin VCh(\overline{\Gamma})$.
Si $\overline{\Gamma} : \theta \mapsto E \Downarrow \overline{\Delta} : \theta \mapsto W$ *entonces para todo entorno* ρ,

$$[\![E]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma})|\!\}\rho} = [\![W]\!]_{\{\!|\mathsf{pr}(\overline{\Delta})|\!\}\rho} \ \wedge \ \{\!|\mathsf{pr}(\overline{\Gamma})|\!\}\rho \leq \{\!|\mathsf{pr}(\overline{\Delta})|\!\}\rho$$

Demostración
La demostración se hará por inducción sobre la derivación $\overline{\Gamma} : \theta \mapsto E \Downarrow \overline{\Delta} : \theta \mapsto W$.

REGLA LAMBDA:
La demostración es trivial.

REGLA VARIABLE:
Primero se demostrará que $\{\!|\Gamma + \{x \mapsto E\}|\!\}\rho \leq \{\!|\mathsf{pr}(\mathsf{sat}(\overline{\Delta} + \{x \mapsto W\} : \theta \mapsto W))|\!\}\rho$.

$$\begin{array}{rl}
\{\!|\Gamma + \{x \mapsto E\}|\!\}\rho & \\
\overset{\text{(def.)}}{=} & \mu\rho'.\{\!|\Gamma|\!\}\rho' \sqcup (x \mapsto [\![E]\!]_{\rho'}) \sqcup \rho \\
= & \mu\rho'.\{\!|\Gamma|\!\}\rho' \sqcup (x \mapsto [\![E]\!]_{\{\!|\Gamma|\!\}\rho'}) \sqcup \rho \\
\overset{\text{(HI)}}{=} & \mu\rho'.\{\!|\Gamma|\!\}\rho' \sqcup (x \mapsto [\![W]\!]_{\{\!|\Delta|\!\}\rho'}) \sqcup \rho \\
\leq & \mu\rho'.\{\!|\Delta|\!\}\rho' \sqcup (x \mapsto [\![W]\!]_{\{\!|\Delta|\!\}\rho'}) \sqcup \rho \\
\leq & \mu\rho'.\{\!|\Delta|\!\}\rho' \sqcup (x \mapsto [\![W]\!]_{\rho'}) \sqcup \rho \\
\overset{\text{(def.)}}{=} & \{\!|\Delta + \{x \mapsto W\}|\!\}\rho \\
\overset{\text{(prop. 4.2.3)}}{\leq} & \{\!|\mathsf{pr}(\mathsf{sat}(\overline{\Delta} + \{x \mapsto W\} : \theta \mapsto W))|\!\}\rho
\end{array}$$

A continuación se demostrará que $[\![x]\!]_{\{\!|\Gamma, x \mapsto E|\!\}\rho} = [\![W]\!]_{\{\!|\mathsf{pr}(\mathsf{sat}(\overline{\Delta} + \{x \mapsto W\} : \theta \mapsto W))|\!\}\rho}$.

$$\begin{array}{rl}
[\![x]\!]_{\{\!|\Gamma + \{x \mapsto E\}|\!\}\rho} & \\
= & \{\!|\Gamma + \{x \mapsto E\}|\!\}\rho(x) \\
\text{(por def. de } \leq) = & \{\!|\mathsf{pr}(\mathsf{sat}(\overline{\Delta} + \{x \mapsto W\} : \theta \mapsto W))|\!\}\rho(x) \\
= & [\![W]\!]_{\{\!|\mathsf{pr}(\mathsf{sat}(\overline{\Delta} + \{x \mapsto W\} : \theta \mapsto W))|\!\}\rho}
\end{array}$$

REGLA APLICACIÓN:
Primero se demostrará que $\{\!|\Gamma + \{x \mapsto E\}|\!\}\rho \leq \{\!|\Theta|\!\}\rho$:

