Apuntes de Teoría de la Programación

22 de marzo de 2021

Contenidos

1 Introducción

1.1. Un ejemplo de la lógica

La pregunta que motiva todo lo siguiente es: ¿qué es el significado? O, más concretamente, ¿cuál es la relación entre la sintaxis y la semántica? Aunque aquí nos centraremos principalmente en especificar el comportamiento de programas, parece conveniente presentar un ejemplo relacionado con la lógica. Recordemos que, en lógica de primer orden, disponíamos de un método para asignar un valor semántico a cada expresión. En este caso, el valor semántico que nos interesa es la *denotación*, es decir, que por ejemplo $\varphi \vee \psi$ denota lo verdadero en función de φ y ψ. El método consistía en:

- Asumir que las fórmulas atómicas tienen una denotación fija, es decir, podemos determinar previamente si es V o F.
- Las denotaciones de $\varphi \vee \psi$, $\neg\varphi$ quedan determinadas por las tablas de verdad correspondientes y el paso anterior.
- La denotación de $\forall x.\varphi$ es V si y solo si, para cada a posible, la denotación de $\varphi[a/x]$ es V.

Con esto ya sabríamos responder a la primera pregunta que nos hicimos para la lógica de primer orden. Sin embargo, esta aproximación no es la única. Pensemos en el concepto de 'prueba' en un sentido computacional. En vez de enfocar el valor semántico hacia la denotación, preguntémonos cuándo un enunciado 'tiene una prueba'. Así obtenemos un método alternativo:

- Asumir que, para las fórmulas atómicas, conocemos lo que significa una prueba. Por ejemplo, la prueba de que $2 + 2 = 4$ consiste en operar con lápiz y papel.
- Una prueba de $\varphi \wedge \psi$ es un par (r, s) donde r es una prueba de φ y donde s es una prueba de ψ.
- Una prueba de $\varphi \vee \psi$ es un par (k, r), donde o bien $r = 0$ y r es prueba de φ o bien $k = 1$ y r es prueba de ψ.
- Una prueba de $\varphi \to \psi$ es una función que lleva pruebas de φ en pruebas de ψ.
- Una prueba de $\neg\varphi$ es una prueba de $\varphi \to \bot$, donde $\bot$ no admite prueba.
- Una prueba de $\forall x.\varphi$ es una función que lleva cada elemento posible a en una prueba de $\varphi[a/x]$.
- Una prueba de $\exists x.\varphi$ es un par (a, r) donde a es un elemento posible y r es una prueba de $\varphi[a/x]$.

¿Qué ha ocurrido aquí? Hemos dado un valor semántico diferente a la sintaxis lógica usual. Nos encontraremos con situaciones similares a ésta a lo largo del curso pero, en vez de hablar de 'proposiciones' y 'fórmulas' trataremos 'programas'.

1.2. Métodos de descripción semántica

Consideremos un programa como $\mathtt{z} := \mathtt{x}; \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{z}$. Un análisis sintáctico nos dice que tenemos tres expresiones separadas por ';' y que cada una tiene la forma de una variable separada de otra por ':='. El análisis semántico depende en gran medida del sintáctico: será entonces conveniente asumir programas que estén sintácticamente bien escritos, como en este ejemplo. La asignación de un valor semántico para tal expresión se tiene que realizar necesariamente en dos pasos:

(i) Dar un significado expresiones separadas por ';'.

(ii) Dar un significado a expresiones formadas por una variable seguida de ':=' y una expresión.

Nosotros nos centraremos en lo sucesivo en tres enfoques distintos (aunque complementarios) para dar significado a expresiones bien formadas.

1.2.1. Semántica operacional

Aquí el valor semántico recae sobre el efecto que tiene el programa sobre la máquina en la que se ejecuta, es decir, una descripción operacional os dirá cómo ejecutar el programa que presentamos antes:

(i) Para ejecutar una serie de expresiones separadas por ';', las ejecutamos una a una de izquierda a derecha.

(ii) Para ejecutar una expresión formada por una variable seguida de ':=' y una variable, determinamos el valor de la segunda variable y se lo damos a la primera.

Por tanto, si ejecutamos el programa $\mathtt{z} := \mathtt{x}; \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{z}$ suponiendo que tenemos las asignaciones iniciales $[\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 0]$, tenemos:

$$\begin{aligned}\langle \mathtt{z} := \mathtt{x}; \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{z}, [\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 0]\rangle &\to \\ \langle \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{z}, [\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 5]\rangle &\to \\ \langle \mathtt{y} := \mathtt{z}, [\mathtt{x} \mapsto 7, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 5]\rangle &\to \\ [\mathtt{x} \mapsto 7, \mathtt{y} \mapsto 5, \mathtt{z} \mapsto 5]&\end{aligned}$$

Esto es lo que se denomina *semántica operacional estructural*. Pero podemos seguir un procedimiento distinto que muestra menos pasos del proceso anterior:

$$\dfrac{\dfrac{\langle \mathtt{z} := \mathtt{x}, s_0\rangle \to s_1 \qquad \langle \mathtt{x} := \mathtt{y}, s_1\rangle \to s_2}{\langle \mathtt{z} := \mathtt{x}; \mathtt{x} := \mathtt{y}, s_0\rangle \to s_2} \qquad \langle \mathtt{y} := \mathtt{z}, s_2\rangle \to s_3}{\langle \mathtt{z} := \mathtt{x}; \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{z}, s_0\rangle \to s_3}$$

Siendo:

$$s_0 := [\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 0], s_1 := [\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 5],$$
$$s_2 := [\mathtt{x} \mapsto 7, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 5], s_3 := [\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 5, \mathtt{z} \mapsto 5].$$

Es decir, aquí hemos resumido toda la información en $\langle e, s\rangle \to t$, que simboliza el hecho de que, al ejecutar la expresión e en el estado s, pasamos al estado t.

1.2.2. Semántica denotacional

En este punto de vista, el valor semántico se encuentra en el efecto de cómo se ejecutan los programas. En el caso que tratamos:

(i) El efecto de una serie de expresiones separadas por ';' consiste en la composición de los efectos de las expresiones.

(ii) El efecto de una expresión formada por una variable seguida por ':=' y otra variable es una función que lleva un estado en uno nuevo, formado a partir del original haciendo que el valor de la primera variable sea el de la segunda.

Es decir, tenemos:

$$\mathcal{S}[\![\mathtt{z} := \mathtt{x}; \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{z}]\!] = \mathcal{S}[\![\mathtt{y} := \mathtt{z}]\!] \circ \mathcal{S}[\![\mathtt{x} := \mathtt{y}]\!] \circ \mathcal{S}[\![\mathtt{z} := \mathtt{x}]\!]$$

Y por tanto,

$$\begin{aligned}
&\mathcal{S}[\![\mathtt{z} := \mathtt{x}; \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{z}]\!]([\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 0]) \\
=&\mathcal{S}[\![\mathtt{y} := \mathtt{z}]\!](\mathcal{S}[\![\mathtt{x} := \mathtt{y}]\!](\mathcal{S}[\![\mathtt{z} := \mathtt{x}]\!]([\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 0]))) \\
=&\mathcal{S}[\![\mathtt{y} := \mathtt{z}]\!](\mathcal{S}[\![\mathtt{x} := \mathtt{y}]\!]([\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 5])) \\
=&\mathcal{S}[\![\mathtt{y} := \mathtt{z}]\!]([\mathtt{x} \mapsto 7, \mathtt{y} \mapsto 7, \mathtt{z} \mapsto 5]) \\
=&[\mathtt{x} \mapsto 7, \mathtt{y} \mapsto 5, \mathtt{z} \mapsto 5].
\end{aligned}$$

Nótese lo que hemos conseguido aquí: hemos traducido el funcionamiento del programa a una serie de objetos matemáticos. Esto será de ayuda en el futuro.

1.2.3. Semántica axiomática

Dado un programa, decimos que es *parcialmente correcto* respecto de una premisa y una consecuencia si, cuando el estado inicial verifica la premisa y el programa termina, el estado final verifica la consecuencia. En nuestro caso, tenemos la propiedad:

$$\{\mathtt{x} = \mathtt{n} \wedge \mathtt{y} = \mathtt{m}\}\mathtt{z} := \mathtt{x}; \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{z}\{\mathtt{y} = \mathtt{n} \wedge \mathtt{x} = \mathtt{m}\}$$

Nótese que esta propiedad no asegura que el programa termine. Desde este punto de vista, lo que queremos es construir un sistema lógico para demostrar la corrección parcial de un programa. En cambio, como veremos, ciertos aspectos de los programas no serán tenidos en cuenta. La deducción de la corrección parcial del programa de ejemplo es la siguiente:

$$\dfrac{\dfrac{\{p_0\}\mathtt{z} := \mathtt{x}\{p_1\} \qquad \{p_1\}\mathtt{x} := \mathtt{y}\{p_2\}}{\{p_0\}\mathtt{z} := \mathtt{x}; \mathtt{x} := \mathtt{y}\{p_2\}} \qquad \{p_2\}\mathtt{y} := \mathtt{z}\{p_3\}}{\{p_0\}\mathtt{z} := \mathtt{x}; \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{z}\{p_3\}}$$

Donde

$$p_0 := \mathtt{x} = \mathtt{n} \wedge \mathtt{y} = \mathtt{m}, p_1 := \mathtt{z} = \mathtt{n} \wedge \mathtt{y} = \mathtt{m},$$

$$p_2 := \mathtt{z} = \mathtt{n} \wedge \mathtt{x} = \mathtt{m}, p_3 := \mathtt{y} = \mathtt{n} \wedge \mathtt{x} = \mathtt{m}.$$

Aunque se pueda observar cierta similitud con el enfoque operacional, la diferencia es que aquí trabajamos con aserciones que no tienen en cuenta el funcionamiento del programa. La ventaja de esto consiste en que podemos describir fácilmente determinadas propiedades de tal programa.

