Semántica Denotacional del Modelo de Actores

Rodolfo Palacios Ancona
Facultad de Ciencias de la Computación
Universidad Autónoma de Puebla

Asesor:
José de Jesús Lavalle Martínez
Facultad de Ciencias de la Computación
Universidad Autónoma de Puebla

Agosto 2000

...a mi mamá Elvira †

...a mi papá Rodolfo

...a mi hermana Claudette

...a mi novia Andrea

...gracias.

Índice General

Introducción

El campo de investigación en cuanto a la semántica de lenguajes de programación se ha centrado principalmente en el enfoque operacional. La semántica operacional describe el significado de un lenguaje de programación especificando como éste se ejecuta en una máquina abstracta. El propósito de esta tesis es el de proporcionar una base teórica para el desarrollo de sistemas distribuidos. Para esto utilizaremos el poder matemático de la semántica denotacional (también conocida como semántica matemática) la cual fue inicialmente introducida por Dana Scott y Christopher Strachey en 1971 [Sto77]. Su inspiración se basa en la observación de que las matemáticas proveen un conjunto poderoso de herramientas para expresar o modelar los efectos de la sintaxis. En particular, el concepto matemático de función provee una forma muy natural y conveniente para modelar el comportamiento de un programa.

La semántica denotacional es considerada como una metodología para proporcionar un significado preciso a los lenguajes de programación. Mientras que la sintaxis de un lenguaje es siempre especificada en una variante de la forma de Bakus-Naur (BNF), la parte más importante de la definición de su semántica es presentada principalmente utilizando lenguaje natural, lo cual es ambiguo y deja muchas preguntas abiertas. Para el estudio más profundo de estos lenguajes de programación, es necesario ir más allá de lo que nos permite el lenguaje natural.

A continuación se presentan dos definiciones que nos permiten tener un enfoque más claro de las diferencias principales entre la semántica denotacional y operacional.

1. **Semántica operacional (Stoy)[Sto77]:** Para la obtención de la semántica operacional de un lenguaje de programación es necesaria la

definición de una máquina abstracta, la cual tiene un estado, posiblemente con varios componentes, y algún conjunto de instrucciones primitivas. Definimos la máquina especificando como los componentes del estado son modificados por cada una de las instrucciones. Es entonces cuando definimos la semántica del lenguaje en cuestión tomando en cuenta estos terminos.

2. **Semántica denotacional (Strachey)[MG91]:** En la semántica denotacional un lenguaje es definido mediante funciones de valuación, las cuales mapean programas en objetos matemáticos llamados denotaciones. De esta forma el razonar sobre un programa se reduce al razonamiento sobre sus denotaciones. Dos programas son equivalentes si y solo si sus denotaciones son iguales.

La semántica denotacional provee el conjunto de técnicas más profundas y más ampliamente aplicables, las cuales a la vez están basadas en una fuerte teoría matemática.

Como se mencionó anteriormente, el estudio de los lenguajes de programación se ha basado principalmente en el punto de vista operacional. El caso del modelo de actores no es la excepción. Es por esto que utilizaremos el enfoque denotacional para la definición de la semántica del modelo de actores.

El modelo computacional de actores fue originalmente propuesto por C. Hewitt en 1977. Los actores son entidades autocontenidas de un sistema que interactúan de forma concurrente. Estas entidades se comunican mediante el paso de mensajes, el cual es asíncrono y efectivo. Una de las principales características del modelo de actores es que los actores pueden ser creados de forma dinámica y la topología de estos sistemas puede cambiar dinámicamente.

El campo de investigación de los sistemas distribuidos es relativamente amplio. Álgebras de procesos como CCS y CSP se han desarrollado como formalismos para el estudio de sistemas concurrentes. Estas álgebras de procesos presentan limitaciones en cuanto a la mobilidad de sus procesos, es decir, la posibilidad de crear nuevos objetos en el momento de ejecución o de cambiar la locación de los mismos. El cálculo-π puede ser considerado como el principal intento para superar estas limitaciones.

El modelo de actores lidia directamente con muchas de las características de los sistemas orientados a objetos, tales como la identidad de objetos, paso asíncrono de mensajes, mecanismos implícitos de recepción de mensajes y soporte para la creación de nuevos objetos; un actor tiene las mismas propiedades estructurales y de interacción de los objetos. La computación local realizada por los actores puede ser especificada en cualquier lenguaje de programación secuencial, siendo los lenguajes funcionales los utilizados con mayor frecuencia.

Dentro del desarrollo de esta tesis haremos a un lado el cálculo local de los actores para centrarnos en los aspectos estructurales y de interacción. Esto nos permite enfocar nuestra atención en la comunicación entre los actores y su concurrencia y de esta forma olvidarnos de los asuntos relacionados con la ejecución secuencial de programas dentro de los actores.

En el primer capítulo presentaremos la semántica operacional del modelo de actores definida por Gul A. Agha en su artículo *A Foundation for Actor Computation* [Agh93], la cual nos será de gran utilidad para la definición de la semántica denotacional. La semántica definida por Agha se caracteriza por su expresividad y sencillez.

Semántica Operacional

Los actores son componentes independientes, interactivos y autocontenidos de un sistema, los cuales se comunican mediante el paso asíncrono de mensajes. Cada actor tiene una locación conceptual, su *dirección* y un *comportamiento*. Un actor puede saber de la existencia de otros actores si es que conoce sus direcciones correspondientes. Con el fin de no tener que lidiar con velocidades de procesadores y permitir el ruteo adaptativo, no se garantiza que se preserve el orden de los mensajes que son enviados. Sin embargo, se garantiza que los mensajes enviados serán recibidos en un lapso de tiempo no acotado pero finito, lo que es conocido como comunicación efectiva.

El cambio de estado de los actores se especifica mediante el uso de comportamientos de reemplazo. Cada vez que un actor procesa una comunicación, también calcula su propio comportamiento en respuesta a la siguiente comunicación que pueda procesar. El cambio en el comportamiento de un actor puede representar un simple cambio de las variables de estado, o puede representar cambios en las operaciones (métodos) los cuales son ejecutados en respuesta a los mensajes.

El modelo de actores provee primitivas para la coordinación de los componentes que llevan a cabo los cálculos locales. La unidad más simple de coordinación es un actor individual. En el modelo de actores también es posible la creación y manipulación de actores. El comportamiento de un actor se describe mediante una abstracción lambda la cual incorpora el código que será ejecutado cuando un mensaje es recibido. Las primitivas de los actores son:

1. **newadr** es una primitiva utilizada para la creación de actores. Esta primitiva crea un nuevo actor y le asigna una dirección nueva. Este

actor recién creado no está iniciado, es decir no cuenta con un comportamiento.

2. **initbeh** se utiliza para iniciar el comportamiento de un actor recién creado con $newadr()$. Al ejecutar $initbeh(a, b)$ el actor a es iniciado con comportamiento b.

3. **become** es utilizada para realizar cambios de comportamiento en los actores; $become(b)$ crea una actor anónimo para realizar el resto del cálculo corriente, cambia el comportamiento del actor que ejecuta la primitiva become por el comportamiento b y libera a dicho actor para poder aceptar otro mensaje.

4. **send** es una primitiva que se utiliza para el envío de mensajes; $send(a, v)$ crea un nuevo mensaje con receptor a y contenido v y coloca dicho mensaje en el sistema de entrega de mensajes.

Notación

Primeramente es necesaria la definición de un conjunto de variables X y un conjunto de átomos At. At contiene t y nil para booleanos, asi como constantes para los números naturales. F_n es el conjunto de operaciones primitivas de aridad n y $F = \bigcup_{n \in N} F_n$. Las primitivas de actores *initbeh*, *newadr*, *send* y *become* están contenidas en F. El conjunto de valores está conformado por átomos, variables y expresiones lambda. Utilizamos el constructor de pares pr. Más formalmente:

$$V = At \cup X \cup \lambda X.E \cup pr(V, V)$$

De la misma forma necesitamos de un conjunto de expresiones el cual se define a continuación.

$$E = At \cup X \cup \lambda X.E \cup app(E, E) \cup F_n(E^n)$$

Es decir, una expresión puede ser un átomo, una variable, una expresión lambda, una aplicación de expresiones utilizando el constructor app o una n-tupla de expresiones (E^n).