$$\begin{array}{rl}
\{\!|\Gamma + \{x \mapsto E\}|\!\}\rho & \\
= & \mu\rho'.\{\!|\Gamma|\!\}\rho' \sqcup (x \mapsto [\![E]\!]_{\rho'}) \sqcup \rho \\
= & \mu\rho'.\{\!|\Gamma|\!\}\rho' \sqcup (x \mapsto [\![E]\!]_{\{\!|\Gamma|\!\}\rho'}) \sqcup \rho \\
\overset{\text{(HI)}}{=} & \mu\rho'.\{\!|\Gamma|\!\}\rho' \sqcup (x \mapsto [\![\lambda z.E']\!]_{\{\!|\Delta|\!\}\rho'}) \sqcup \rho \\
\leq & \mu\rho'.\{\!|\Delta|\!\}\rho' \sqcup (x \mapsto [\![\lambda z.E']\!]_{\rho'}) \sqcup \rho \\
= & \{\!|\Delta + \{x \mapsto \lambda z.E'\}|\!\}\rho \\
\leq & \{\!|\mathsf{pr}(\mathsf{sat}(\overline{\Delta} + \{x \mapsto \lambda z.E'\} : \theta \mapsto E'[y/z]))|\!\}\rho \\
\overset{\text{(HI)}}{\leq} & \{\!|\Theta|\!\}\rho
\end{array}$$

A continuación se demostrará que $[\![x\ y]\!]_{\{\!\{\Gamma+\{x\mapsto E\}\}\!\}\rho} = [\![W]\!]_{\{\!\{\Theta\}\!\}\rho}$:

$$\begin{array}{rl}
[\![x\ y]\!]_{\{\!\{\Gamma+\{x\mapsto E\}\}\!\}\rho} & \\
= & drop([\![x]\!]_{\{\!\{\Gamma+\{x\mapsto E\}\}\!\}\rho})([\![y]\!]_{\{\!\{\Gamma+\{x\mapsto E\}\}\!\}\rho}) \\
= & drop([\![E]\!]_{\{\!\{\Delta+\{x\mapsto E\}\}\!\}\rho})([\![y]\!]_{\{\!\{\Delta+\{x\mapsto E\}\}\!\}\rho}) \\
\overset{(\text{HI})}{=} & drop(\{\!\{\Delta+\{x\mapsto \lambda z.E'\}\}\!\}\rho(x))([\![y]\!]_{\{\!\{\Delta+\{x\mapsto\lambda z.E'\}\}\!\}\rho}) \\
= & drop([\![\lambda z.E']\!]_{\{\!\{\Delta+\{x\mapsto\lambda z.E'\}\}\!\}\rho})([\![y]\!]_{\{\!\{\Delta+\{x\mapsto\lambda z.E'\}\}\!\}\rho}) \\
= & (\lambda\nu.[\![E']\!]_{\{\!\{\Delta+\{x\mapsto\lambda z.E', z\mapsto\nu\}\}\!\}\rho})([\![y]\!]_{\{\!\{\Delta+\{x\mapsto\lambda z.E'\}\}\!\}\rho}) \\
= & [\![E']\!]_{\{\!\{\Delta+\{x\mapsto\lambda z.E', z\mapsto [\![y]\!]_{\{\!\{\Delta+\{x\mapsto\lambda z.E'\}\}\!\}\rho}\}\}\!\}\rho} \\
= & [\![E'[y/z]]\!]_{\{\!\{\Delta+\{x\mapsto\lambda z.E'\}\}\!\}\rho} \\
\overset{(\text{prop. 4.2.3})}{=} & [\![E'[y/z]]\!]_{\{\!\{\mathsf{pr}(\mathsf{sat}(\overline{\Delta}+\{x\mapsto\lambda z.E'\}:\theta\mapsto E'[y/z]))\}\!\}\rho} \\
\overset{(\text{HI})}{=} & [\![W]\!]_{\{\!\{\Theta\}\!\}\rho}
\end{array}$$

REGLA LET:
Se comienza demostrando que $\{\!\{\Gamma\}\!\}\rho \leq \{\!\{\Delta\}\!\}\rho$:

$$\begin{array}{rl}
\{\!\{\Gamma\}\!\}\rho & \\
\leq_{(y_i \text{ variables frescas})} & \{\!\{\Gamma + \{y_i \mapsto E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n\}\!\} \\
\leq_{(\text{prop. 4.2.3})} & \{\!\{\mathsf{pr}(\mathsf{sat}(\overline{\Gamma} + \{y_i \mapsto E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n : \theta \mapsto x[y_j/x_j]_{j=1}^n))\}\!\} \\
\overset{(\text{HI})}{\leq} & \{\!\{\Delta\}\!\}\rho
\end{array}$$