1.3. El lenguaje While

Veamos a continuación un ejemplo que iremos desarrollando durante el curso, el lenguaje **While**. Para especificar las *categorías sintácticas*, especificamos una *metavariable* específica que toma valores en los elementos de cada una:

- Numerales: $n \in \mathbf{Num}$.
- Variables: $x \in \mathbf{Var}$.
- Expresiones aritméticas: $a \in \mathbf{Aexp}$.
- Expresiones booleanas: $b \in \mathbf{Bexp}$.

- Sentencias: $S \in \mathbf{Stm}$.

En caso de que hiciera falta emplear más de una metavariable, emplearemos, por ejemplo, $n', n'', \ldots$ y $n_1, n_2, \ldots$. Supondremos que las categorías **Num** y **Var** se construyen de manera natural. Las categorías sintácticas **Aexp**, **Bexp** y **Stm** se construyen, respectivamente, de la siguiente forma:

$$\begin{aligned} a &::= n \mid x \mid a_1 + a_2 \mid a_1 \times a_2 \mid a_1 - a_2 \\ b &::= \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\ S &::= x := a \mid \texttt{skip} \mid S_1; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \mid \texttt{while } b \texttt{ do } S \end{aligned}$$

Si volvemos al ejemplo de antes, la expresión $\texttt{z} := \texttt{x}; \texttt{x} := \texttt{y}; \texttt{y} := \texttt{z}$, podemos definir una *sintaxis concreta*, en el sentido de que, de los diferentes árboles de derivación para tal expresión, debemos escoger uno. En cambio, lo que acabamos de definir arriba es un ejemplo de *sintaxis abstracta*, y los árboles de derivación posibles son todos distintos elementos de la categoría **Stm**. De hecho, en caso de que queramos expresar la prioridad de las operaciones, lo haremos mediante el uso de paréntesis.

1.4. Semántica para expresiones

Veamos, a modo de ejemplo, cómo dar significado a los numerales. Expresaremos la gramática de **Num** por

$$n ::= \texttt{0} \mid \texttt{1} \mid n\ \texttt{0} \mid n\ \texttt{1}$$

Intepretaremos los numerales como si fuesen la expresión binaria de un número natural. Es decir, se tendrá una *aplicación semántica* $\mathcal{N} : \mathbf{Num} \to \mathbb{Z}$ dada recursivamente por:

$$\begin{aligned} \mathcal{N}[\![\texttt{0}]\!] &= 0 \\ \mathcal{N}[\![\texttt{1}]\!] &= 1 \\ \mathcal{N}[\![n\texttt{0}]\!] &= 2 \otimes \mathcal{N}[\![n]\!] \\ \mathcal{N}[\![n\texttt{1}]\!] &= 2 \otimes \mathcal{N}[\![n]\!] \oplus 1 \end{aligned}$$

Donde $\oplus, \otimes$ expresan las operaciones de suma y producto en $\mathbb{Z}$ y $0, 1, 2$ los enteros correspondientes: la distinción entre objetos sintácticos y semánticos tiene que ser cuidadosa. Las igualdades anteriores se llaman *ecuaciones semánticas*, nos indican cómo asociar un objeto matemático a un símbolo. Además, tenemos aquí un ejemplo de lo que se llama el *principio de composición*, es decir, construimos el significado de una expresión (*elemento compuesto*) en función del de sus componentes (*elementos base*). Ésto facilita aplicar el método de demostración general que seguiremos, la *inducción estructural*, que consiste en primero demostrar una propiedad para cada elemento base y después demostrarla para los compuestos empleando la hipótesis de inducción. Veamos un ejemplo a continuación.

Primero recordemos que una función $f : A \to B$ es *parcial* si hay elementos $a \in A$ en los que no está definida. En caso contrario, se denomina *función total*.

Proposición 1.1. *La función $\mathcal{N} : \mathbf{Num} \to \mathbb{Z}$ es total.*

Demostración. Por inducción. De los siguientes casos, $(a), (b)$ son los casos base y $(c), (d)$ los *inductivos*:

(a) Si $n = \texttt{0}$, $\mathcal{N}[\![n]\!] = 0$.

(b) Si $n = \texttt{1}$, $\mathcal{N}[\![n]\!] = 1$.

(c) Si $n = m\texttt{0}$, $\mathcal{N}[\![n]\!] = \mathcal{N}[\![m\texttt{0}]\!] = 2 \otimes \mathcal{N}[\![m]\!]$, y por hipótesis de inducción tenemos el resultado.

(d) Análogo a (c).

□

Ejemplo 1.2. Definamos una gramática y una semántica asociada para interpretar los numerales (binarios), de modo que el primer caracter de cada cadena represente el signo (positivo o negativo) del número representado por el resto de la misma. Una gramática posible es:

$$n ::= \texttt{0} \mid \texttt{1} \mid \texttt{0}n \mid \texttt{1}n$$

Definiremos su semántica atendiendo a que las cadenas `0` y `1` representan el número 0, pues se compondrían solo del signo, sin acompañar ningún número.

La función semántica es $\mathcal{S} : \mathbf{Num} \to \mathbb{Z}$, definida por:

$$\begin{aligned}
\mathcal{S}[\![\mathtt{0}]\!] &= 0 \\
\mathcal{S}[\![\mathtt{1}]\!] &= 0 \\
\mathcal{S}[\![\mathtt{0}n]\!] &= \mathcal{N}[\![n]\!] \\
\mathcal{S}[\![\mathtt{1}n]\!] &= -\mathcal{N}[\![n]\!]
\end{aligned}$$

Notemos el siguiente detalle: en principio, la función $\mathcal{N}$, como tal, no puede tomar valores del modo anterior. Sin embargo, se puede probar que, como la gramática que dimos en la definición de $\mathcal{N}$ y la que hemos escrito arriba generan las mismas cadenas, la función $\mathcal{N}$ se puede identificar fácilmente con la función que buscamos.

Pese a ello es posible definir dicha función de la siguiente forma

$$\begin{aligned}
\mathcal{N}[\![\mathtt{0}]\!] &= 0 \\
\mathcal{N}[\![\mathtt{1}]\!] &= 1 \\
\mathcal{N}[\![\mathtt{0}n]\!] &= \mathcal{AUX}[\![n, 0]\!] \\
\mathcal{N}[\![\mathtt{1}n]\!] &= \mathcal{AUX}[\![n, 1]\!]
\end{aligned}$$

y la función $\mathcal{AUX} : \mathbf{Num} \times \mathbb{N} \to \mathbb{N}$

$$\begin{aligned}
\mathcal{AUX}[\![\mathtt{0}, x]\!] &= 2 \otimes x \\
\mathcal{AUX}[\![\mathtt{1}, x]\!] &= 2 \otimes x \oplus 1 \\
\mathcal{AUX}[\![\mathtt{0}n, x]\!] &= \mathcal{AUX}[\![n, 2 \otimes x]\!] \\
\mathcal{AUX}[\![\mathtt{1}n, x]\!] &= \mathcal{AUX}[\![n, 2 \otimes x \oplus 1]\!]
\end{aligned}$$

donde el segundo parámetro representa un acumulador. Por construcción de la gramática la lectura de la cadena se hace de izquierda a derecha, dificultando un poco la construcción de la semántica.

Desde la perspectiva de la semántica denotacional, el significado de una expresión está determinado por los valores que toman en ella las variables. Esto motiva el concepto de *estado*, definido en nuestro caso como un elemento del conjunto $\mathbf{State} := \mathbf{Var} \to \mathbb{Z}$, es decir, como una función que lleva una variable en su valor (un entero positivo). Por tanto, el significado de la expresión viene dado por una función auxiliar $\mathcal{A} : \mathbf{Aexp} \to (\mathbf{State} \to \mathbb{Z})$, donde $\mathcal{A}$ toma una exprexión aritmética y un estado s[1]:

$$\begin{aligned}
\mathcal{A}[\![n]\!]s &= \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s &= s\ x \\
\mathcal{A}[\![a_1 + a_2]\!]s &= \mathcal{A}[\![a_1]\!]s \oplus \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 \times a_2]\!]s &= \mathcal{A}[\![a_1]\!]s \otimes \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s &= \mathcal{A}[\![a_1]\!]s \ominus \mathcal{A}[\![a_2]\!]s
\end{aligned}$$

Ejemplo 1.3. Podemos añadir a la definición la ecuación semántica

$$\mathcal{A}[\![-a]\!]s = 0 \ominus \mathcal{A}[\![a]\!]s$$

Incluso podemos prescindir del 0 por cómo está definida $\ominus$. En cambio,

$$\mathcal{A}[\![-a]\!]s = \mathcal{A}[\![\mathtt{0} - a]\!]s$$

no está definida de forma composicional.

[1] Nótese que $\mathcal{A}$ nos lleva a a una función $\mathcal{A}[\![a]\!]$ y aplicamos tal función a s escribiendo $\mathcal{A}[\![a]\!]s$. Por otro lado, con $s\,x$ nos referimos a s aplicado a x.