Con el propósito de poder modelar la comunicación efectiva, es necesaria la introducción del concepto de *hoyo*. Debido a que el orden de la recepción de los mensajes no está garantizado, muchas veces un mensaje llega antes que otro provocando que el actor que recibe el mensaje no sepa qué hacer con él. Esto se debe a que la información necesaria para continuar con el cálculo no está completa, generando así un *hoyo*. Utilizamos $\bullet$ para denotar a un hoyo.

El concepto de hoyo nos es muy útil principalmente cuando queremos hablar de modelos de distribución. Supongamos que tenemos un sistema de actores, los cuales interactúan unos con otros de forma distribuída. Ahora, supongamos que uno de estos actores envía una serie de mensajes a otro actor para que éste último lleve a cabo una acción. Como mencionamos anteriormente, el que una serie de mensajes llegue a su destino en el mismo orden en el que fue enviada no está garantizado. Debido a esto, es posible que unos mensajes lleguen a su destino antes que otros y no precisamente en el orden en el que fueron enviados. Al recibir uno de estos mensajes, el actor receptor no sabe que hacer con él, ya que la información que necesita para completar la acción está incompleta. En este momento se genera un hoyo, el cual será llenado posteriormente con los mensajes faltantes, permitiendo asi que el actor que los recibió pueda llevar a cabo la acción indicada.

Como podemos ver, los hoyos son una parte muy importante en los modelos de distribución, ya que nos permiten utilizar el envío asíncrono de mensajes.

Ahora, con los conceptos definidos anteriormente podemos definir el concepto de *contexto*. Un contexto es simplemente una expresión con hoyos. $C[e]$ denota el resultado de reemplazar los hoyos del contexto C por la expresión e.

Los contextos se definen más formalmente como sigue:

$$C = At \cup X \cup \lambda X.C \cup app(C, C) \cup F_n(C^n) \cup \{\bullet\}$$

Configuraciones de Actores

Una configuración de actores puede ser vista como una representación global de un sistema de actores con respecto a un observador idealizado. Contiene una colección de actores, mensajes, nombres de actores externos y nombres de actores recepcionistas. Los conjuntos de recepcionistas y de actores externos constituyen la interfaz de una configuración de actores con el exterior. Estos conjuntos especifican los actores que son visibles y qué conexiones se deben proveer para que la configuración funcione apropiadamente.

Un actor, puede estar únicamente en uno de tres estados a la vez, dichos estados se definen a continuación:

$(?_a)$:	no iniciado, un actor a recién creado.
(b):	listo para aceptar mensajes, donde b es un comportamiento.
$[e]$:	ocupado ejecutando e, donde e representa el estado de procesamiento local del actor.

Formalmente, el conjunto de los estados de actores se define como:

$$As = (?_x) \oplus (V) \oplus [E]$$

Donde $(?_x)$ representa el conjunto de los actores no iniciados, (V) el conjunto de los actores listos para aceptar mensajes y $[E]$ el conjunto de los actores que se encuentran actualmente ocupados. Es importante hacer notar que el conjunto de los estados de actores se forma mediante la unión exclusiva de estos tres conjuntos, ya que un actor puede estar sólo en uno de los tres conjuntos a la vez.

$$(?_x) = \{a \in X \mid (?_{a'})_a\}$$

$$(V) = \{a \in X \mid (b)_a\}$$

$$[E] = \{a \in X \mid [e]_a\}$$

Un mensaje m contiene la dirección del actor al cual se le envía, asi como el contenido del mensaje, es decir $m = \langle a \Leftarrow v \rangle$.

Restringimos el contenido de v a cualquier valor construido a base de átomos y direcciones de actores usando el constructor de pares pr. Llamamos a estos valores *valores comunicables*. No está permitido comunicar abstracciones lambda ni estructuras que contengan estructuras lambda debido a que si se permite dicha comunicación se violaría el principio de los actores de que sólo un actor puede cambiar su propio comportamiento. Por lo tanto los valores comunicables se definen como:

$$cV = At \cup X \cup pr(cV, cV)$$

Por lo tanto, un mensaje contiene la dirección del actor al cual está dirigido y el contenido mismo del mensaje el cual como ya mencionamos, debe pertenecer al conjunto de los valores comunicables cV:

$$M = \{(a, cv) \mid a \in X,\ cv \in cV\}$$

Dividimos las transiciones de configuraciones de actores en internas y externas. Las transiciones internas son:

rcv : recepción de un mensaje por un actor que no se encuentra actualmente ocupado realizando algún cálculo.

$exec$: un actor ejecutando un paso del cálculo actual, el cual puede ser cualquiera de las primitivas de actores, es decir, $newadr$, $initbeh$, $become$ o $send$.

En adición a las transiciones internas de una configuración, se encuentran las transiciones externas:

in : llegada de un mensaje a un recepcionista desde el exterior.

out : envío de un mensaje a un actor externo.

Sea Ad un conjunto contable de direcciones de actores. Para simplificar la notación identificaremos a Ad con X y llamaremos a las variables usadas de esta forma *nombres de actores*.

Definición (Configuraciones de actores (K)): Una configuración con mapeo de actores α, un multiconjunto de mensajes μ, recepcionistas ρ y actores externos χ se escribe como:

$$\langle\langle \alpha \mid \mu \rangle\rangle^{\rho}_{\chi}$$

donde $\rho, \chi \in P_w[X]$ (subconjuntos finitos de X), $\alpha \in X \to As$ y $\mu \in M_w[M]$ (multiconjuntos finitos con elementos en M). Más aún, se requiere que siendo $A = Dom(\alpha)$ se satisfagan las siguientes restricciones:

1. $\rho \subseteq A$ y $A \cap \chi = \emptyset$
2. si $a \in A$ entonces $FV(\alpha(a)) \subseteq A \cup \chi$ y si $\alpha(a) = (?_{a'})$ entonces $a' \in A$
3. si $\langle a \Leftarrow v \rangle \in \mu$ entonces $FV(v) \cup \{a\} \subseteq A \cup \chi$

es decir, los actores recepcionistas ρ son un subconjunto de los actores que se encuentran dentro de la configuración y la intersección entre el conjunto de actores externos y el conjunto de actores que se encuentran dentro de la configuración es vacía. Decimos que una variable es fresca con respecto a un contexto si no ocurre libre o acotada en ninguna entidad sintáctica. Escribimos $FV(e)$ para denotar al conjunto de las variables libres de e.

Los recepcionistas ρ son nombres de actores dentro de la configuración, los cuales son visibles desde el exterior; todos los demás actores dentro de la configuración son locales y por lo tanto no son accedibles desde el exterior. Los actores externos χ son actores que se encuentran fuera de la configuración a los cuales se les puede enviar mensajes.

El mapa de actores α de una configuración se representa como una lista de estados de actores subindicados con la dirección del actor, la cual es mapeada a dicho estado.

Descomposición y Reducción

Con el motivo de poder describir las transiciones internas que no involucren recepción de mensajes, una expresión se descompone de forma única en un contexto de reducción llenado con un redex. Los contextos de reducción identifican la subexpresión de una expresión que va a ser evaluada a continuación.

Definición (E_{rdx}, R): El conjunto de redexes E_{rdx}, y el conjunto de contextos de reducción R, son definidos de la siguiente forma:

$$E_{rdx} = app(V, V) \cup (F_n(V^n) - pr(V, V))$$

$$R = \{\bullet\} \cup app(R, E) \cup app(V, R) \cup F_{n+m+1}(V^n, R, E^m)$$

Es decir, un redex no es otra cosa mas que un tipo especial de expresion. Un redex puede ser una aplicación de valores contenidos en V o una de las operaciones primitivas de aridad n, excluyendo a los pares compuestos por valores. Un contexto de reducción puede ser un hoyo, una aplicación de contextos de reducción en expresiones, una aplicación de valores en contextos de reducción o alguna de las operaciones primitivas con aridad $n + m + 1$.

Una expresión e es ya sea un valor o puede ser descompuesta de forma única en un contexto de reducción llenado con un redex ($R[r]$). De esta forma, el cálculo local de los actores es determinístico. A continuación se presenta el lema de descomposición única, el cual nos permite razonar más formalmente sobre las expresiones.