A continuación se demuestra que $[\![\texttt{let}\ \{x_i = E_i\}_{i=1}^n\ \texttt{in}\ x]\!]_{\{\!\{\Gamma\}\!\}\rho} = [\![W]\!]_{\{\!\{\Delta\}\!\}\rho}$:

$$\begin{array}{rl}
[\![\texttt{let}\ \{x_i = E_i\}_{i=1}^n\ \texttt{in}\ x]\!]_{\{\!\{\Gamma\}\!\}\rho} & \\
= & [\![x[y_j/x_j]_{j=1}^n]\!]_{\{\!\{\Gamma+\{y_i\mapsto E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n\}\!\}\rho} \\
= & [\![x[y_j/x_j]_{j=1}^n]\!]_{\{\!\{\mathsf{pr}(\mathsf{sat}(\overline{\Gamma}+\{y_i\mapsto E_i[y_j/x_j]_{j=1}^n\}_{i=1}^n:\theta\mapsto x[y_j/x_j]_{j=1}^n))\}\!\}\rho} \\
\overset{(\text{HI})}{=} & [\![W]\!]_{\{\!\{\Delta\}\!\}\rho}
\end{array}$$

REGLA CREACIÓN DE PROCESO:
Sea $\overline{\Gamma}_1 = \overline{\Gamma} + \{i \mapsto y, z \mapsto i, o \mapsto \eta(x)z\}a + eta(\mathsf{nh}(x, \overline{\Gamma}))$ Se comienza demostrando que $\{\!\{\Gamma\}\!\}\rho \leq \{\!\{\Delta\}\!\}\rho$:

$$\begin{array}{rl}
\{\!\{\Gamma\}\!\}\rho & \\
\leq_{(\text{las vars. nuevas son frescas})} & \{\!\{\mathsf{pr}(\overline{\Gamma}_1)\}\!\}\rho \\
\overset{(\text{HI})}{\leq} & \{\!\{\Delta\}\!\}\rho
\end{array}$$

A continuación se demuestra que $[\![x\#y]\!]_{\{\!\{\Gamma\}\!\}\rho} = [\![W]\!]_\rho\{\!\{\Delta\}\!\}\rho$ y que coincide con la denotación de la aplicación.
Por un lado utilizando la definición dada en la figura 4.1 se tiene

$$\begin{array}{rl}
[\![x\#y]\!]_{\{\!\{\Gamma\}\!\}\rho} & \\
= & [\![x\ y]\!]_{\{\!\{\Gamma\}\!\}\rho} \\
= & drop([\![x]\!]_{\{\!\{\Gamma\}\!\}\rho})([\![y]\!]_{\{\!\{\Gamma\}\!\}\rho})
\end{array}$$

Por otro lado, se sabe que

$$\begin{array}{rl} & [\![x\#y]\!]_{\{\!|\Gamma|\!\}\rho} \\ = & [\![o]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma}_1)|\!\}\rho} \\ = & [\![\eta(x)\ z]\!]_\rho \{\!|\mathsf{pr}(\overline{\Gamma}_1)|\!\}\rho \\ = & drop([\![\eta(x)]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma}_1)|\!\}\rho})([\![z]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma}_1)|\!\}}) \\ = & drop([\![\eta(x)]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma}_1)|\!\}\rho})([\![i]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma}_1)|\!\}}) \\ = & drop([\![\eta(x)]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma}_1)|\!\}\rho})([\![y]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma}_1)|\!\}}) \\ = & drop([\![x]\!]_{\{\!|\Gamma|\!\}\rho})([\![y]\!]_{\{\!|\Gamma|\!\}}) \\ = & [\![x\ y]\!]_{\{\!|\Gamma|\!\}\rho} \end{array}$$

Se ha aplicado el lema 4.2.2 y que para calcular la denotación de una expresión es suficiente conocer un entorno en el que las variables y canales del *heap* necesario de la expresión no estén indefinidas.
Además, por hipótesis de inducción se sabe que

$$[\![o]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma}_1)|\!\}\rho} = [\![W]\!]_{\{\!|\mathsf{pr}(\overline{\Delta})|\!\}}$$

DEMANDA DE CREACIÓN DE PROCESO:
Se probará que $\{\!|\Gamma + \{z \overset{I}{\mapsto} E'\}|\!\}\rho \leq \{\!|\Theta|\!\}\rho$