Se puede repetir el procedimiento anterior para definir una función semántica para los booleanos, $\mathcal{B} : \mathbf{Bexp} \to (\mathbf{State} \to \mathit{Bool})$, siendo $\mathit{Bool} := \{\mathbf{tt}, \mathbf{ff}\}$, definida por:

$$\mathcal{B}[\![\texttt{true}]\!]s = \mathbf{tt}$$
$$\mathcal{B}[\![\texttt{false}]\!]s = \mathbf{ff}$$
$$\mathcal{B}[\![a_1 = a_2]\!]s = \begin{cases} \mathbf{tt}, \text{ si } \mathcal{A}[\![a_1]\!]s \text{ es igual a } \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff}, \text{ en otro caso} \end{cases}$$
$$\mathcal{B}[\![a_1 \leq a_2]\!]s = \begin{cases} \mathbf{tt}, \text{ si } \mathcal{A}[\![a_1]\!]s \text{ es menor que } \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff}, \text{ en otro caso} \end{cases}$$
$$\mathcal{B}[\![\neg b]\!]s = \begin{cases} \mathbf{tt}, \text{ si } \mathcal{B}[\![b]\!]s \text{ es } \mathbf{ff} \\ \mathbf{ff}, \text{ en otro caso} \end{cases}$$
$$\mathcal{B}[\![b_1 \wedge b_2]\!]s = \begin{cases} \mathbf{tt}, \text{ si } \mathcal{B}[\![b_1]\!]s \text{ es } \mathbf{tt} \text{ y } \mathcal{B}[\![b_2]\!]s \text{ es } \mathbf{tt} \\ \mathbf{ff}, \text{ en otro caso} \end{cases}$$

De nuevo, por inducción estructural, es fácil demostrar el siguiente resultado, que es análogo al que vimos para los numerales:

Proposición 1.4. *La función $\mathcal{B} : \mathbf{Bexp} \to (\mathbf{State} \to \mathit{Bool})$ es total.*

El siguiente ejemplo ilustra cómo podemos extender una categoría sintáctica (de forma cuidadosa):

Ejemplo 1.5. Consideremos la extensión $\mathbf{Bexp}'$ de **Bexp**:

$$b ::= \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \neq a_2 \mid a_1 \leq a_2 \mid a_1 \geq a_2 \mid a_1 < a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid b_1 \Rightarrow b_2 \mid b_1 \Leftrightarrow b_2$$

Dos expresiones booleanas b_1, b_2 se dicen *equivalentes* si, para cada estado s, $\mathcal{B}[\![b_1]\!]s = \mathcal{B}[\![b_2]\!]s$. Veamos que, dada una expresión $b' \in \mathbf{Bexp}'$, existe $b \in \mathbf{Bexp}$ equivalente a b'. La demostración consiste en dos pasos: (i) Dar un valor semántico a cada expresión de la extensión, (ii) Comprobar que podemos expresar el valor semántico de b' mediante b, empleando las igualdades sintácticas naturales.

- Si b' es una expresión de **Bexp**, $b := b'$.
- Si b' es de la forma $a_1 \neq a_2$, tomamos b como $\neg(a_1 = a_2)$.
- Si b' es de la forma $a_1 \geq a_2$, tomamos b como $a_2 \leq a_1$.
- Si b' es de la forma $a_1 < a_2$, tomamos b como $(a_1 \leq a_2) \wedge \neg(a_1 = a_2)$.
- Si b' es de la forma $a_1 > a_2$, tomamos b como $(a_2 \leq a_1) \wedge \neg(a_1 = a_2)$.
- Si b' es de la forma $b_1 \vee b_2$, tomamos b como $\neg(\neg b_1 \wedge \neg b_2)$.
- Si b' es de la forma $b_1 \Rightarrow b_2$, tomamos b como $\neg(b_1 \wedge \neg b_2)$.
- Si b' es de la forma $b_1 \Leftrightarrow b_2$, tomamos b como $\neg(b_1 \wedge \neg b_2) \wedge \neg(b_2 \wedge \neg b_1)$.

Notemos que podríamos haber razonado inductivamente, pero para mayor claridad hemos indicado cuál es la traducción concreta de cada expresión.

1.5. Propiedades semánticas

En esta sección introducimos dos conceptos fundamentales que son comunes a la lógica.

1.5.1. Variables libres

Dada una expresión aritmética a, su conjunto de *variables libres*, $\mathrm{FV}(a) \subseteq \mathbf{Var}$, se define composicionalmente como:

$$\begin{aligned}
\mathrm{FV}(n) &= \emptyset \\
\mathrm{FV}(x) &= \{x\} \\
\mathrm{FV}(a_1 + a_2) &= \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2) \\
\mathrm{FV}(a_1 \times a_2) &= \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2) \\
\mathrm{FV}(a_1 - a_2) &= \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2)
\end{aligned}$$

El siguiente resultado nos dice que $\mathrm{FV}(a)$ determina el valor semántico de a:

Lema 1.6. *Sea $a \in$ **Aexp**. Sean $s, s' \in$ **State** tales que, para cada $x \in \mathrm{FV}(a)$, $s\ x = s'\ x$. Entonces $\mathcal{A}[\![a]\!]s = \mathcal{A}[\![a]\!]s'$.*

Demostración. Veamos los casos base:

- Si $a := n$, sabemos que $\mathcal{A}[\![a]\!]s := \mathcal{N}[\![n]\!] =: \mathcal{A}[\![a]\!]s'$.
- Si $a := x$, entonces, como $x \in \mathrm{FV}(a)$, por hipótesis tenemos que $\mathcal{A}[\![a]\!]s := s\ x = s'\ x := \mathcal{A}[\![n]\!]s'$.

Los casos inductivos son:

- Si a es de la forma $a_1 + a_2$, $\mathcal{A}[\![a]\!]s := \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s$ y $\mathcal{A}[\![a]\!]s' := \mathcal{A}[\![a_1]\!]s' + \mathcal{A}[\![a_2]\!]s'$. Como $\mathrm{FV}(a_i) \subseteq \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2) = \mathrm{FV}(a_1 + a_2)$, por la hipótesis de inducción aplicada a a_i, tenemos que $\mathcal{A}[\![a_i]\!]s = \mathcal{A}[\![a_i]\!]s'$, para $i = 1, 2$. Entonces,

  $$\mathcal{A}[\![a_1 + a_2]\!]s = \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s = \mathcal{A}[\![a_1]\!]s' + \mathcal{A}[\![a_2]\!]s' = \mathcal{A}[\![a_1 + a_2]\!]s',$$

  como queríamos.
- Para $a_1 * a_2$ y $a_1 - a_2$ basta repetir lo anterior (ya que el conjunto de variables libres es el mismo).

□

De la misma forma, para expresiones booleanas, tenemos:

$$\begin{aligned}
\mathrm{FV}(\texttt{true}) &= \emptyset \\
\mathrm{FV}(\texttt{true}) &= \emptyset \\
\mathrm{FV}(a_1 = a_2) &= \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2) \\
\mathrm{FV}(a_1 \leq a_2) &= \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2) \\
\mathrm{FV}(\neg b) &= \mathrm{FV}(b) \\
\mathrm{FV}(b_1 \wedge b_2) &= \mathrm{FV}(b_1) \cup \mathrm{FV}(b_2)
\end{aligned}$$

La demostración del anterior lema se puede repetir de nuevo:

Lema 1.7. *Sea $b \in$ **Bexp**. Sean $s, s' \in$ **State** tales que, para cada $x \in \mathrm{FV}(b)$, $s\ x = s'\ x$. Entonces $\mathcal{B}[\![b]\!]s = \mathcal{B}[\![b]\!]s'$.*

Demostración. Casos base:

- Si $b :=$ `true`, $\mathcal{B}[\![b]\!]s := V =: \mathcal{B}[\![b]\!]s'$ y análogamente con `false`.
- Si b es de la forma $a_1 = a_2$, con $a_1, a_2 \in$ **Aexp**, sabemos que

  $$\mathcal{B}[\![a_1 = a_2]\!]s = \begin{cases} V, \text{ si } \mathcal{A}[\![a_1]\!]s \text{ es igual a } \mathcal{A}[\![a_2]\!]s \\ F, \text{ en otro caso} \end{cases} \quad \text{y que } \mathcal{B}[\![a_1 = a_2]\!]s' = \begin{cases} V, \text{ si } \mathcal{A}[\![a_1]\!]s' \text{ es igual a } \mathcal{A}[\![a_2]\!]s' \\ F, \text{ en otro caso} \end{cases}$$

  Como suponemos que para cada $x \in \mathrm{FV}(b)$, $s\ x = s'\ x$ y $\mathrm{FV}(a_1), \mathrm{FV}(a_2) \subseteq \mathrm{FV}(b)$, se sigue que se verifican las hipótesis del lema anterior, y que por tanto $\mathcal{A}[\![a_i]\!]s = \mathcal{A}[\![a_i]\!]s'$, para $i = 1, 2$. Ahora bien, $\mathcal{B}[\![b]\!]s$ es V si y solo si $\mathcal{A}[\![a_1]\!]s$ es igual a $\mathcal{A}[\![a_2]\!]s$ y, por lo que acabamos de decir, esto es cierto si y solo si $\mathcal{A}[\![a_1]\!]s'$ es igual a $\mathcal{A}[\![a_2]\!]s'$, que es precisamente equivalente a que $\mathcal{B}[\![b]\!]s'$ sea V.

- Si b es de la forma $a_1 \leq a_2$, con con $a_1, a_2 \in \mathbf{Aexp}$, el procedimiento es análogo al anterior.