Lema (Descomposición única)

1. $e \in V$
2. $(\exists! R, r)(e = R[r])$

Los redexes pueden ser divididos en dos clases, puramente funcionales y redexes de actores. Como mencionamos anteriormente no nos ocuparemos del aspecto funcional. Los redexes de actores son *newadr()*, *initbeh*(a, b), *become*(b) y *send*(a, v).

La relación de transición de un solo paso $\mapsto$ sobre configuraciones de actores es generada por las siguientes reglas. A cada regla se le asigna una etiqueta l la cual indica la instrucción primitiva y los parámetros adicionales.

En todos los casos, menos en las transiciones de entrada/salida (con etiquetas *in*, *out*) el primer parámetro especifica al actor de foco de la transición.

Definición ($\mapsto$): La relación de transición de un solo paso $\mapsto$, sobre configuraciones de actores es la relación menor que satisfaga las siguientes condiciones.

$$\langle new : a, a' \rangle \qquad \langle\langle \alpha, [R[newadr()]]_a \mid \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[a']]_a, (?_a)_{a'} \mid \mu \rangle\rangle^{\rho}_{\chi}$$

$$\langle init : a, a' \rangle \qquad \langle\langle \alpha, [R[initbeh(a', b)]]_a, (?_a)_{a'} \mid \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_a, (b)_{a'} \mid \mu \rangle\rangle^{\rho}_{\chi}$$

$$\langle bec : a, a' \rangle \qquad \langle\langle \alpha, [R[become(b)]]_a \mid \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_{a'}, (b)_a \mid \mu \rangle\rangle^{\rho}_{\chi}$$

$$\langle send : a, m \rangle \qquad \langle\langle \alpha, [R[send(a', cv)]]_a \mid \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_a \mid \mu, m \rangle\rangle^{\rho}_{\chi}$$

$$\langle rcv : a, cv \rangle \qquad \langle\langle \alpha, (b)_a \mid \langle a \Leftarrow cv \rangle, \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [app(b, cv)]_a \mid \mu \rangle\rangle^{\rho}_{\chi}$$

$$\langle out : m \rangle \qquad \langle\langle \alpha \mid \mu, m \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha \mid \mu \rangle\rangle^{\rho'}_{\chi}$$

$$m = \langle a \Leftarrow cv \rangle,\ a \in \chi,\ \rho' = \rho \cup (FV(cv) \cap Dom(\alpha))$$

$$\langle in : m \rangle \qquad \langle\langle \alpha \mid \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha \mid \mu, \langle a \Leftarrow cv \rangle \rangle\rangle^{\rho}_{\chi \cup (FV(cv) - Dom(\alpha))}$$

$$m = \langle a \Leftarrow cv \rangle,\ a \in \rho,\ FV(cv) \cap Dom(\alpha) \subseteq \rho$$

En la regla de reducción lambda, las expresiones son reducidas bajo la suposición de que $Dom(a) \cup \{a\}$ son de hecho direcciones.

Hasta el momento hemos sido capaces de comprender más a fondo el modelo de actores, es decir, la forma en que los actores son creados y como se comunican. Hemos presentado la semántica operacional del modelo de actores según Agha [Agh93]

Semántica Denotacional

En el artículo escrito por Janet Goslett, Hardy Hulley y Austin Melton titulado *Denotational Semantics and Domain Teory* [MG91], se expone una metodología a seguir para la definición de la semántica denotacional de lenguajes de programación. En lo siguiente adoptaremos dicha metodología para la definición de la semántica denotacional del modelo de actores.

Definición de la Semántica

El significado de una construcción sintáctica es algún objeto matemático definido en términos de las denotaciones de sus subexpresiones. La *composicionalidad* es una técnica que nos permite construir el significado de expresiones complejas a partir de los significados de sus subexpresiones.

En el nivel más bajo permitimos que los caracteres terminales de la gramática del lenguaje sean denotados por entidades simples:

$C \in Contexto$

$$C ::= \quad newadr()_a \mid initbeh(a', b)_a \mid become(b)_a \mid send(a', cv)_a \mid rcv(a \Leftarrow cv) \mid$$

$$out(m) \mid in(m) \mid C_1; C_2$$

Dichas entidades simples son elementos de conjuntos llamados *dominios primitivos*. Para la definición de la semántica denotacional de actores se requerirán los siguentes dominios primitivos:

$P_w[X]$:	Conjunto de subconjuntos finitos de X.
$M_w[M]$:	Conjunto de multiconjuntos finitos con elementos en M.
X :	Nombres de actores.
As :	Estados de actores.
C :	Contextos.

Otro requerimiento para la definición de la semántica denotacional es la definición de ciertas operaciones conocidas como *constructores de dominios*, las cuales actúan sobre los dominios primitivos y forman lo que se conoce como *dominios compuestos*. El único constructor de dominios que utilizaremos será el producto cartesiano $\times$.

Para la definición de la semántica denotacional del modelo de actores son requeridos los siguientes dominios compuestos:

1. $K = (X \to As) \times M_w[M] \times P_w[X] \times P_w[X]$.

2. $(C \times K)$

En [MG91] se menciona que es posible que sean necesarias ciertas operaciones, llamadas *operaciones de dominio*, las cuales mapean elementos de un dominio a otro. En nuestro caso no utilizaremos ninguna operación de dominio.

Funciones de valuación

Las funciones que mapean los elementos de la sintaxis a sus respectivas denotaciones son llamadas *funciones de valuación*. En este caso sólo es necesaria la definición de una sola función de valuación como se muestra a continuación.

$\mathcal{AC} : (C \times K) \to K$

A continuación se presentan las funciones de valuación que tratan con los contextos de actores, siendo estos *new, init, bec, send, rcv, in* y *out*.

Para poder crear un nuevo actor utilizamos la primitiva de actores *newadr*, la cual devuelve un actor no iniciado a partir de un actor conocido. El único dominio que se ve afectado con la utilización de esta primitiva es α, que como mencionamos anteriormente es un conjunto de mapeos de nombres de actores a estados de actores. Esto se debe a que el actor recientemente creado se agrega al conjunto de los actores existentes, sin embargo, este nuevo actor no estará listo para recibir mensajes hasta que sea iniciado. Los demás dominios, μ, ρ, χ, no se ven afectados de ninguna forma.

$$\mathcal{AC}\lceil\langle new : a, a'\rangle, k\rceil = \qquad \{k((\alpha, [R[a']]_a), \mu, \rho, \chi) \mid a' \in (?_x)\}$$

Una vez que se crea un actor es necesario iniciarlo. Para hacer esto se requiere de la utilización de la primitiva *initbeh*, la cual le indica al nuevo actor el comportamiento que debe adoptar. Al igual que en la función anterior, la primitiva *initbeh* trabaja únicamente con actores, afectando de esta forma al dominio de actores α. Los demás dominios permanecen sin cambio alguno.

$$\mathcal{AC}\lceil\langle init : a, a'\rangle, k\rceil = \qquad \{k((\alpha, (b)_a), \mu, \rho, \chi) \mid a \in (V)\}$$

Como mencionamos anteriormente en ocasiones es necesario que un actor cambie su comportamiento con fines de distribución. La primitiva *become* nos permite cambiar el comportamiento de un actor que ha sido creado e iniciado anteriormente. *become* también es una primitiva que sólo afecta a los actores por lo que el único dominio modificado es α, mientras que los demás permanecen sin cambio.

$$\mathcal{AC}\lceil\langle bec : a\rangle, k\rceil = \qquad \{k((\alpha, (b)_a), \mu, \rho, \chi) \mid a \in (V)\}$$

El envío de mensajes es una característica primordial en el modelo de actores, ya que es mediante mensajes que los actores se comunican unos con otros. Para que un mensaje sea enviado es necesaria la utilización de la primitiva de actores *send*, la cual envía el contenido de un mensaje m a un actor a. El mensaje m es colocado en el sistema de entrega de mensajes. Posteriormente es recibido por el actor al cual le fue enviado. Durante este proceso se modifican dos de los cuatro dominios involucrados en una configuración de actores: α y μ. α es modificado por el actor que se encuentra ejecutando el envío del mensaje, mientras que μ se ve afectado debido a la adición de un nuevo mensaje en el conjunto de mensajes.