$$\begin{array}{rl} & \{\!|\Gamma + \{z \overset{E'}{\mapsto}\}|\!\}\rho \\ = & \mu\rho'.\{\!|\Gamma|\!\}\rho' \sqcup (z \mapsto [\![E']\!]_{\rho'}) \sqcup \rho \\ = & \mu\rho'.\{\!|\Gamma|\!\}\rho' \sqcup (z \mapsto [\![E']\!]_{\{\!|\Gamma|\!\}\rho'}) \sqcup \rho \\ \overset{(\mathrm{HI})}{\leq} & \mu\rho'.\{\!|\Delta|\!\}\rho' \sqcup (z \mapsto [\![W']\!]_{\{\!|\Delta|\!\}\rho'}) \sqcup \rho \\ \leq & \mu\rho'.\{\!|\Delta|\!\}\rho' \sqcup (z \mapsto [\![W']\!]_{\rho'}) \sqcup \rho \\ \overset{(\mathrm{def.})}{=} & \{\!|\Delta + \{z \mapsto W'\}|\!\}\rho \\ \overset{(\mathrm{HI})}{\leq} & \{\!|\Theta|\!\}\rho \end{array}$$

COMUNICACIÓN:
Se demuestra que $\{\!|\Gamma + \{ch \mapsto E\}|\!\}\rho \leq \{\!|\Delta + \eta(\mathsf{nh}(W, \Delta))|\!\}\rho$.

$$\begin{array}{rl} & \{\!|\Gamma + \{ch \overset{E}{\mapsto}\}|\!\}\rho \\ = & \mu\rho'.\{\!|\Gamma|\!\}\rho' \sqcup (ch \mapsto [\![E]\!]_{\rho'}) \sqcup \rho \\ = & \mu\rho'.\{\!|\Gamma|\!\}\rho' \sqcup (ch \mapsto [\![E]\!]_{\{\!|\Gamma|\!\}\rho'}) \sqcup \rho \\ \overset{(\mathrm{HI})}{\leq} & \mu\rho'.\{\!|\Delta|\!\}\rho' \sqcup (ch \mapsto [\![W]\!]_{\{\!|\Delta|\!\}\rho'}) \sqcup \rho \\ \leq & \mu\rho'.\{\!|\Delta|\!\}\rho' \sqcup (ch \mapsto [\![W]\!]_{\rho'}) \sqcup \rho \\ \overset{(\mathrm{def.})}{=} & \{\!|\Delta + \{ch \mapsto W\}|\!\}\rho \\ (*) \leq & \{\!|\Delta + \eta(\mathsf{nh}(W, \Delta))|\!\}\rho \end{array}$$

(∗) En este paso se ha tenido en cuenta que los canales no influyen en la definición de $\leq$ de los entornos, por ello no importa que el canal ch no esté en el entorno final. Por otro lado el entorno ha sido ampliado con las ligaduras de $\eta(\mathsf{nh}(W, \Delta))$ que están formadas por variables frescas (luego es consistente con Δ).

DEMANDA DE COMUNICACIÓN:
Se demuestra que $\{\!\{\Gamma + \{ch \mapsto E\}\}\!\}\rho \leq \{\!\{\Lambda\}\!\}\rho$.

$$\begin{array}{rl}
\{\!\{\Gamma + \{ch \overset{E}{\mapsto}\}\}\!\}\rho & \\
= & \mu\rho'.\{\!\{\Gamma\}\!\}\rho' \sqcup (ch \mapsto [\![E]\!]_{\rho'}) \sqcup \rho \\
= & \mu\rho'.\{\!\{\Gamma\}\!\}\rho' \sqcup (ch \mapsto [\![E]\!]_{\{\!\{\Gamma\}\!\}\rho'}) \sqcup \rho \\
\overset{\text{(HI)}}{\leq} & \mu\rho'.\{\!\{\Delta\}\!\}\rho' \sqcup (ch \mapsto [\![W]\!]_{\{\!\{\Delta\}\!\}\rho'}) \sqcup \rho \\
\leq & \mu\rho'.\{\!\{\Delta\}\!\}\rho' \sqcup (ch \mapsto [\![W]\!]_{\rho'}) \sqcup \rho \\
\overset{\text{(def.)}}{=} & \{\!\{\Delta + \{ch \mapsto W\}\}\!\}\rho \\
\overset{\text{(HI)}}{\leq} & \{\!\{\Theta + \{ch \mapsto W\}\}\!\}\rho \\
(1) \leq & \{\!\{\Theta + \{ch \mapsto W, z \mapsto W'\}\}\!\}\rho \\
\overset{\text{(HI)}}{\leq} & \{\!\{\Lambda\}\!\}\rho
\end{array}$$

(1) Por la formación de los *heaps* $z \mapsto E \notin \Theta$, luego si a dicho *heap* se le añade dicha ligadura el entorno obtenido será mayor.