Para los casos inductivos tenemos:

- Si b es de la forma $\neg b'$, para cierta $b' \in$ Bexp, sabemos que entonces $\mathrm{FV}(b) = \mathrm{FV}(b')$. Como suponemos que, para cada $x \in \mathrm{FV}(b)$, $s\ x = s'\ x$, podemos aplicar la hipótesis de inducción, y entonces obtenemos que $\mathcal{B}[\![b]\!]s$ es V si y solo si $\mathcal{B}[\![b']\!]s = \mathcal{B}[\![b']\!]s'$ es V, que es equivalente a que $\mathcal{B}[\![b]\!]s'$ sea V. Por tanto, $\mathcal{B}[\![b]\!]s = \mathcal{B}[\![b]\!]s'$.
- Si b es de la forma $b_1 \wedge b_2$, para ciertas $b_1, b_2 \in$ Bexp, sabemos que entonces $\mathrm{FV}(b) = \mathrm{FV}(b_1) \cup \mathrm{FV}(b_2)$ y, siguiendo los razonamientos que ya hemos hecho antes, podemos aplicar la hipótesis de inducción sobre b_1 y b_2. Entonces $\mathcal{B}[\![b]\!]s$ es V si y solo si $\mathcal{B}[\![b_1]\!]s$ es V y $\mathcal{B}[\![b_2]\!]s$ es v, que equivale a que $\mathcal{B}[\![b_1]\!]s'$ sea V y $\mathcal{B}[\![b_2]\!]s'$ sea V, que es cierto si y solo si $\mathcal{B}[\![b]\!]s'$ es V y, por tanto, $\mathcal{B}[\![b]\!]s = \mathcal{B}[\![b]\!]s'$.

□

1.5.2. Sustitución

Si tenemos dos expresiones aritméticas a, a_0 y $x \in \mathrm{FV}(a)$, entonces denotamos por $a[x \mapsto a_0]$ a la expresión obtenida al *sustituir* cada ocurrencia de x en a por a_0. Se define composicionalmente como:

$$\begin{aligned}
n[x \mapsto a_0] &= n \\
y[x \mapsto a_0] &= \begin{cases} a_0, \text{ si } x = y \\ y, \text{ si } x \neq y \end{cases} \\
(a_1 + a_2)[x \mapsto a_0] &= a_1[x \mapsto a_0] + a_2[x \mapsto a_0] \\
(a_1 \times a_2)[x \mapsto a_0] &= a_1[x \mapsto a_0] \times a_2[x \mapsto a_0] \\
(a_1 - a_2)[x \mapsto a_0] &= a_1[x \mapsto a_0] - a_2[x \mapsto a_0]
\end{aligned}$$

También podemos definir la sustitución en relación con los estados:

$$(s[y \mapsto v])x := \begin{cases} v, \text{ si } x = y \\ s\ x, \text{ si } x \neq y \end{cases}$$

La relación entre ambos conceptos se muestra en el siguiente resultado:

Lema 1.8. *Dadas $a, a_0 \in$ **Aexp**, para todo $s \in$ **State** se cumple que*

$$\mathcal{A}[\![a[y \mapsto a_0]]\!]s = \mathcal{A}[\![a]\!](s[y \mapsto \mathcal{A}[\![a_0]\!]s]).$$

Demostración. De nuevo, una demostración rutinaria por inducción estructural. Los casos base son los siguientes:

- Si $a := n$, $\mathcal{A}[\![a[y \mapsto a_0]]\!]s = \mathcal{A}[\![n]\!]s = \mathcal{N}[\![n]\!] = \mathcal{A}[\![a]\!]([s \mapsto \mathcal{A}[\![a_0]\!]s])$.
- Si $a := x$, entonces

$$\mathcal{A}[\![a[y \mapsto a_0]]\!]s = \begin{cases} \mathcal{A}[\![a_0]\!]s & x = y \\ \mathcal{A}[\![x]\!]s & x \neq y \end{cases}$$

$$\mathcal{A}[\![a]\!](s[y \mapsto \mathcal{A}[\![a_0]\!]s]) = (s[y \mapsto \mathcal{A}[\![a_0]\!]s])\ x = \begin{cases} \mathcal{A}[\![a_0]\!]s & x = y \\ s\ x & x \neq y \end{cases}$$

- Si $a := a_1 + a_2$ con a_1, a_2 cumpliendo la proposición. Se tiene

$$\mathcal{A}[\![a_i[y \mapsto a_0]]\!]s = \mathcal{A}[\![a_i]\!](s[y \mapsto \mathcal{A}[\![a_0]\!]s]) = \mathcal{A}[\![a_i]\!]s'$$

para $i \in \{1, 2\}$. Se denota $s' := (s[y \mapsto \mathcal{A}[\![a_0]\!]s])$. Entonces

$$\begin{aligned}
\mathcal{A}[\![(a_1 + a_2)[y \mapsto a_0]]\!]s &= \mathcal{A}[\![a_1[y \mapsto a_0] + a_2[y \mapsto a_0]]\!]s \\
&= \mathcal{A}[\![a_1[y \mapsto a_0]]\!]s \oplus \mathcal{A}[\![a_2[y \mapsto a_0]]\!]s \\
&\overset{hip.ind.}{=} \mathcal{A}[\![a_1]\!]s' \oplus \mathcal{A}[\![a_2]\!]s' \\
&= \mathcal{A}[\![a_1 + a_2]\!]
\end{aligned}$$

- El caso $a := a_1 \times a_2$ es análogo.

□

En general, si tratamos con conjuntos de variables, necesitaremos lo siguiente. Sea $X \subseteq \mathbf{Var}$ y $s, s' \in \mathbf{State}$. Dada $\mathtt{x} \in \mathbf{Var}$, definimos la *sustitución múltiple en X* como:

$$(s'[X \mapsto s])\ x := \begin{cases} s\ x, \text{ si } x \in X \\ s'\ x, \text{ en otro caso} \end{cases}$$

Todo lo anterior justifica una noción que será importante a lo largo del curso. Dada una categoría sintáctica **Cat**, dos expresiones $b_1, b_2 \in \mathbf{Cat}$ y una función semántica $\mathcal{C} : \mathbf{Cat} \to C$, se dice que b_1, b_2 son *semánticamente equivalentes* si para todo $s \in \mathbf{State}$ se tiene

$$\mathcal{C}[\![b_1]\!]s = \mathcal{C}[\![b_2]\!]s.$$

De todos modos, veremos que esta noción depende en gran medida del conjunto de reglas que empleemos.

2 | Semántica operacional

En el anterior capítulo hemos visto cómo dar un valor semántico al lenguaje While mediante el punto de vista de la semántica denotacional. Centrémonos ahora en la semántica operacional. La distinción fundamental que hacíamos de este enfoque es la siguiente:

- Semántica operacional *natural*, que describe cómo se han obtenido los resultados generales de las ejecuciones.
- Semántica operacional *estructural*, que describe cómo se ha obtenido cada paso en la ejecución.

Para ambos tipos de semántica operacional, el valor semántico de cada expresión será especificado por un *sistema de transiciones*, compuesto de dos configuraciones distintas:

$\langle S, s\rangle$, que denota que la expresión S se ejecutará desde el estado s.

s, que denota un estado terminal. Las *configuraciones terminales* tendrán esta forma.

Finalmente, es necesaria una *relación de transición* que describa cómo tiene lugar la ejecución. La diferencia entre las dos semánticas se encuentra principalmente en ésta. De hecho, veremos que ambos tipos de semántica son, en cierto sentido, equivalentes.

2.1. Semántica operacional natural

2.1.1. Sistema de transiciones

La relación de transición $\langle S, s\rangle \to s'$ se puede leer como que, la ejecución de S desde el estado s terminará y el nuevo estado será s'. Está determinada por las siguientes reglas[1]:

Sistema (While_{ns})**.**

[ass_{ns}]

$$\frac{}{\langle x := a, s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]}$$

[skip_{ns}]

$$\frac{}{\langle \texttt{skip}, s\rangle \to s}$$

[comp_{ns}]

$$\frac{\langle S_1, s\rangle \to s' \qquad \langle S_2, s'\rangle \to s''}{\langle S_1; S_2, s\rangle \to s''}$$

[1] Nótese que las variables S_1, S_2, s, s', s'', etc. son libres.

$[\text{if}^{\text{tt}}_{\text{ns}}]$

$$\frac{\langle S_1, s\rangle \to s'}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \to s'} \quad \text{si } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$[\text{if}^{\text{ff}}_{\text{ns}}]$

$$\frac{\langle S_2, s\rangle \to s'}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \to s'} \quad \text{si } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

$[\text{while}^{\text{tt}}_{\text{ns}}]$

$$\frac{\langle S, s\rangle \to s' \qquad \langle \texttt{while } b \texttt{ do } S, s'\rangle \to s''}{\langle \texttt{while } b \texttt{ do } S, s\rangle \to s''} \quad \text{si } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$[\text{while}^{\text{ff}}_{\text{ns}}]$

$$\frac{}{\langle \texttt{while } b \texttt{ do } S, s\rangle \to s} \quad \text{si } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

Aclaremos un poco la terminología:

Definición 2.1. Una *regla* en general tiene la forma general

$$\frac{\langle S_1, s_1\rangle \to s_1' \ldots \langle S_n, s_n\rangle \to s_n'}{\langle S, s\rangle \to s'} \quad \text{si } \varphi$$

donde los términos que aparecen encima y bajo la línea son, respectivamente, las *premisas* y la *conlusión*, y donde φ es la *condición*. Cuando empleamos las reglas anteriores para obtener una transición $\langle S, s\rangle \to s'$, obtenemos un *árbol de derivación*. Una regla sin premisas se llama *axioma*.