$$\mathcal{AC}\lceil\langle send : a, m\rangle, k\rceil = \qquad \{k(\alpha, (\mu, m), \rho, \chi) \mid m = \langle a' \Leftarrow cv\rangle\}$$

De nada serviría el poder enviar un mensaje si no hay una forma de recibirlo. Para que un actor pueda recibir un mensaje es necesario que éste no se encuentre actualmente ocupado realizando algún cálculo. La función de valuación *rcv* nos permite analizar el comportamiento de un actor cuando éste recibe un mensaje. Al ser recibido, dicho mensaje es apendizado con la información con la que cuenta actualmente el actor receptor (si ese es el caso), y finalmente el conjunto de mensajes es modificado debido a que fue atendido uno de los mensajes que se encontraban en dicho conjunto.

$$\mathcal{AC}\lceil\langle rcv : a, cv\rangle, k\rceil = \qquad \{k((\alpha, [app(b, cv)]_a), \mu, \rho, \chi) \mid m = \langle a \Leftarrow cv\rangle\}$$

Otra de las características antes mencionadas del modelo de actores es que es posible conocer la existencia de actores fuera de la configuración, permitiendo de esta forma la existencia de varias configuraciones de actores en un mismo sistema. Para que esta coexistencia de configuraciones tenga sentido es necesario permitir la comunicación entre ellas. La función de valuación *out* describe el significado del envío de un mensaje al exterior por medio de un actor recepcionista. El contenido de dicho mensaje cv es enviado a un actor externo a. El conjunto de mensajes μ es modificado debido a que uno de los mensajes que contenía fue enviado; el conjunto de actores recepcionistas es modificado de modo que $\rho' = \rho \cup (FV(cv) \cap Dom(\alpha))$.

$$\begin{aligned}\mathcal{AC}\lceil\langle out : m\rangle, k\rceil = \qquad & \{k(\alpha, \mu, \rho', \chi) \mid m = \langle a \Leftarrow cv\rangle,\ a \in \chi, \\ & \rho' = \rho \cup (FV(cv) \cap Dom(\alpha))\}\end{aligned}$$

De forma análoga es posible que uno de los actores externos conocidos envíe un mensaje a un actor recepcionista de otra configuración. La función de valuación *in* nos permite observar lo que sucede cuando uno de estos mensajes es recibido. De la misma forma que en la función anterior, el conjunto de mensajes se ve modificado, sólo que en esta ocasión se agrega un mensaje nuevo al conjunto el cual debe ser atendido. El conjunto de los actores recepcionistas no sufre cambio alguno, mientras que el conjunto de actores externos se modifica de tal forma que $\chi' = \chi \cup (FV(cv) - Dom(\alpha))$.

$$\begin{aligned}\mathcal{AC}\lceil\langle in : m\rangle, k\rceil = \qquad & \{k(\alpha, (\mu, m), \rho, \chi') \mid m = \langle a \Leftarrow cv\rangle,\ a \in \rho, \\ & \chi' = \chi \cup (FV(cv) - Dom(\alpha))\}\end{aligned}$$

Otro aspecto importante de la definición de la semántica es que la gramática que presentamos nos permite formar expresiones del tipo $L_1; L_2$. Para poder lidiar con expresiones de este tipo es necesaria la definición de una nueva denotación, a la cual nos referiremos como *composición*. A continuación presentamos dicha denotación.

$$\mathcal{AC}\lceil C_1; C_2, k\rceil = \mathcal{AC}\lceil C_2, k'\rceil \circ \mathcal{AC}\lceil C_1, k\rceil \qquad \text{donde } k' \in \mathcal{AC}\lceil C_1, k\rceil$$

Es de gran importancia hacer notar que los elementos de L que se encuentran dentro de los delimitadores "$\lceil$ "y "$\rceil$ "no son rédexes de actores, sino que son simplemente una denotación de los mismos.

Bisimulación

En este capítulo nos valdremos de la utilización de la bisimulación con fines de demostración de equivalencia de términos de actores. De la misma forma presentaremos una serie de ejemplos que nos permitirán observar las diferencias entre la utilización de la semántica operacional y la semántica denotacional y las ventajas de expresividad que presenta ésta última.

Existen dos tipos básicos de bisimulación: la *bisimulación síncrona* y la *bisimulación debil asíncrona*, la cual es la equivalencia correspondiente para lenguajes basados en comunicación asíncrona.

Bisimulación Síncrona

Una relación simétrica $\mathcal{R}$ sobre términos de actores ($\mathcal{R} \subseteq \Gamma \times \Gamma$) es una bisimulación si $(A, B) \in \mathcal{R}$ implica:

si $A \overset{\tau}{\mapsto} A'$ entonces existe B' tal que $B \overset{\tau}{\mapsto} B'$ y $(A', B') \in \mathcal{R}$.

donde Γ representa al conjunto de los términos de actores bien formados.

Dos actores A y B son bisimilares, $A \approx B$, si existe una bisimulación $\mathcal{R}$ tal que $(A, B) \in \mathcal{R}$.

Para los lenguajes basados en comunicación asíncrona se ha introducido una nueva noción de bisimulación asíncrona. La principal diferencia entre la bisimulación asíncrona y la bisimulación estándar (síncrona), es que en el caso asíncrono la acción de remover un mensaje y reintroducirlo inmediatamente es considerada como no observable.

A continuación presentaremos un ejemplo extraido del álgebra de actores de Gaspari y Zavattaro descrita en [GZ97]. En este caso utilizaremos la semántica denotacional del modelo de actores definida en capítulos anteriores junto con las propiedades descritas de bisimulación para poder probar la equivalencia de diferentes términos de actores.

Ejemplo 1
Debido a que en el modelo de actores existe no-determinismo en cuanto a la recepción de mensajes, se espera que la equivalencia de bisimulación no dependa del orden en el cual las operaciones *send* sean ejecutadas. Para ilustrar que la bisimulación captura esta noción consideramos un ejemplo simple de dos actores $A = Pair1$ y $B = Pair2$, los cuales reciben pares y envían los elementos del par al actor b de la siguiente forma: $Pair1$ los envía en el mismo orden en el que fueron recibidos, mientras que $Pair2$ los envía en orden inverso.

$$Pair1 \stackrel{def}{=} send(b, 1st(m)).send(b, 2nd(m)).become(Pair1)$$

$$Pair2 \stackrel{def}{=} send(b, 2nd(m)).send(b, 1st(m)).become(Pair2)$$

Presentaremos el ejemplo utilizando la semántica operacional definida por Agha [Agh93] y después utilizaremos la semántica denotacional que definimos anteriormente.

Comenzamos con el primer término $Pair1$. Inicialmente tenemos una configuración de actores, la cual cuenta con tres contextos de reducción que contienen las primitivas de actores que serán ejecutadas.

$Pair1$:

$$\begin{aligned} \langle\langle \alpha, &[R[send(b, 1st(m))]]_a, \\ &[R[send(b, 2nd(m))]]_a, \\ &[R[become(Pair1)]]_a \mid \mu\rangle\rangle^{\rho}_{\chi} \mapsto \end{aligned}$$

Al ser ejecutada la primera primitiva, observamos que la configuración ha cambiado conforme lo dictan las reglas para la evaluación de primitivas de la semántica operacional propuesta por Agha [Agh93]. En este caso la primitiva ejecutada fue $send(b, 1st(m))$, es decir, se le envió la primera parte del mensaje m al actor b. Como resultado tenemos ahora un contexto de reducción vacío $[R[nil]]_a$ y un nuevo mensaje m_1 productos de dicha ejecución.

$$\langle\langle \alpha, [R[nil]]_a, [R[send(b, 2nd(m))]]_a, \\ \quad [R[become(Pair1)]]_a \mid \mu, m_1\rangle\rangle^{\rho}_{\chi} \mapsto$$

Ahora es ejecutada la segunda primitiva $send(b, 2nd(m))$ la cual envía la segunda parte del mensaje al actor b. De la misma forma que en la configuración anterior, ahora tenemos un contexto de reducción vacío y un nuevo mensaje m_2 en el conjunto de mensajes μ.