La demostración de $[\![E]\!]_{\{\!\{\mathsf{pr}(\overline{\Gamma})\}\!\}\rho} = [\![W]\!]_{\{\!\{\mathsf{pr}(\overline{\Delta})\}\!\}\rho}$ en las últimas reglas se realiza razonando de forma similar a como se ha hecho con las demás reglas. c.q.d.

DEFINICIÓN 4.3.2. Sea $H \in EHeap$ un *heap* etiquetado alcanzable. La función

$$\{\!\{\ldots\}\!\} : \ EHeap \ \rightarrow Env \rightarrow Env$$

viene definida como $\{\!\{H\}\!\} = \{\!\{\mathsf{pr}(\overline{\Gamma}_H)\}\!\}$.

□

TEOREMA 4.3.3. Corrección. *Sean $E \in EExp \cup \mathcal{C}han$ y $H + \{\theta \overset{A}{\mapsto} E\}$ un heap etiquetado alcanzable. Si $H + \{\theta \overset{A}{\mapsto} E\} \Longrightarrow_1^* H' + \{\theta \overset{A}{\mapsto} W\}$ entonces para todo entorno ρ,*

$$[\![E]\!]_{\{\!\{H\}\!\}\rho} = [\![W]\!]_{\{\!\{H'\}\!\}\rho} \ \wedge \ \{\!\{H\}\!\}\rho \leq \{\!\{H'\}\!\}\rho$$

Demostración
Aplicando la equivalencia entre ambas semánticas dada en el Teorema 4.1.9 y la corrección para la semántica natural extendida expuesta en el Teorema 4.3.1.

c.q.d.

4.4. Adecuación

En la sección anterior se ha probado que si una reducción existe entonces la denotación de los términos se conserva. En esta sección se quiere ver cuándo existe dicha reducción.

El siguiente teorema establece que si una expresión $E \in EExp \cup \mathcal{C}han$ evalúa a un valor W, entonces su denotación no está indefinida.

Teorema 4.4.1. Adecuación I. *Sean $E \in EExp \cup \mathcal{C}han$, $\overline{\Gamma}, \overline{\Delta} \in AHeap$ y $\theta \notin VCh(\overline{\Gamma})$. Si $\overline{\Gamma} : \theta \mapsto E \Downarrow \overline{\Delta} : \theta \mapsto W$ entonces $[\![E]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma})|\!\}_\rho} \neq \perp$.*

Demostración
Es fácil de demostrar por tratarse de un corolario del Teorema 4.3.3. Puesto que W es un valor entonces $W \equiv \lambda x.E'$ y $[\![\lambda x.E']\!]_{\mathsf{pr}(\Delta)_\rho} \neq \perp$. c.q.d.

Por otro lado se establece también que si la denotación de una expresión está definida entonces existirá una derivación en la que dicha expresión obtenga un valor.

Teorema 4.4.2. Adecuación II. *Sean $E \in EExp \cup \mathcal{C}han$ y $\overline{\Gamma} \in AHeap$. Si $[\![E]\!]_{\{\!|\mathsf{pr}(\overline{\Gamma})|\!\}_\rho} \neq \perp$ entonces existe un heap ampliado $\overline{\Delta} \in AHeap$ y un valor $W \in Val$ tal que $\overline{\Gamma} : \theta \mapsto E \Downarrow \overline{\Delta} : \theta \mapsto W$ (para cualquier $\theta \notin VCh(\overline{\Gamma}) \cup VCh(\overline{\Delta})$).*

Demostración
Será necesario extender la semántica denotacional de recursos, $\mathcal{N}$, dada por Launchbury en [21]. Se trata de una variante de la semántica denotacional estándar y coincide con ésta si se dispone de infinitos recursos.
PENDIENTE

4.5. Determinación

En esta sección se va a demostrar que se obtiene el mismo resultado independientemente de si se aplica la semántica operacional de Jauja básico para un procesador o para n procesadores.

Por un lado se verá que si para la evaluación de la variable principal (*main*) es necesario evaluar una variable o canal θ, entonces θ también será evaluado aplicando la semántica para n procesadores.

Proposición 4.5.1. *Sea $E \in EExp$ una expresión extendida. Se supone que*
$\{main \overset{A}{\mapsto} E\} \Longrightarrow_1 H_1 \Longrightarrow_1 H_2 \Longrightarrow_1 \ldots$ *y*
$\{main \overset{A}{\mapsto} E\} \Longrightarrow K_1 \Longrightarrow K_2 \Longrightarrow \ldots$.
Si θ está activa en algún heap etiquetado H_i entonces existe j tal que θ está activa en K_j.