Consideremos el problema de construir un árbol de derivación para una expresión S y un estado s. El método general consiste en partir de la 'raíz' y encontrar las 'hojas', es decir, el paso inicial consiste en buscar una regla de modo que su conclusión tenga la ejecución $\langle S, s\rangle$ en su parte izquierda. Los pasos inductivos son:

- Si la regla encontrada es un axioma, entonces podemos determinar el estado terminal y terminamos.
- Si la regla encontrada no es un axioma, entonces el siguiente paso consiste en buscar un árbol de derivación para sus premisas.

Nótese que, en cada paso, las condiciones para aplicar cada regla tienen que ser verificadas. En el futuro demostraremos algo que parece falso a primera vista: que en el lenguaje While hay a lo sumo un árbol de derivación posible para cada ejecución $\langle S, s\rangle$.

Definición 2.2. Decimos que una ejecución de la expresión S desde el estado s, $\langle S, s\rangle$, *termina* si existe un estado s' tal que $\langle S, s\rangle \to s'$. Si tal estado no existe entonces decimos que la ejecución *cicla*. Para una expresión S, decimos que *siempre termina* si $\langle S, s\rangle$ termina para cada elección de s y que *siempre cicla* si $\langle S, s\rangle$ cicla para cada elección de s.

Ejemplo 2.3. Podemos tratar de determinar si las siguientes expresiones terminan o ciclan siempre:

1. $\texttt{while}\ \neg(\texttt{x} = 1)\ \texttt{do}\ (\texttt{y} := \texttt{y} \times \texttt{x}; \texttt{x} := \texttt{x} - 1)$.
2. $\texttt{while}\ 1 \leq \texttt{x}\ \texttt{do}\ (\texttt{y} := \texttt{y} \times \texttt{x}; \texttt{x} := \texttt{x} - 1)$.
3. `while true do skip`.

La primera para si se inicializa x con un valor mayor o igual que 1 y cicla en caso contrario: nótese que si $\mathtt{x} < 1$ decrecerá continuamente, luego es imposible que la condición del bucle llegue a no cumplirse.

La segunda para siempre. Es parecida a la anterior salvo por el hecho de que la condición del bucle hace que, para $\mathtt{x} < 1$, se pare.

La tercera dejaría intacta la configuración inicial. Sin embargo, la ejecución cicla pese a no realizar ninguna acción. Esto es porque la condición del bucle es siempre cierta.

2.1.2. Propiedades

El sistema de transición nos da un entorno en el que estudiar las propiedades de las expresiones. Veamos a continuación una definición precisa de un concepto que introdujimos al final de la introducción:

Definición 2.4. Dos expresiones S_1, S_2 se dicen *semánticamente equivalentes* si para cada par $s, s' \in \mathbf{State}$,

$$\langle S_1, s\rangle \to s' \text{ si y solo si } \langle S_2, s\rangle \to s'.$$

Lema 2.5. $\mathtt{while}\ b\ \mathtt{do}\ S$ *es semánticamente equivalente a* $\mathtt{if}\ b\ \mathtt{then}\ (S; \mathtt{while}\ b\ \mathtt{do}\ S)\ \mathtt{else\ skip}$.

Demostración. Dividimos la prueba en dos implicaciones:
Parte 1. Supongamos que se cumple $\langle \mathtt{while}\ b\ \mathtt{do}\ S, s\rangle \to s''$. Entonces existe un árbol de derivación para él, T. T puede tener dos formas en función de la regla que hayamos aplicado: o bien hemos aplicado la regla o el axioma $[\text{while}^{\text{ff}}_{\text{ns}}]$. Veamos cada caso:

(a) Si hemos aplicado la regla $[\text{while}^{\text{tt}}_{\text{ns}}]$, T es de la forma:

$$[\text{while}^{\text{tt}}_{\text{ns}}]\ \frac{\overset{\cdots}{\langle S, s\rangle \to s'} \qquad \overset{\cdots}{\langle \mathtt{while}\ b\ \mathtt{do}\ S, s'\rangle \to s''}}{\langle \mathtt{while}\ \ b\ \ \mathtt{do}\ \ S, s\rangle \to s''}$$

con $\mathcal{B}[\![b]\!]s = \mathbf{tt}$. Ahora bien, notemos que:

$$[\text{comp}_{\text{ns}}]\ \frac{\overset{\cdots}{\langle S, s\rangle \to s'} \qquad \overset{\cdots}{\langle \mathtt{while}\ b\ \mathtt{do}\ S, s'\rangle \to s''}}{\langle S; \mathtt{while}\ \ b\ \ \mathtt{do}\ \ S, s\rangle \to s''}$$

Usando que $\mathcal{B}[\![b]\!]s = \mathbf{tt}$, podemos aplicar:

$$[\text{if}^{\text{ff}}_{\text{ns}}]\ \frac{\langle S; \mathtt{while}\ \ b\ \ \mathtt{do}\ \ S, s\rangle \to s''}{\langle \mathtt{if}\ b\ \mathtt{then}\ (S; \mathtt{while}\ \ b\ \ \mathtt{do}\ \ S)\ \mathtt{else\ skip}, s\rangle \to s''}$$

Y por tanto, $\langle \mathtt{if}\ b\ \mathtt{then}\ (S; \mathtt{while}\ \ b\ \ \mathtt{do}\ \ S)\ \mathtt{else\ skip}, s\rangle \to s''$.

(b) Si hemos aplicado la regla $[\text{while}^{\text{ff}}_{\text{ns}}]$, T es de la forma:

$$\frac{}{\langle \mathtt{while}\ b\ \mathtt{do}\ S, s\rangle \to s}$$

es decir, necesariamente $s = s''$ y $\mathcal{B}[\![b]\!]s = \mathbf{ff}$. Usando el axioma $[\text{skip}_{\text{ns}}]$, directamente obtenemos que

$$[\text{skip}_{\text{ns}}]\ \frac{}{\langle \mathtt{skip}, s\rangle \to s''}$$

Pero entonces,

$$[\text{if}^{\text{ff}}_{\text{ns}}]\ \frac{\langle \mathtt{while}\ b\ \mathtt{do}\ S, s\rangle \to s \qquad \langle \mathtt{skip}, s\rangle \to s''}{\langle \mathtt{if}\ b\ \mathtt{then}\ (S; \mathtt{while}\ \ b\ \ \mathtt{do}\ \ S)\ \mathtt{else\ skip}, s\rangle \to s''}$$

y por tanto obtenemos el resultado.

Parte 2. Supongamos ahora que se cumple $\langle \texttt{if } b \texttt{ then } (S; \texttt{while } b \texttt{ do } S) \texttt{ else skip}, s\rangle \to s''$. Entonces, tenemos un árbol de derivación T y, de nuevo, podemos distinguir qué forma tendrá según las reglas que hayamos aplicado:

(a) Si hemos aplicado la regla $[\text{if}^{\text{tt}}_{\text{ns}}]$, T es de la forma:

$$[\text{if}^{\text{tt}}_{\text{ns}}]\ \frac{\dfrac{\dots}{\langle S; \texttt{while } b \texttt{ do } s, s\rangle \to s''}}{\langle \texttt{if } b \texttt{ then } (S; \texttt{while } b \texttt{ do } S) \texttt{ else skip}, s\rangle \to s''}$$

y con $\mathcal{B}[\![b]\!]s = \mathbf{tt}$. Ahora bien, solo hemos podido obtener la premisa anterior mediante $[\text{comp}_{\text{ns}}]$, por tener una expresión de la forma $S_1; S_2$ en la ejecución. Entonces deducimos que T es:

$$[\text{comp}_{\text{ns}}]\ \frac{\dfrac{\dots}{\langle S, s\rangle \to s'} \qquad \dfrac{\dots}{\langle \texttt{while } b \texttt{ do } S, s'\rangle \to s''}}{\langle S; \texttt{while } b \texttt{ do } s, s\rangle \to s''}$$

Pero entonces notemos que, usando la hipótesis $\mathcal{B}[\![b]\!]s = \mathbf{tt}$:

$$[\text{while}^{\text{tt}}_{\text{ns}}]\ \frac{\dfrac{\dots}{\langle S, s\rangle \to s'} \qquad \dfrac{\dots}{\langle \texttt{while } b \texttt{ do } S, s'\rangle \to s''}}{\langle \texttt{while } b \texttt{ do } S, s\rangle \to s''}$$

y obtenemos el resultado.

(b) Si hemos usado la regla $[\text{if}^{\text{ff}}_{\text{ns}}]$, deducimos que $\mathcal{B}[\![b]\!]s = \mathbf{ff}$ y que por tanto tenemos un árbol de derivación para $\langle \texttt{skip}, s\rangle \to s''$ y, por tanto, que $s = s''$. Pero usando $[\text{while}^{\text{ff}}_{\text{ns}}]$, tenemos el resultado (el razonamiento ha sido análogo al apartado (b) de la Parte 1).

□

Ejemplo 2.6. Veamos que $S_1; (S_2; S_3)$ y $(S_1; S_2); S_3$ son semánticamente equivalentes. Si suponemos que $\langle S_1; (S_2; S_3), s\rangle \to s'$, entonces es porque en su árbol de derivación hemos empleado $[\text{comp}_{\text{ns}}]$ a las premisas $\langle S_1, s\rangle \to s''$ y $\langle S_2; S_3, s''\rangle \to s'$. A su vez, la segunda premisa proviene del mismo modo de las premisas $\langle S_2, s''\rangle \to t$ y $\langle S_3, t\rangle \to s'$. Es decir, tenemos las siguientes hojas:

(a) $\langle S_1, s\rangle \to s''$.

(b) $\langle S_2, s''\rangle \to t$.

(c) $\langle S_3, t\rangle \to s'$.