$$\langle\langle \alpha, [R[nil]]_a, [R[nil]]_a, \\ \quad [R[become(Pair1)]]_a \mid \mu, m_1, m_2\rangle\rangle^{\rho}_{\chi} \mapsto$$

La última primitiva en ejecutarse es $become(Pair1)$, es decir, el actor reinicia su comportamiento. Como podemos ver en las reglas de la semántica operacional, la ejecución de una primitiva become nos deja con un contexto de reducción vacío y un actor listo para recibir mensajes $(Pair1)_a$.

$$\langle\langle \alpha, [R[nil]]_a, [R[nil]]_a, [R[nil]]_{a'}, \\ \quad (Pair1)_a \mid \mu, m_1, m_2\rangle\rangle^{\rho}_{\chi} \mapsto$$

Finalmente eliminamos los contextos de reducción vacíos para obtener la siguiente configuración, la cual cuenta con un actor a con comportamiento $Pair1$ listo para recibir mensajes y dos mensajes m_1 y m_2 en el conjunto de mensajes μ.

$$\langle\langle \alpha, (Pair1)_a \mid \mu, m_1, m_2\rangle\rangle^{\rho}_{\chi}$$

Ahora continuamos con el segundo término $Pair2$. Recordemos que este término envía el mensaje m al actor b en orden inverso, es decir, envía primero la segunda parte y después envía la primera parte.

$Pair2$:

$$\langle\langle \alpha, [R[send(b, 2nd(m))]]_a, \\ \quad [R[send(b, 1st(m))]]_a, \\ \quad [R[become(Pair2)]]_a \mid \mu\rangle\rangle^{\rho}_{\chi} \mapsto$$

En este caso la primitiva que se ejecuta primero es $send(b, 2nd(m))$ lo cual, según la regla de evaluación de la semántica operacional, nos deja un contexto de reducción vacío y un nuevo mensaje m_2 en el conjunto de mensajes μ.

$$\langle\langle \alpha, [R[nil]]_a, [R[send(b, 1st(m))]]_a,\\ \quad [R[become(Pair2)]]_a \mid \mu, m_2\rangle\rangle^{\rho}_{\chi} \mapsto$$

La siguiente primitiva en ejecutarse es $send(b, 1st(m))$, la cual envía la primera parte del mensaje al actor b dejando un contexto de reducción vacío y agregando un nuevo mensaje m_1 al conjunto de mensajes.

$$\langle\langle \alpha, [R[nil]]_a, [R[nil]]_a,\\ \quad [R[become(Pair2)]]_a \mid \mu, m_2, m_1\rangle\rangle^{\rho}_{\chi} \mapsto$$

Por último la primitiva become es ejecutada dejando a un actor a listo para recibir mensajes con comportamiento $Pair2$ y un contexto de reducción vacío.

$$\langle\langle \alpha, [R[nil]]_a, [R[nil]]_a, [R[nil]]_{a'},\\ \quad (Pair2)_a \mid \mu, m_2, m_1\rangle\rangle^{\rho}_{\chi} \mapsto$$

Finalmente, al igual que en el término $Pair1$ eliminamos los contextos de reducción innecesarios, lo que nos deja una configuración con un actor listo para recibir mensajes $(Pair2)_a$ y dos mensajes m_1 y m_2 en el conjunto de mensajes μ.

$$\langle\langle \alpha, (Pair2)_a \mid \mu, m_1, m_2\rangle\rangle^{\rho}_{\chi}$$

Si comparamos esta última expresión con la última expresión que obtuvimos del término $Pair1$ podemos concluir que son prácticamente la misma configuración, excepto por el nombre del comportamiento que cambia de $Pair1$ a $Pair2$.

A continuación presentamos el mismo análisis utilizando la semántica denotacional definida anteriormente.

Iniciamos con una función que contiene una lista de contextos, cada uno representando la denotación de las primitivas que se ejecutarán.

$Pair1:$

$\mathcal{AC}\lceil send(b, 1st(m))_a;\ send(b, 2nd(m))_a;\ bec(Pair1)_a,\ k\rceil =$

Para poder evaluar esta función es necesario utilizar la regla de composición de funciones que definimos anteriormente. Así, formamos una nueva función con el primer contexto y la componemos con la función resultante.

$\mathcal{AC}\lceil send(b, 2nd(m))_a;\ become(Pair1)_a,\ k'\rceil \circ \mathcal{AC}\lceil send(b, 1st(m))_a,\ k\rceil =$

Debido a que en la función original tenemos todavía más de un contexto, es necesario aplicar la composición de funciones una vez más, teniendo como resultado tres funciones con un término cada una.

$$\mathcal{AC}\lceil become(Pair1)_a,\ k''\rceil \circ \mathcal{AC}\lceil send(b, 2nd(m))_a,\ k'\rceil \circ \\ \mathcal{AC}\lceil send(b, 1st(m))_a,\ k\rceil =$$

Ahora procedemos a evaluar las funciones. Al evaluar la función que contiene el contexto $send(b, 1st(m))_a$ obtenemos de ella el siguiente conjunto $\{k(\alpha, (\mu, m_1), \rho, \chi)\}$, dentro del cual podemos notar que en el conjunto de mensajes encontramos un nuevo mensaje m_1 el cual es producto de la evaluación de la primera función. Tomamos este conjunto y lo sustituímos por k' en la segunda función para poder evaluarla.

$$\mathcal{AC}\lceil become(Pair1)_a,\ k''\rceil \circ \\ \mathcal{AC}\lceil send(b, 2nd(m))_a, k(\alpha, (\mu, m_1), \rho, \chi)\rceil =$$

De la misma forma evaluamos la segunda función y obtenemos un nuevo conjunto $\{k(\alpha, (\mu, m_1, m_2), \rho, \chi)\}$ con un nuevo mensaje m_2 producto de la evaluación de la segunda función. Utilizamos este resultado para poder evaluar la última función.

$\mathcal{AC}\lceil become(Pair1)_a, k(\alpha, (\mu, m_1, m_2), \rho, \chi)\rceil =$

Finalmente evaluamos la última función, con lo cual obtenemos un conjunto de configuraciones k.

$\{k((\alpha, (Pair1)_a), (\mu, m_1, m_2), \rho, \chi) \mid (Pair1)_a \in (V)\}$

Ahora realizaremos el mismo proceso con el segundo término de actores, $Pair2$, el cual envía el mensaje al actor b en orden inverso, es decir, envía primero la segunda parte y la primera parte después.

De la misma forma que en el término anterior iniciamos con una función que cuenta con tres contextos. Cada uno de estos contextos es la denotación de una primitiva de actores.

$Pair2:$

$\mathcal{AC}\lceil send(b, 2nd(m))_a;\ send(b, 1st(m))_a; become(Pair2)_a,\ k\rceil =$

Utilizamos la regla de composición de funciones y obtenemos ahora dos funciones nuevas.

$\mathcal{AC}\lceil send(b, 1st(m))_a;\ become(Pair2)_a,\ k'\rceil \circ \mathcal{AC}\lceil send(b, 2nd(m))_a,\ k\rceil =$

Debido a que la segunda función todavía cuenta con más de un contexto, es necesario volver a aplicar la regla de composición de funciones, obteniendo así tres funciones diferentes, una por cada contexto de la función original.

$$\mathcal{AC}\lceil become(Pair2)_a,\ k''\rceil \circ \mathcal{AC}\lceil send(b, 1st(m))_a,\ k'\rceil \circ \\ \mathcal{AC}\lceil send(b, 2nd(m))_a,\ k\rceil =$$

Lo que haremos a continuación será evaluar la primera función y obtenemos una nueva k que utilizamos para evaluar la segunda función.

$\mathcal{AC}\lceil become(Pair2)_a,\ k''\rceil \circ \mathcal{AC}\lceil send(b, 1st(m))_a, k(\alpha, (\mu, m_2), \rho, \chi)\rceil =$

De la misma forma utilizamos la configuración resultante de evaluar la segunda función para poder evaluar la tercera.