Demostración
Sea H_i con alguna variable o canal activo. Por la proposición 3.1.10 se tiene que

$$\mathcal{LAB}(H_i) = C_{H_i} = \{main \overset{B(\tau_1)}{\mapsto} E_1, \tau_1 \overset{B(\tau_2)}{\mapsto} E_2, \ldots, \tau_n \overset{A}{\mapsto} E\}$$

Es decir, τ_n corresponde a la única ligadura activa en H_i y está demandada por la variable principal.
En realidad, esto ocurre también para cada K_j $(j \leq i)$, porque si en alguno no hubiera ninguna ligadura activa, entonces el cómputo estaría completamente bloqueado. Así se tiene:

$$main \overset{A}{\mapsto} \tau_1, \tau_1 \overset{A}{\mapsto} E_1, \tau_2 \overset{A}{\mapsto} E_2, \ldots, \tau_i \overset{A}{\mapsto} E_i$$

con $\tau_j \in H_j$.
Basta ver que para cada regla $\Longrightarrow_1$ que nos permite pasar de $\tau_j \overset{A}{\mapsto} E_j$ a $\tau_{j+1} \overset{A}{\mapsto} E_{j+1}$ tiene su correspondencia en $\Longrightarrow$. c.q.d.

El siguiente teorema establece que el valor obtenido es el mismo independientemente de la semántica operacional que se utilize.

Teorema 4.5.2. Determinación *Sea $E \in EExp$. Si $\{main \overset{A}{\mapsto} E\} \Longrightarrow_1^* H + \{main \overset{A}{\mapsto} W\}$ y $\{main \overset{A}{\mapsto} E\} \Longrightarrow K_1 \Longrightarrow K_2 \Longrightarrow \ldots$, entonces*

1. *existe un $i \geq 1$ tal que $K_i = K_i' + \{main \overset{I}{\mapsto} W'\}$*
2. *para todo entorno ρ, $[\![W']\!]_{\{\!\{K_i'\}\!\}\rho} = [\![W]\!]_{\{\!\{H\}\!\}\rho}$*

Demostración
PENDIENTE

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

En este trabajo se ha querido demostrar la equivalencia entre la semántica operacional para una pequeña parte del lenguaje funcional paralelo Eden (el núcleo funcional junto con la creación de procesos) y una semántica denotacional estándar.

La semántica denotacional estándar solo refleja la relación funcional entrada-salida. Sin embargo, no se han querido abandonar por completo las características del modelo operacional que reflejan el paralelismo del lenguaje Eden, ya que aspiramos a poder establecer la equivalencia entre semánticas para Eden en un contexto más amplio.

Así, por ejemplo, aunque se han secuencializado por completo los cómputos, manteniendo una única hebra activa en cada paso, se ha mantenido la creación impaciente de procesos. Esto ha obligado a complicar sustancialmente las reglas de la semántica para un único procesador y también la extensión de la semántica natural de Launchbury. En realidad, esta extensión de la semántica natural representa la transformación de una semántica de paso corto en una de paso largo. Los efectos colaterales que provoca la creación impaciente de procesos son más fácilmente manejables en un sistema de transiciones de paso corto, pero no lo son tanto en uno de paso largo. Es por ello que ha habido que incrementar la información recogida en los *heaps* para la extensión de la semántica natural.

Si bien el artículo [5] ha servido de inspiración y guía para este trabajo, la tarea no ha sido fácil e inmediata. Al margen de numerosos errores encontrados en el citado trabajo, debe tenerse en cuenta que el paralelismo en el lenguaje GpH es mucho más limitado que en Eden, pues en el primero no existe la noción de proceso, y tampoco hay comunicaciones porque existe un heap único donde se comparten todas las ligaduras. Todo ello hace que el proceso de secuencialización del cómputo y la extensión correspondiente de la semántica de

Launchbury sean mucho más sencillos para GpH.

Las dificultades encontradas durante el desarrollo del trabajo han impedido completar todos los objetivos perseguidos. Aunque se ha logrado demostrar la equivalencia entre la semántica natural extendida y la semántica operacional restringida a un único procesador, y también la corrección de la semántica natural extendida con respecto a la semántica denotacional (teorema 4.3.1); han quedado pendientes tanto la adecuación entre las semánticas operacional y denotacional, como la determinación.