Ahora, combinando (a) y (b) con $[\text{comp}_{\text{ns}}]$, obtenemos $\langle S_1; S_2, s\rangle \to t$ y, combinando esto con (c) de la misma forma, obtenemos que $\langle (S_1; S_2); S_3, s\rangle \to s'$, como queríamos ver. La otra implicación es análoga.

Notemos, por otro lado, que en general $S_1; S_2$ y $S_2; S_1$ no son semánticamente equivalentes: si tratásemos de hacer lo mismo que antes, obtendríamos las hojas $\langle S_1, s\rangle \to s''$ y $\langle S_2, s''\rangle \to s'$ por un lado y $\langle S_2, s\rangle \to s''$ y $\langle S_1, s''\rangle \to s'$ por otro, y en general no hay forma de combinar cada par de premisas para obtener la conclusión deseada.

Ejemplo 2.7. Podemos expandir el sistema While_{ns} el siguiente modo: añadimos dos reglas que permitan dar una semántica de la expresión $\texttt{for } x := a_1 \texttt{ to } a_2 \texttt{ do } S$, es decir,

$$[\text{for}^{\text{tt}}_{\text{ns}}]\quad \frac{\langle x := a_1; S, s\rangle \to s' \qquad \langle \texttt{for x} := \texttt{x} + \texttt{1 to } a_2 \texttt{ do } S, s'\rangle \to s''}{\langle \texttt{for x} := a_1 \texttt{ to } a_2 \texttt{ do } S, s\rangle \to s''} \quad \text{si } \mathcal{B}[\![a_1 \le a_2]\!]s = \mathbf{tt}$$

$$[\text{for}^{\text{ff}}_{\text{ns}}]\quad \frac{}{\langle \texttt{for x} := a_1 \texttt{ to } a_2 \texttt{ do } S, s\rangle \to s[\texttt{x} \mapsto \mathcal{A}[\![a_1]\!]]s} \quad \text{si } \mathcal{B}[\![a_1 \le a_2]\!]s = \mathbf{ff}$$

Pero debemos tener un especial cuidado con este tipo de reglas, por ejemplo, podemos descuidar que en a_1 aparezca la variable $\texttt{y}$, a saber, que a_1 contenga $\texttt{y} + 3$, y que por otro lado en S tengamos $\texttt{y} = 5$. Del mismo modo, podríamos tener que la variable $\texttt{x}$ ya aparece del mismo modo como $\texttt{x} = 4$, por ejemplo. Si $\texttt{x}$ apareciera en a_2 entonces también tendríamos este problema.

Aunque no lo demostraremos, se puede observar que el sistema While_{ns} es *Turing-completo*, es decir, en él podemos simular cualquier computación posible en una máquina de Turing. Por tanto, se podía pensar que podemos introducir reglas para ciertas expresiones en función de su correlato en While_{ns} (que existe, por lo anterior). Sin embargo, si quisiéramos introducir `for` $x := a_1$ `to` a_2 `do` S como un bucle `while ... do ...`, acabaríamos teniendo apariciones de `while ... do ...` en las reglas asociadas a `for` $x := a_1$ `to` a_2 `do` S, lo que difiere de la semántica operacional que hemos visto hasta ahora.

Ejemplo 2.8. Podríamos extender el lenguaje While con dos reglas para la expresión `repeat` S `until` b:

$$[\text{repeat}^{\text{tt}}_{\text{ns}}] \quad \frac{\langle S, s\rangle \to s'}{\langle \texttt{repeat } S \texttt{ until } b, s\rangle \to s'} \qquad \text{si } \mathcal{B}[\![b]\!]s' = \mathbf{tt}$$

$$[\text{repeat}^{\text{ff}}_{\text{ns}}] \quad \frac{\langle S, s\rangle \to s' \qquad \langle \texttt{repeat } S \texttt{ until } b, s'\rangle \to s''}{\langle \texttt{repeat } S \texttt{ until } b, s\rangle \to s''} \qquad \text{si } \mathcal{B}[\![b]\!]s' = \mathbf{ff}$$

Proposición 2.9. *Son semánticamente equivalentes:*

- *`repeat` S `until` b.*
- *S;`if` b `then skip else` (`repeat` S `until` b).*

Demostración. Parte 1. Supongamos que $\langle \texttt{repeat } S \texttt{ until } b, s\rangle \to s'$. Solo tenemos las siguientes posibilidades:

(a) Si hemos aplicado la regla $[\text{repeat}^{\text{tt}}_{\text{ns}}]$, tenemos:

$$[\text{repeat}^{\text{tt}}_{\text{ns}}] \quad \frac{\langle S, s\rangle \to s'}{\langle \texttt{repeat } S \texttt{ until } b, s\rangle \to s'} \qquad \text{si } \mathcal{B}[\![b]\!]s' = \mathbf{tt}$$

Ahora bien, por otro lado, podemos aplicar el axioma $[\text{skip}_{\text{ns}}]$ para obtener directamente que $\langle \texttt{skip}, s'\rangle \to s'$. Ahora, como si $\mathcal{B}[\![b]\!]s' = \mathbf{tt}$, podemos aplicar la regla $[\text{if}^{\text{tt}}_{\text{ns}}]$:

$$[\text{if}^{\text{tt}}_{\text{ns}}] \ \frac{\langle \texttt{skip}, s'\rangle \to s'}{\langle \texttt{if } b \texttt{ then skip else (repeat } S \texttt{ until } b), s'\rangle \to s'}$$

Y, entonces,

$$[\text{comp}_{\text{ns}}] \ \frac{\langle S, s\rangle \to s' \qquad \langle \texttt{if } b \texttt{ then skip else (repeat } S \texttt{ until } b), s'\rangle \to s'}{\langle S; \texttt{if } b \texttt{ then skip else (repeat } S \texttt{ until } b), s\rangle \to s'}$$

Luego obtenemos el resultado.

(b) Si hemos aplicado la regla $[\text{repeat}^{\text{ff}}_{\text{ns}}]$, tenemos:

$$[\text{repeat}^{\text{ff}}_{\text{ns}}] \quad \frac{\langle S, s\rangle \to s'' \qquad \langle \texttt{repeat } S \texttt{ until } b, s''\rangle \to s'}{\langle \texttt{repeat } S \texttt{ until } b, s\rangle \to s'} \qquad \text{si } \mathcal{B}[\![b]\!]s' = \mathbf{ff}$$

Ahora, usando que si $\mathcal{B}[\![b]\!]s' = \mathbf{ff}$,

$$[\text{if}^{\text{ff}}_{\text{ns}}] \ \frac{\langle \texttt{repeat } S \texttt{ until } b, s''\rangle \to s'}{\langle \texttt{if } b \texttt{ then skip else (repeat } S \texttt{ until } b), s''\rangle \to s'}$$

Pero entonces,

$$[\text{comp}_{\text{ns}}] \ \frac{\langle S, s\rangle \to s'' \qquad \langle \texttt{if } b \texttt{ then skip else (repeat } S \texttt{ until } b), s''\rangle \to s'}{\langle S; \texttt{if } b \texttt{ then skip else (repeat } S \texttt{ until } b), s\rangle \to s'}$$

Parte 2. Supongamos que $\langle S;\ \texttt{if } b \texttt{ then skip else repeat } S \texttt{ until } b, s\rangle \to s'$. La única posibilidad es haber aplicado la regla $[\text{comp}_{\text{ns}}]$

$$\frac{\langle S, s\rangle \to s_0 \qquad \langle \texttt{if } b \texttt{ then skip else (repeat } S \texttt{ until } b), s_0\rangle \to s'}{\langle S; \texttt{if } b \texttt{ then skip else repeat } S \texttt{ until } b, s\rangle \to s'}$$

para algún $s_0 \in \mathbf{State}$. Para la transición $\langle \texttt{if } b \texttt{ then skip else } (\texttt{repeat } S \texttt{ until } b), s_0\rangle \to s'$ tenemos dos posibilidades

(a) Si $\mathcal{B}[\![b]\!]s_0 = \mathbf{tt}$ entonces únicamente existe la posibilidad de que se haya derivado de $[\text{if}^{\text{tt}}_{\text{ns}}]$:

$$\frac{\langle skip, s_0\rangle \to s'}{\langle \texttt{if } b \texttt{ then skip else } (\texttt{repeat } S \texttt{ until } b), s_0\rangle \to s'}$$

y la única forma de que sea cierto $\langle skip, s_0\rangle \to s'$ es que $s_0 = s'$. Como se verifica $\langle S, s\rangle \to s_0$ entonces se verifica $\langle S, s\rangle \to s'$ y se puede aplicar la regla $[\text{repeat}^{\text{tt}}_{\text{ns}}]$:

$$\frac{\langle S, s\rangle \to s'}{\langle \texttt{repeat } S \texttt{ until } b, s\rangle \to s'}$$

obteniendo el resultado pues ya se sabe que $\langle S, s\rangle \to s_0$.

(b) Si $\mathcal{B}[\![b]\!]s_0 = \mathbf{ff}$ entonces solo cabe la posibilidad de que haya partido de $[\text{if}^{\text{ff}}_{\text{ns}}]$:

$$\frac{\langle \texttt{repeat } S \texttt{ until } b, s_0\rangle \to s'}{\langle \texttt{if } b \texttt{ then skip else } (\texttt{repeat } S \texttt{ until } b), s_0\rangle \to s'}$$

teniendo así $\langle \texttt{repeat } S \texttt{ until } b, s_0\rangle \to s'$ y entonces se puede deducir

$$\frac{\langle S, s\rangle \to s_0 \qquad \langle \texttt{repeat } S \texttt{ until } b, s_0\rangle \to s'}{\langle \texttt{repeat } S \texttt{ until } b, s\rangle \to s'}$$

mediante $[\text{repeat}^{\text{ff}}_{\text{ns}}]$ pues $\mathcal{B}[\![b]\!]s_0 = \mathbf{ff}$.