$\mathcal{AC}\lceil become(Pair2)_a,\ k(\alpha, (\mu, m_2, m_1), \rho, \chi)\rceil =$

Finalmente contamos con un conjunto de configuraciones k

$\{k((\alpha, (Pair2)_a), (\mu, m_1, m_2), \rho, \chi) \mid (Pair2)_a \in (V)\}$

Como podemos observar el conjunto que obtuvimos de la evaluación de las funciones del segundo término es prácticamente el mismo que el que obtuvimos de la evaluación de las funciones del primer término. La única diferencia que podemos notar es el nombre del término de actores que cambia de $Pair1$ a $Pair2$. Esto es irrelevante en cuanto a cuestiones de semántica. De esta forma podemos afirmar lo siguiente:

$$Pair1 \approx Pair2$$

Finalmente podemos asegurar que los términos A y B son bisimilares, esto es, $A \approx B$. De hecho, A y B no pueden ser distinguidos debido a que el orden de envío de mensajes no puede ser observado. De esta forma hemos podido observar la equivalencia de los actores A y B y mediante el uso de la semántica denotacional somos capaces de razonar más fácilmente sobre éstos.

A continuación presentamos la descripción de ambos términos de actores utilizando tanto la semántica operacional de Agha, como la semántica denotacional descrita anteriormente. De esta forma esperamos que sea más evidente la expresividad y sencillez de la semántica denotacional sobre la semántica operacional.

$Pair1 =$

$$\langle\langle \alpha, [R[send(b, 1st(m))]]_a,\\ [R[send(b, 2nd(m))]]_a,\\ [R[become(Pair1)]]_a \mid \mu\rangle\rangle^{\rho}_{\chi} \mapsto$$

$$\langle\langle \alpha, [R[nil]]_a, [R[send(b, 2nd(m))]]_a,\\ [R[become(Pair1)]]_a \mid \mu, m_1\rangle\rangle^{\rho}_{\chi} \mapsto$$

$$\langle\langle \alpha, [R[nil]]_a, [R[nil]]_a,\\ [R[become(Pair1)]]_a \mid \mu, m_1, m_2\rangle\rangle^{\rho}_{\chi} \mapsto$$

$$\langle\langle \alpha, [R[nil]]_a, [R[nil]]_a, [R[nil]]_{a'},\\ (Pair1)_a \mid \mu, m_1, m_2\rangle\rangle^{\rho}_{\chi} \mapsto$$

$$\langle\langle \alpha, (Pair1)_a \mid \mu, m_1, m_2\rangle\rangle^{\rho}_{\chi}$$

$Pair2 =$

$$\langle\langle \alpha, [R[send(b, 2nd(m))]]_a,\\ [R[send(b, 1st(m))]]_a,\\ [R[become(Pair2)]]_a \mid \mu\rangle\rangle^{\rho}_{\chi} \mapsto$$

$$\langle\langle \alpha, [R[nil]]_a, [R[send(b, 1st(m))]]_a,\\ [R[become(Pair2)]]_a \mid \mu, m_2\rangle\rangle^{\rho}_{\chi} \mapsto$$

$$\langle\langle \alpha, [R[nil]]_a, [R[nil]]_a,\\ [R[become(Pair2)]]_a \mid \mu, m_2, m_1\rangle\rangle^{\rho}_{\chi} \mapsto$$

$$\langle\langle \alpha, [R[nil]]_a, [R[nil]]_a, [R[nil]]_{a'},\\ (Pair2)_a \mid \mu, m_2, m_1\rangle\rangle^{\rho}_{\chi} \mapsto$$

$$\langle\langle \alpha, (Pair2)_a \mid \mu, m_1, m_2\rangle\rangle^{\rho}_{\chi}$$

$Pair1 =$

$\mathcal{AC}\lceil send(b, 1st(m))_a;\ send(b, 2nd(m))_a;\ bec(Pair1)_a,\ k\rceil =$

$\mathcal{AC}\lceil become(Pair1)_a,\ k''\rceil \circ$
$\quad \mathcal{AC}\lceil send(b, 2nd(m))_a,\ k'\rceil \circ$
$\quad \mathcal{AC}\lceil send(b, 1st(m))_a,\ k\rceil =$

$\mathcal{AC}\lceil become(Pair1)_a,\ k''\rceil \circ$
$\quad \mathcal{AC}\lceil send(b, 2nd(m))_a,\ k(\alpha, (\mu, m_1), \rho, \chi)\rceil =$

$\mathcal{AC}\lceil become(Pair1)_a,\ k(\alpha, (\mu, m_1, m_2), \rho, \chi)\rceil =$

$\{k((\alpha, (Pair1)_a), (\mu, m_1, m_2), \rho, \chi) \mid (Pair1)_a \in (V)\}$

$Pair2 =$

$\mathcal{AC}\lceil send(b, 2nd(m))_a;\ send(b, 1st(m))_a;\ become(Pair2)_a,\ k\rceil =$

$\mathcal{AC}\lceil become(Pair2)_a,\ k''\rceil \circ$
$\quad \mathcal{AC}\lceil send(b, 1st(m))_a,\ k'\rceil \circ$
$\quad \mathcal{AC}\lceil send(b, 2nd(m))_a,\ k\rceil =$

$\mathcal{AC}\lceil become(Pair2)_a,\ k''\rceil \circ$
$\quad \mathcal{AC}\lceil send(b, 1st(m))_a,\ k(\alpha, (\mu, m_2), \rho, \chi)\rceil =$

$\mathcal{AC}\lceil become(Pair2)_a,\ k(\alpha, (\mu, m_2, m_1), \rho, \chi)\rceil =$

$\{k((\alpha, (Pair2)_a), (\mu, m_1, m_2), \rho, \chi) \mid (Pair2)_a \in (V)\}$

Equivalencia

Es evidente que la semántica denotacional del modelo de actores que presentamos en capítulos anteriores está basada en la semántica operacional descrita por Gul A. Agha. Uno de los objetivos de esta tesis es demostrar la equivalencia entre estas dos semánticas y demostrar así mismo que la semántica denotacional que proponemos aventaja en varios aspectos a la operacional.

Debido a que la semántica denotacional fue obtenida basándonos en la semántica operacional de Agha, lo que resta es demostrar que nuestra semántica concuerda con ésta última. Comenzaremos por demostrar la concordancia de las primitivas de actores y finalmente demostraremos la equivalencia entre la composición de contextos de actores.

newadr(): Comenzaremos por analizar la primitiva $newadr()$. La representación de esta primitiva de forma operacional es la siguiente:

$$\langle new : a, a' \rangle \qquad \langle\langle \alpha, [R[newadr()]]_a | \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[a']]_a, (?_a)'_a | \mu \rangle\rangle^{\rho}_{\chi}$$

y su contraparte denotacional se representa de la siguiente forma:

$$\mathcal{AC}\lceil newadr()_a, k \rceil \;=\; \{k((\alpha, [R[a']]_a), \mu, \rho, \chi) \mid a' \in (?_x)\}$$

De primera instancia podemos observar que existen ciertos elementos que son comunes en ambas representaciones, mientras que hay algunos que aparecen en la semántica operacional pero que no se integran explícitamente en las denotaciones de nuestra semántica. Esto se debe principalmente a que los elementos que aparecen en las denotaciones son únicamente los elementos que se involucran directamente en la ejecución de la primitiva.

Supongamos que tenemos una denotación del tipo

$$\mathcal{AC}\lceil newadr()_a, k\rceil \;=\; \{k((\alpha, [R[a']]_a), \mu, \rho, \chi) \mid a' \in (?_x)\}$$

esto significa que la primitiva $newadr()$ crea un nuevo actor y lo mantiene no iniciado, es decir, $a' \in (?_x)$. Ahora, el hecho de que a' sea un elemento de $(?_x)$ significa que algún actor en un momento tuvo que crear a este nuevo actor a', es decir, como $[R[newadr()]]_a$ entonces $[R[a']]_a$ y $(?_a)'_a$.