5.2. Trabajo futuro

Este trabajo supone una primera aproximación al objetivo principal perseguido, de demostrar la equivalencia entre las semánticas operacional y denotacional de Eden definidas en [9]. En este sentido, además de completar las demostraciones pendientes (adecuación y determinación), hay dos líneas claras de continuación del trabajo. Por un lado, ampliar la demostración de las tres propiedades principales aquí consideradas para la sintaxis completa de Jauja (el cálculo que representa la esencia del lenguaje Eden), es decir, incluyendo el operador de no determinismo, la comunicación mediante streams y los canales dinámicos. Y por otro lado, establecer la adecuación y la corrección de las semánticas no solo en términos de la relación funcional entrada-salida, sino también teniendo en cuenta otros aspectos observables en las semánticas de Eden y que son propios del paralelismo, tales como el sistema de procesos creado durante la ejecución de un programa y las comunicaciones realizadas entre esos procesos.

Con respecto al primer punto hay que señalar que Eden no es un lenguaje determinista por lo que la propiedad de *determinación* dejará de tener sentido en cuanto se introduzcan el operador de no determinismo. Aunque será posible probar un concepto similar basándose no en el valor final obtenido sino en el conjunto de posibles valores que se pueden producir.

Hay que señalar también que la semántica denotacional definida para Eden en [9] está inspirada en la propuesta dada en [17] y se trata de una semántica denotacional con continuaciones. Parte del trabajo que queda por realizar es relacionar la semántica denotacional estándar con la de continuaciones en el sentido de que el valor obtenido al calcular la denotación de una expresión es el mismo. También hay que investigar la forma de relacionar el mecanismo de continuaciones con el de pasos de reducción en la semántica operacional.

Para el estudio de la equivalencia entre ambos tipos de semántica podría resultar útil disponer de una implementación de las mismas en algún sistema de demostración automático (como por ejemplo Isabelle (http://www.cl.cam.ac.uk/research/hvg/Isabelle/index.html). De esta forma se facilitaría la comprobación de muchas propiedades sencillas cuya demostración (en general, por inducción) puede resultar larga y tediosa por la cantidad de reglas

involucradas y casos a considerar.

Bibliografía

[1] S. Abramsky. *Research Topics in Functional Programming*, chapter 4: The lazy lambda calculus, pages 65–117. Ed. D. A. Turner. Addison Wesley, 1990.

[2] S. Aditya, Arvind, L. Augustsson, J. Maessen, and R. Nikhil. Semantics of pH: A parallel dialect of Haskell. In P. Hudak, editor, *Haskell Workshop*, pages 35–49, La Jolla, Cambridge, MA, USA. Technical Report YALEU/DCS/RR-1075, June 1995.

[3] C. Baker-Finch. An abstract machine for parallel lazy evaluation. In *Trends in Functional Programming (Selected papers of the First Scottish Functional Programming Workshop)*, volume 1, pages 153–161. Intellect, 2000.

[4] C. Baker-Finch, D. King, and P. W. Trinder. An operational semantics for parallel lazy evaluation. In *ACM-SIGPLAN International Conference on Functional Programming (ICFP'00)*, pages 162–173, Montreal, Canada, September 2000.

[5] C. Baker-Finch, D. J. King, J. Hall, and P. W. Trinder. An operational semantics for parallel call-by-need. Technical Report 99/1, Faculty of Mathematics and Computing, The Open University, 1999.

[6] S. Breitinger, R. Loogen, Y. Ortega-Mallén, and R. Peña. Eden: Language definition and operational semantics. Technical Report 96/10, Reihe Informatik, FB Mathematik, Philipps-Universität Marburg, Germany, URL `http://www.mathematik.uni-marburg.de/~eden/`, 1996.

[7] M. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. Research Monographs in Parallel and Distributed Computing. Pitman, 1989.

[8] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):96–107, February 1992.

[9] M. Hidalgo-Herrero. *Semánticas formales para un lenguaje funcional paralelo*. PhD thesis, Dept. Sistemas Informáticos y Programación, Universidad Complutense de Madrid, 2004.

[10] M. Hidalgo-Herrero and Y. Ortega-Mallén. An operational semantics for the parallel language Eden. *Parallel Processing Letters (World Scientific Publishing Company)*, 12(2):211–228, 2002.

[11] M. Hidalgo-Herrero and Y. Ortega-Mallén. An operational semantics for the parallel language Eden. In *Proceedings of the 3rd International Workshop on Constructive Methods for Parallel Programming, CMPP'02*, pages 63–79. Technische Universitaet Berlin, Germany, 2002.