□

Para poder demostrar que una propiedad como la anterior se verifica en árboles sencillos y compuestos, emplearemos la demostración por *inducción sobre reglas*, que se compone de dos pasos:

1. Primero comprobamos que la propiedad se verifica para los axiomas del sistema.
2. Para cada regla, suponiendo que las premisas verifican la propiedad , comprobamos que también se cumple para la conclusión (siempre y cuando se verifiquen las condiciones de la regla).

El siguiente resultado nos dice que, en general, hay *una* manera de deducir una configuración mediante las reglas del sistema de transición While_{ns}:

Teorema 2.10. *El sistema de transiciones* While_{ns} *es determinista, es decir, para cada* $S \in \mathbf{Stm}$, $s, s', s'' \in \mathbf{State}$,

$$\langle S, s\rangle \to s' \text{ y } \langle S, s\rangle \to s'' \text{ implica que } s' = s''.$$

Demostración. Para simplificar la demostración, vamos a definir una propiedad sintáctica de las reglas del sistema While_{ns}. Decimos que dos reglas son *independientes entre sí* cuando no es posible obtener una mediante la aplicación de la otra. Notemos que este es el caso de nuestro sistema: las reglas $[\text{while}^{\text{tt}}_{\text{ns}}]$ y $[\text{while}^{\text{ff}}_{\text{ns}}]$ son independientes entre sí porque ambas tienen premisas distintas (suponemos que $\mathbf{tt}$ y $\mathbf{ff}$ son distintos). Entonces, como cada regla es independiente de la otra (y evidentemente, cada regla es determinista), deducimos que, en caso de que tengamos $\langle S, s\rangle \to s'$ y $\langle S, s\rangle \to s''$, necesariamente tendremos que haber aplicado la misma única regla posible en los dos casos para llegar a las respectivas configuraciones. Es fácil convencerse entonces de que, por inducción sobre las reglas, la propiedad deseada se cumple[2]. □

Ejemplo 2.11. Podemos añadir una semántica $\texttt{forVar } x \texttt{ do } S$ que ejecute la sentencia S siempre que x sea distinto de 0 y lo incremente en 1 en cada iteración. Veamos que sería semánticamente equivalente a $\texttt{while } \neg(x = 0) \texttt{ do } (S; x := x + 1)$.

Primero, definimos la semantica de $\texttt{forVar } x \texttt{ do } S$:

[2]Obviamente, esa demostración no es intercambiable con la demostración formal por inducción estructural. Podría demostrarse el caso general del que hemos hablado, a saber, formalizando lo que significa precisamente la independencia de dos reglas.

$$[\text{for}^0]\ \frac{}{\langle \texttt{forVar}\ x\ \texttt{do}\ S, s\rangle \to s}\ \text{si } \mathcal{A}[\![x]\!]s = 0$$

$$[\text{for}^{\neq 0}]\ \frac{\langle S; x := x+1, s\rangle \to s' \qquad \langle \texttt{forVar}\ x\ \texttt{do}\ S, s'\rangle \to s_1}{\langle \texttt{forVar}\ x\ \texttt{do}\ S, s\rangle \to s_1}\ \text{si } \mathcal{A}[\![x]\!]s \neq 0$$

Veamos que $\langle \texttt{forVar}\ x\ \texttt{do}\ S, s\rangle \to s_1$ implica $\langle \texttt{while}\ \neg(x = 0)\ \texttt{do}\ (S; x := x + 1), s\rangle \to s_1$.
Para empezar, sabemos que $\mathcal{A}[\![x]\!]s = 0$ si y solo si $\mathcal{B}[\![\neg(x = 0)]\!]s = \mathbf{ff}$, dividimos la demostración en dos pasos:

1. Si $x = 0$ tenemos por $[\text{for}^0]$ que:

   $$\langle \texttt{forVar}\ x\ \texttt{do}\ S, s\rangle \to s$$

   Como $\mathcal{B}[\![\neg(x = 0)]\!]s = \mathbf{ff}$ por la regla $[\text{while}^{\text{ff}}_{\text{ns}}]$ sabemos que:

   $$\langle \texttt{while}\ \neg(x = 0)\ \texttt{do}\ (S; x := x + 1), s\rangle \to s$$

2. Si $x \neq 0$, entonces suponemos ciertas las siguientes premisas:

   a) $\langle S; x := x + 1, s\rangle \to s_2$
   b) $\langle \texttt{forVar}\ x\ \texttt{do}\ S, s_2\rangle \to s_1$

   pues la transición $\langle \texttt{forVar}\ x\ \texttt{do}\ S, s\rangle \to s'$ solo puede haber provenido de:

   $$[\text{for}^{\neq 0}]\ \frac{\langle S; x := x+1, s\rangle \to s_2 \qquad \langle \texttt{forVar}\ x\ \texttt{do}\ S, s_2\rangle \to s'}{\langle \texttt{forVar}\ x\ \texttt{do}\ S, s\rangle \to s'}$$

   Podemos aplicar la hipótesis de inducción sobre $\langle \texttt{forVar}\ x\ \texttt{do}\ S, s_2\rangle \to s'$ y por lo tanto tenemos que $\langle \texttt{forVar}\ x\ \texttt{do}\ S, s_2\rangle \to s'$ implica que $\langle \texttt{while}\ \neg(x = 0)\ \texttt{do}\ (S; x := x + 1), s_2\rangle \to s'$, luego podemos costruir el siguiente árbol de derivación:

   $$[\text{while}^{\text{tt}}_{\text{ns}}]\ \frac{\langle S; x := x+1, s\rangle \to s_2 \qquad \langle \texttt{while}\ \neg(x = 0)\ \texttt{do}\ (S; x := x + 1), s_2\rangle \to s'}{\langle \texttt{while}\ \neg(x = 0)\ \texttt{do}\ (S; x := x + 1), s\rangle \to s'}$$

Supongamos ahora que $\langle \texttt{while}\ \neg(x = 0)\ \texttt{do}\ (S; x := x + 1), s\rangle \to s'$. Entonces, distinguimos los siguientes casos:

1. Si hemos aplicado $[\text{while}^{\text{ff}}_{\text{ns}}]$, entonces

   $$[\text{while}^{\text{ff}}_{\text{ns}}]\ \frac{}{\langle \texttt{while}\ \neg(x = 0)\ \texttt{do}\ (S; x := x + 1), s\rangle \to s'}$$

   y además deducimos que $s = s'$ y que $x \neq 0$. Pero entonces tenemos que, directamente:

   $$[\text{for}^0]\ \frac{}{\langle \texttt{forVar}\ x\ \texttt{do}\ S, s\rangle \to s}$$

   es decir, obtenemos la implicación deseada.

2. Si hemos aplicado $[\text{while}^{\text{tt}}_{\text{ns}}]$,

   $$[\text{while}^{\text{tt}}_{\text{ns}}]\ \frac{\langle (S; x := x+1), s\rangle \to s' \qquad \langle \texttt{while}\ \neg(x = 0)\ \texttt{do}\ (S; x := x + 1), s'\rangle \to s''}{\langle \texttt{while}\ \neg(x = 0)\ \texttt{do}\ (S; x := x + 1), s\rangle \to s''}$$

   y además deducimos que $x \neq 0$. Si aplicamos hipótesis de inducción sobre $\langle \texttt{while}\ \neg(x = 0)\ \texttt{do}\ (S; x := x + 1), s'\rangle \to s''$, obtenemos que $\langle \texttt{forVar}\ x\ \texttt{do}\ S, s'\rangle \to s''$. Pero entonces, juntando las premisas anteriores,

   $$[\text{for}^{\neq 0}]\ \frac{\langle S; x := x+1, s\rangle \to s' \qquad \langle \texttt{forVar}\ x\ \texttt{do}\ S, s'\rangle \to s''}{\langle \texttt{forVar}\ x\ \texttt{do}\ S, s\rangle \to s''}$$

   luego obtenemos el resultado.

2.1.3. Expresiones

Finalmente, podemos definir el valor semántico de cada $S \in \mathbf{Stm}$ mediante una aplicación $\mathcal{S}_{\text{ns}} : \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$, donde

$$\mathcal{S}_{\text{ns}}[[S]] : \mathbf{State} \hookrightarrow \mathbf{State}$$
$$s \mapsto \begin{cases} s', \text{ si } \langle S, s\rangle \to s' \\ \text{indefinido, en otro caso} \end{cases}$$

El determinismo de While_{ns} implica que está bien definida. Además, es parcial porque, como vimos, la expresión `while true do skip` siempre entra en bucle, es decir, $\mathcal{S}_{\text{ns}}[[\texttt{while true do skip}]]s = \text{indefinido}$, para cada $s \in \mathbf{State}$.

Ejemplo 2.12. Podemos definir, por ejemplo, una semántica de paso largo para **Aexp** mediante la relación de transición $\langle a, s\rangle \to_A z$, donde $\langle a, s\rangle$ significa que $a \in \mathbf{Aexp}$ se evalúa en $s \in \mathbf{State}$ y $z \in \mathbb{Z}$ es un estado final:

Sistema $(\mathbf{Aexp}_{\text{ns}})$**.**

$$\frac{}{\langle n, s\rangle \to_A \mathcal{N}[\![n]\!]} \text{ si } n \in \mathbf{Num}$$

$$\frac{}{\langle x, s\rangle \to_A sx} \text{ si } x \in \mathbf{Var}$$

$$\frac{\langle a_1, s\rangle \to_A z_1 \qquad \langle a_2, s\rangle \to_A z_2}{\langle a_1 \ \texttt{op} \ sa_2, s\rangle \to_A z_1 * z_2}$$

donde `op` se refiere a $\oplus, \ominus, \otimes$ y $*$ a $+, -, \times$.