Conocemos el hecho de que a' es un actor, de esta forma $a' \in \alpha$, es por esto que podemos afirmar que $(\alpha, [R[a']]_a, (?_a)'_a)$. Recordemos que uno de los principios básicos del modelo de actores es que un actor no puede existir por si mismo, sino que tiene que pertenecer por fuerza a una configuración, de tal forma que

$$\langle\langle \alpha, [R[newadr()]]_a | \mu \rangle\rangle^{\rho}_{\chi}$$

por otro lado tenemos que

$$\langle\langle \alpha, [R[a']]_a, (?_a)'_a | \mu \rangle\rangle^{\rho}_{\chi}$$

y una consecuencia inmediata de esto es que

$$\langle\langle \alpha, [R[newadr()]]_a | \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[a']]_a, (?_a)'_a | \mu \rangle\rangle^{\rho}_{\chi}$$

de tal forma que podemos asegurar que

$$\mathcal{AC}\lceil newadr()_a, k\rceil \;=\; \{k((\alpha, [R[a']]_a), \mu, \rho, \chi) \mid a' \in (?_x)\}$$

es equivalente a

$$\langle\langle \alpha, [R[newadr()]]_a | \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[a']]_a, (?_a)'_a | \mu \rangle\rangle^{\rho}_{\chi}$$

initbeh(a',b): La siguiente primitiva que analizaremos es $initbeh(a', b)$. Su representación de forma operacional es la siguiente:

$$\langle init : a, a' \rangle \qquad \langle\langle \alpha, [R[initbeh(a', b)]]_a, (?_a)_{a'} | \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_a, (b)_{a'} | \mu \rangle\rangle^{\rho}_{\chi}$$

y su representación en la semántica denotacional es

$$\mathcal{AC}\lceil initbeh(a', b)_a, k\rceil = \{k((\alpha, (b)_{a'}), \mu, \rho, \chi) \mid a' \in (V)\}$$

El propósito de la primitiva $initbeh(a', b)$ es el de inicializar el comportamiento de un actor recién creado por la primitiva $newadr()$, es decir, $a \in (V)$ es de la forma $(b)_a$. Como a es un actor al cual se le ha dado un comportamiento inicial tuvo que haberse llevado a cabo la ejecución de una primitiva $initbeh(a, b)$ de tal forma que si $[R[initbeh(a', b)]]_a$ entonces $(b)_{a'}$.

Ahora aplicamos el principio de que ningún actor puede existir de forma independiente, sino que por fuerza debe pertenecer a alguna configuración de forma que si

$$\langle\langle \alpha, [R[initbeh(a', b)]]_a, (?_a)_{a'} | \mu \rangle\rangle^{\rho}_{\chi}$$

entonces

$$\langle\langle \alpha, [R[nil]]_a, (b)_{a'} | \mu \rangle\rangle^{\rho}_{\chi}$$

de tal forma que

$$\langle\langle \alpha, [R[initbeh(a', b)]]_a, (?_a)_{a'} | \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_a, (b)_{a'} | \mu \rangle\rangle^{\rho}_{\chi}$$

y de esta forma podemos asegurar que

$$\mathcal{AC}\lceil initbeh(a', b)_a, k\rceil = \{k((\alpha, (b)_{a'}), \mu, \rho, \chi) \mid a' \in (V)\}$$

es equivalente a

$$\langle\langle \alpha, [R[initbeh(a', b)]]_a, (?_a)_{a'} | \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_a, (b)_{a'} | \mu \rangle\rangle^{\rho}_{\chi}$$

become(b): Si nos enfocamos en esta primitiva podemos observar que nos encontramos prácticamente con mismo caso que con la primitiva anterior *initbeh*(a', b), con la diferencia de que el actor a ya cuenta con un comportamiento. La representación operacional de esta primitiva es la siguiente:

$$\langle bec : a, a' \rangle \qquad \langle\langle \alpha, [R[become(b)]]_a | \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_{a'}, (b)_a | \mu \rangle\rangle^{\rho}_{\chi}$$

mientras que su función de valuación descrita en la semántica denotacional es

$$\mathcal{AC}\lceil become(b)_a, k \rceil = \{k((\alpha, (b)_a), \mu, \rho, \chi) \mid a \in (V)\}$$

Como ya sabemos, la primitiva *become*(b) nos permite cambiar el comportamiento de un actor por el comportamiento descrito por b, por lo que este actor debe ser de la forma $(b)_a \in (V)$. Para que esto sucediera sabemos que debió de haberse ejecutado una primitiva *become*(b), por lo tanto si $[R[become(b)]]_a$ entonces $[R[nil]]_{a'}$ y $(b)_a$ y como un actor no puede existir por si solo a menos que se encuentre dentro de una configuración entonces tenemos que

$$\langle\langle \alpha, [R[become(b)]]_a | \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_{a'}, (b)_a | \mu \rangle\rangle^{\rho}_{\chi}$$

con lo que podemos concluir que

$$\mathcal{AC}\lceil become(b)_a, k \rceil = \{k((\alpha, (b)_a), \mu, \rho, \chi) \mid a \in (V)\}$$

es equivalente a

$$\langle\langle \alpha, [R[become(b)]]_a | \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_{a'}, (b)_a | \mu \rangle\rangle^{\rho}_{\chi}$$

send(a',cv): Las tres funciones de valuación anteriores son denotaciones de primitivas que lidian principalmente con actores. Ahora nos enfocaremos a las primitivas que se ocupan del manejo de mensajes. Comenzaremos con la primitiva $send(a', cv)_a$, cuya representación operacional es la siguiente:

$$\langle send : a, m \rangle \quad \langle\langle \alpha, [R[send(a', cv)]]_a | \mu \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_a | \mu, m \rangle\rangle^{\rho}_{\chi}$$

y su contraparte en nuestra semántica denotacional es

$$\mathcal{AC}\lceil send(a', cv)_a, k\rceil = \{k(\alpha, (\mu, m), \rho, \chi) \mid m = \langle a' \Leftarrow cv\rangle\}$$

Mediante el uso de la primitiva $send(a', cv)_a$ le es posible a un actor a mandar un mensaje con contenido cv a un segundo actor a', es decir, existe un mensaje de la forma $m = \langle a' \Leftarrow cv\rangle$ que es agregado al conjunto de mensajes μ. De esta forma podemos asegurar que si se ejecutó una primitiva $[R[send(a', cv)]]_a$ entonces $[R[nil]]_a$ y $m \in \mu$. Una vez más debemos colocar a los actores dentro de sus respectivas configuraciones de modo que si

$$\langle\langle \alpha, [R[send(a', cv)]]_a | \mu\rangle\rangle^{\rho}_{\chi}$$

entonces

$$\langle\langle \alpha, [R[nil]]_a | \mu, m\rangle\rangle^{\rho}_{\chi}$$

por lo que tenemos que

$$\langle\langle \alpha, [R[send(a', cv)]]_a | \mu\rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_a | \mu, m\rangle\rangle^{\rho}_{\chi}$$

por lo que podemos asegurar que la expresión

$$\mathcal{AC}\lceil send(a', cv)_a, k\rceil = \{k(\alpha, (\mu, m), \rho, \chi) \mid m = \langle a' \Leftarrow cv\rangle\}$$

es equivalente a la expresión

$$\langle\langle \alpha, [R[send(a', cv)]]_a | \mu\rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [R[nil]]_a | \mu, m\rangle\rangle^{\rho}_{\chi}$$

rcv(a⇐cv): Otra de las primitivas que se ocupa del manejo de mensajes dentro de una configuración de actores es $rcv(a \Leftarrow cv)$. Como se mencionó en capitulos anteriores la representación de esta primitiva desde el punto de vista operacional es la siguiente:

$$\langle rcv : a, cv\rangle \qquad \langle\langle \alpha, (b)_a | \langle a \Leftarrow cv\rangle, \mu\rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha, [app(b, cv)]_a | \mu\rangle\rangle^{\rho}_{\chi}$$

mientras que desde el punto de vista denotacional es vista como

$$\mathcal{AC}\lceil rcv(a \Leftarrow cv), k\rceil = \{k((\alpha, [app(b, cv)]_a), \mu, \rho, \chi) \mid m = \langle a \Leftarrow cv\rangle\}$$

Esta primitiva le permite a los actores llevar a cabo la recepción de un mensaje enviado por otro actor, es decir, el actor receptor es un actor del tipo $[app(b, cv)]_a$, donde cv es el valor comunicable del mensaje recibido. Pero para que esto sea posible sabemos que es necesario que un actor haya enviado un mensaje del tipo $m = \langle a \Leftarrow cv\rangle$ dirigido a cierto actor $(b)_a$.