[12] M. Hidalgo-Herrero and Y. Ortega-Mallén. Continuation semantics for parallel Haskell dialects. In *Proc. of the 1st Asian Symposium on Programming Languages and Systems*, pages 303–321. LNCS 2895, Springer, 2003.

[13] M. Hidalgo-Herrero and Y. Ortega-Mallén. Dealing denotationally with stream-based communication. *Electronic Notes in Theoretical Computer Science*, 137(1):47–68, 2005.

[14] M. Hidalgo-Herrero, Y. Ortega-Mallén, and F. Rubio. Analyzing the influence of mixed evaluation on the performance of Eden skeletons. *Parallel Computing*, 32(7-8):523–538, 2006.

[15] C. A. R. Hoare. An axiomatic definition of the programming language PASCAL. *Acta Informatica*, 2(4):335–355, 1973.

[16] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science, 1985.

[17] M. B. Josephs. The semantics of lazy functional languages. *Theoretical Computer Science*, 68(1):105–111, 1989.

[18] G. Kahn and D. MacQueen. Coroutines and networks of parallel processes. In *IFIP'77*, pages 993–998. Eds. B. Gilchrist. North-Holland, 1977.

[19] P. Kelly. *Functional Programming for Loosely-Coupled Multiprocessors*. Pitman, 1989.

[20] U. Klusik, R. Loogen, S. Priebe, and F. Rubio. Implementation skeletons in Eden: Low-effort parallel programming. In *Proceedings of the 12th International Workshop on Implementation of Functional Languages, (IFL'00 selected papers)*, pages 71–88. LNCS 2011, Springer, 2001.

[21] J. Launchbury. A natural semantics for lazy evaluation. In *ACM Symposium on Principles of Programming Languages, POPL'93*, pages 144–154. ACM Press, 1993.

[22] R. Loogen. *Research Directions in Parallel Functional Programming*, chapter 3: Programming Language Constructs, pages 63–92. Eds. K. Hammond and G. Michaelson. Springer, 1999.

[23] R. Loogen, Y. Ortega-Mallén, and R. Peña. Parallel functional programming in Eden. *Journal of Functional Programming*, 15(3):431–475, 2005.

[24] R. Loogen, Y. Ortega-Mallén, R. Peña, S. Priebe, and F. Rubio. *Patterns and Skeletons for Parallel and Distributed Computing*, chapter 4: Parallelism Abstractions in Eden, pages 95–128. Eds. F. A. Rabhi and S. Gorlatch. Springer, 2002.

[25] R. Milner, M. Tofte, and R. Harper. *The definition of Standard ML*. MIT Press, 1990.

[26] R. Nikhil and Arvind. *Implicit Parallel Programing in pH*. Academic Press, 2001.

[27] S. L. Peyton Jones. *Haskell 98 language and libraries: the Revised Report*. Cambridge University Press, 2003.

[28] J. H. Reppy. Concurrent ML: Design, application and semantics. In *Proceedings of Functional Programming, Concurrency, Simulation and Automated Reasoning*, pages 165–198. LNCS 693, Springer, 1993.

[29] F. Rubio. *Programación funcional paralela eficiente en Eden*. PhD thesis, Dept. Sistemas Informáticos y Programación, Universidad Complutense de Madrid, 2001.

[30] J. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, Cambridge, MA, 1977.

[31] C. Strachey and C. P. Wadsworth. Continuations: A mathematical semantics for handling full jumps. *Higher-Order and Symbolic Computarion*, 13:135–152, 2000. Original Manuscript: Technical Monograph PRG-11, Oxford University Computing Laboratory, January, 1974.

[32] F. S. Taylor. *Parallel Functional Programming by Partitioning*. PhD thesis, Imperial College, 1997.

[33] P. W. Trinder, K. Hammond, H. W. Loidl, and S. L. Peyton Jones. Algorithm + Strategy = Parallelism. *Journal of Funcional Programming*, 8(1):23–60, 1998.

[34] P. W. Trinder, K. Hammond, J. Mattson Jr., A. Partridge, and S. L. Peyton Jones. GUM: a portable implementation of Haskell. In *Proceedings of Programming Language Design and Implementation, PLDI'96*, pages 78–88, Philadephia, USA, 1996. ACM Press.

[35] P. W. Trinder, H. W. Loidl, and R. F. Pointon. Parallel and Distributed Haskells. *Journal of Functional Programming*, 12(4+5):469–510, 2003.