Teniendo esto en cuenta podemos colocar a dichos actores dentro de sus respectivas configuraciones de modo que si $m = \langle a \Leftarrow cv\rangle$ entonces $[app(b, cv)]_a$, de tal forma que si

$$\langle\langle\alpha, (b)_a | \langle a \Leftarrow cv\rangle, \mu\rangle\rangle^{\rho}_{\chi}$$

entonces

$$\langle\langle\alpha, [app(b, cv)]_a | \mu\rangle\rangle^{\rho}_{\chi}$$

por lo tanto podemos afirmar que la expresión

$$\mathcal{AC}\lceil rcv(a \Leftarrow cv), k\rceil = \{k((\alpha, [app(b, cv)]_a), \mu, \rho, \chi) \mid m = \langle a \Leftarrow cv\rangle\}$$

y la expresión

$$\langle\langle\alpha, (b)_a | \langle a \Leftarrow cv\rangle, \mu\rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle\alpha, [app(b, cv)]_a | \mu\rangle\rangle^{\rho}_{\chi}$$

son de hecho equivalentes.

out(m): Ahora nos enfocaremos a las primitivas de actores que nos permiten monitorear la entrada y la salida de mensajes que se manipulan entre diferentes configuraciones. La primera de estas primitivas es $out(m)$ cuya representación operacional es:

$$\langle out : m\rangle \qquad \langle\langle\alpha | \mu, m\rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle\alpha | \mu\rangle\rangle^{\rho'}_{\chi}$$

$$m = \langle a \Leftarrow cv \rangle,\ a \in \chi,\ \rho' = \rho \cup (FV(cv) \cap Dom(\alpha))$$

mientras que su representación denotacional es:

$$\mathcal{AC}\lceil out(m), k\rceil =$$

$$\{k(\alpha, \mu, \rho', \chi) \mid m = \langle a \Leftarrow cv \rangle,\ a \in \chi,\ \rho' = \rho \cup (FV(cv) \cap Dom(\alpha))\}$$

La primitiva $out(m)$ nos permite observar qué es lo que ocurre cuando un mensaje sale de una configuración dirigido a uno de los actores externos, es decir, si tenemos un mensaje del tipo $m \in \mu$ dirigido a un actor externo a entonces $a \in \chi$ de forma que $m = \langle a \Leftarrow cv \rangle$. Poniendo todo dentro de sus respectivas configuraciones podemos afirmar que si

$$\langle\langle \alpha | \mu, m \rangle\rangle^{\rho}_{\chi}$$

y

$$m = \langle a \Leftarrow cv \rangle,\ a \in \chi$$

entonces

$$\langle\langle \alpha | \mu \rangle\rangle^{\rho'}_{\chi}$$

por lo que podemos asegurar que

$$\langle\langle \alpha | \mu, m \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha | \mu \rangle\rangle^{\rho'}_{\chi}$$

y si tomamos en cuenta que $m = \langle a \Leftarrow cv \rangle,\ a \in \chi,\ \rho' = \rho \cup (FV(cv) \cap Dom(\alpha))$ entonces podemos asegurar que la expresión

$$\langle\langle \alpha | \mu, m \rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle \alpha | \mu \rangle\rangle^{\rho'}_{\chi}$$

es equivalente a tener

$$\mathcal{AC}\lceil out(m), k\rceil =$$

$$\{k(\alpha, \mu, \rho', \chi) \mid m = \langle a \Leftarrow cv\rangle,\ a \in \chi,\ \rho' = \rho \cup (FV(cv) \cap Dom(\alpha))\}$$

in(m): Finalmente tenemos otra de las primitivas que nos permiten manipular el envío de mensajes de una configuración a otra. Esta primitiva es $in(m)$ y su representación operacional es la siguiente:

$$\langle in : m\rangle \qquad \langle\langle\alpha|\mu\rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle\alpha|\mu, \langle a \Leftarrow cv\rangle\rangle\rangle^{\rho}_{\chi\cup(FV(cv)-Dom(\alpha))}$$

$$m = \langle a \Leftarrow cv\rangle,\ a \in \rho,\ FV(cv) \cap Dom(\alpha) \subseteq \rho$$

mientras que su representación denotacional es

$$\mathcal{AC}\lceil in(), k\rceil =$$

$$\{k(\alpha, (\mu, m), \rho, \chi')) \mid m = \langle a \Leftarrow cv\rangle,\ a \in \rho,$$

$$\chi' = \chi \cup (FV(cv) - Dom(\alpha))\}$$

La primitiva $in(m)$ nos permite observar qué es lo que ocurre cuando un mensaje entra a una configuración dirigido a uno de los actores recepcionistas, es decir, si tenemos un mensaje del tipo $m \in \mu$ dirigido a un actor recepcionista a entonces $a \in \rho$ de forma que $m = \langle a \Leftarrow cv\rangle$. Poniendo todo dentro de sus respectivas configuraciones podemos afirmar que si

$$\langle\langle\alpha|\mu\rangle\rangle^{\rho}_{\chi}$$

y

$$m = \langle a \Leftarrow cv\rangle,\ a \in \rho$$

entonces

$$\langle\langle\alpha|\mu, \langle a \Leftarrow cv\rangle\rangle\rangle^{\rho}_{\chi\cup(FV(cv)-Dom(\alpha))}$$

por lo que podemos asegurar que

$$\langle\langle\alpha|\mu\rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle\alpha|\mu, \langle a \Leftarrow cv\rangle\rangle\rangle^{\rho}_{\chi\cup(FV(cv)-Dom(\alpha))}$$

y si tomamos en cuenta que $m = \langle a \Leftarrow cv\rangle,\ a \in \rho,\ FV(cv) \cap Dom(\alpha) \subseteq \rho$ entonces podemos asegurar que la expresión

$$\langle\langle\alpha|\mu\rangle\rangle^{\rho}_{\chi} \mapsto \langle\langle\alpha|\mu, \langle a \Leftarrow cv\rangle\rangle\rangle^{\rho}_{\chi\cup(FV(cv)-Dom(\alpha))}$$

es equivalente a tener

$$\mathcal{AC}\lceil in(), k\rceil =$$

$$\{k(\alpha, (\mu, m), \rho, \chi')) \mid m = \langle a \Leftarrow cv\rangle,\ a \in \rho,$$

$$\chi' = \chi \cup (FV(cv) - Dom(\alpha))\}$$

Conclusiones

Como ya hemos mencionado anteriormente el propósito de esta tesis era el de proponer una semántica denotacional para el modelo de actores que aventajara en varios aspectos a la semántica operacional de actores propuesta por Gul A. Agha. En los últimos capítulos hemos sido capaces de demostrar precisamente que la semántica denotacional propuesta por nosotros no solo es más fácil de entender, sino que gracias a su expresividad matemática nos permite llegar mucho más allá de lo que podríamos hacerl con la semántica operacional.

Es necesario hacer notar que hemos demostrado la equivalencia entre la semántica denotacional que proponemos y la semántica operacional propuesta por Agha. Este es un aspecto muy importante ya que al ser equivalentes ambas semánticas es posible aplicar el desarrollo que se ha realizado en el campo operacional pero con las ventajas que aporta la visión denotacional.

Una de las principales ventajas que presenta la semántica denotacional que proponemos es que ya no es necesario lidiar con configuraciones del tipo que presenta Agha en su semántica operacional, sino que ahora tratamos únicamente con conjuntos, haciendo su manipulación de esta forma mucho más fácil. Esto se debe a que éstos conjuntos están sujetos a las mismas leyes y propiedades que cualquier conjunto matemático.

Bibliografía

[Agh93] Gul A. Agha. A foundation for actor computation. *Functional Programming 1 Cambridge University*, 1993.

[GZ97] Mauro Gaspari and Gianluigi Zavattaro. An algebra of actors. 1997.

[MG91] Austin Melton and Jane Goslet. Denotational semantics and domain theory. 1991.

[Sto77] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Aproach to Programming-Language Theory.* MIT Press, 1977.