Proposición 2.13. *Sean $a \in \mathbf{Aexp}$, $s \in \mathbf{State}$, $z \in \mathbb{Z}$. Entonces $\langle a, s\rangle \to_A z$ si y solo si $\mathcal{A}[\![a]\!]s = z$.*

Demostración. La demostración es por inducción estructural. Los casos base son:

- Si $a = n$, entonces $\langle a, s\rangle \to_A \mathcal{N}[\![n]\!] = \mathcal{A}[\![n]\!]s$.
- Si $a = x$, entonces $\langle a, s\rangle \to_A sx = \mathcal{A}[\![x]\!]s$.

Resumimos los casos inductivos en:

- Si $a = a_1 \ \texttt{op} \ a_2$, entonces, empleando la última regla de $\mathbf{Aexp}_{\text{ns}}$, tenemos que $\langle a_i, s\rangle \to_A z_i$ si y solo si (por hipótesis de inducción) $\mathcal{A}[\![a_i]\!]s = z_i$, con $i = 1, 2$. Pero entonces sabemos que $\mathcal{A}[\![a_1 \ \texttt{op} \ a_2]\!]S = \mathcal{A}[\![a_1]\!]s * \mathcal{A}[\![a_2]\!]s = z_1 * z_2 = z$.

□

Ejemplo 2.14. Siguiendo el ejemplo anterior, podemos definir un sistema de transiciones para expresiones booleanas como sigue. Definimos $\langle b, s\rangle \to_B X$, donde $\langle b, s\rangle$ indica que b se evalúa en el estado s y donde $X \in \mathit{Bool}$.

Sistema $(\mathbf{Bexp}_{\text{ns}})$**.**

$$\frac{}{\langle \texttt{true}, s\rangle \to_B \mathbf{tt}}$$

$$\frac{}{\langle \texttt{false}, s\rangle \to_B \mathbf{ff}}$$

$$\frac{\langle a_1, s\rangle \to_A z_1 \qquad \langle a_2, s\rangle \to_A z_2}{\langle a_1 = a_2, s\rangle \to_A X}$$

donde X es el booleano correspondiente (véase la definición de $\mathcal{B}[\![\cdot]\!]$).

$$\frac{\langle a_1, s\rangle \to_A z_1 \qquad \langle a_2, s\rangle \to_A z_2}{\langle a_1 \leq sa_2, s\rangle \to_A X}$$

donde X es el booleano correspondiente (de nuevo, empleando la definición de $\mathcal{B}[\![\cdot]\!]$).

$$\frac{\langle a_1, s\rangle \to_A z_1 \qquad \langle a_2, s\rangle \to_A z_2}{\langle a_1 \leq sa_2, s\rangle \to_A X}$$

donde X es el booleano correspondiente.

$$\frac{\langle b, s\rangle \to_B X}{\langle \neg b, s\rangle \to_B X'}$$

donde X' es el booleano correspondiente (negación de X).

$$\frac{\langle b_1, s\rangle \to_B X \qquad \langle b_2, s\rangle \to_B Y}{\langle b_1 \wedge b_2, s\rangle \to_B Z}$$

donde Z es el booleano correspondiente (conjunción de X e Y).

El siguiente resultado es análogo al último que dimos antes:

Proposición 2.15. *Sean $b \in$ **Bexp**, $s \in$ **State** y $X \in Bool$. Entonces $\langle b, s\rangle \to_B X$ si y solo si $\mathcal{B}[\![b]\!]s = X$.*

Demostración. Por inducción estructural. Véase la demostración del anterior teorema y la definición de $\mathcal{B}[\![\cdot]\!]$. La demostración es análoga. □

2.2. Semántica operacional estructural

2.2.1. Sistema de transiciones

Ahora nos centramos en los pasos concretos de la ejecución de un programa. Para ello, definimos una relación de transición $\langle S, s\rangle \Rightarrow \gamma$ como:

- Si γ es de la forma $\langle S', s'\rangle$, entonces la ejecución de S desde s no se completa y sigue en $\langle S', s'\rangle$.
- Si γ es de la forma s', entonces la ejecución finaliza en el estado s'.

La nueva relación de transición queda determinada por el conjunto de reglas:

Sistema ($\text{While}_{\text{sos}}$)**.**

[ass_{sos}]

$$\frac{}{\langle x := a, s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]}$$

[skip_{sos}]

$$\frac{}{\langle \texttt{skip}, s\rangle \Rightarrow s}$$

[$\text{comp}^1_{\text{sos}}$]

$$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s'\rangle}$$

[$\text{comp}^2_{\text{sos}}$]

$$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

[$\text{if}^{\text{tt}}_{\text{sos}}$]

$$\frac{}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle} \text{ si } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

[$\text{if}^{\text{ff}}_{\text{sos}}$]

$$\frac{}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2; s, s\rangle \Rightarrow \langle S_2, s\rangle} \text{ si } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

[$\text{while}_{\text{sos}}$]

$$\frac{}{\langle \texttt{while } b \texttt{ do } S, s\rangle \Rightarrow \langle \texttt{if } b \texttt{ then } (S; \texttt{while } b \texttt{ do } S) \texttt{ else skip}, s\rangle}$$

Notemos que podríamos haber incluido, por ejemplo, dos reglas para la semántica de `while` b `do` S:

$$[\text{while}^{\text{ff}}_{\text{sos}}] \ \frac{}{\langle \texttt{while } b \texttt{ do } S, s\rangle \Rightarrow s} \text{ si } \mathcal{B}[\![s]\!] = \mathbf{ff}$$

y

$$[\text{while}^{\text{tt}}_{\text{sos}}] \ \frac{}{\langle \texttt{while } b \texttt{ do } S, s\rangle \Rightarrow \langle S; \texttt{while } b \texttt{ do } S, s\rangle} \text{ si } \mathcal{B}[\![s]\!] = \mathbf{tt}$$

Definición 2.16. Se dirá que $\langle S, s\rangle$ está *bloqueada* si no existe γ tal que $\langle S, s\rangle \Rightarrow \gamma$. Una secuencia de derivación es finita cuando llega a un bloqueo o a un estado final:

$$\gamma_0 \Rightarrow \gamma_1 \Rightarrow ... \Rightarrow \gamma_k$$

donde $\gamma_0 = \langle S, s\rangle$, $\gamma_i \Rightarrow \gamma_{i+1}$ para $i \in \{0, ..., k-1\}$ y γ_k es una configuración bloqueada.

Normalmente escribiremos $\gamma_0 \Rightarrow^i \gamma$ si hay i pasos en la ejecución de γ_0 a γ. Si hay finitos pasos, denotamos $\gamma_0 \Rightarrow^* \gamma$. $\gamma_0 \Rightarrow^i \gamma$ y $\gamma_0 \Rightarrow^* \gamma$ no tiene por qué ser secuencias de derivación, solo si γ es configuración final o de bloqueo.

Definición 2.17. La ejecución $\langle S, s\rangle$ de la expresión S en un estado s:

1. *Termina* si existe una única secuencia de derivación finita comenzando en $\langle S, s\rangle$.
2. *Termina con éxito* si $\langle S_1, s\rangle \Rightarrow^* s'$ para algún estado s'.
3. *Cicla* si existe una secuencia de derivación infinita comenzando en $\langle S, s\rangle$.

Nótese que estas definiciones son mutuamente excluyentes si y solo si las secuencias de derivacion son únicas. Por comodidad, las definimos de este modo porque, si extendemos el lenguaje, no nos tendremos que preocupar.

Ejemplo 2.18. Supongamos que queremos extender $\text{While}_{\text{sos}}$ con la expresión `repeat` S `until` b. Podemos añadir la regla:

$$[\text{repeat}_{\text{sos}}] \ \frac{}{\langle \texttt{repeat } S \texttt{ until } b, s\rangle \Rightarrow \langle S; \texttt{if } b \texttt{ then skip else } (\texttt{repeat } S \texttt{ until } b), s\rangle}$$

La idea es que la expresión `repeat` S `until` b sea equivalente a S; `while` $\neg b$ `do` S. Se definirá posteriormente el concepto de equivalencia semántica y se demostrará este resultado.

2.2.2. Propiedades

El método de demostración principal consiste en hacer *inducción sobre la longitud de las secuencias de derivación* (finitas) que se estudian, es decir, si queremos demostrar una propiedad acerca de nuestro sistema de transiciones:

- Demostramos que la propiedad se cumple para secuencias de derivación de longitud 0 (en ocasiones nos encontraremos que se cumple la propiedad de forma vacía).
- Demostramos que si la propiedad se cumple para secuencias de longitud (a lo sumo) k, entonces se cumple para secuencias de longitud $k+1$.

A modo de ejemplo de este método, veamos el siguiente resultado:

Lema 2.19. *Si* $\langle S_1; S_2, s\rangle \Rightarrow^k s''$*, entonces existen* $s' \in \mathbf{State}$*,* $k_1, k_2 \in \mathbb{N}$ *tales que* $k = k_1 + k_2$ *y*

$$\langle S_1, s\rangle \Rightarrow^{k_1} s' \quad y \quad \langle S_2, s'\rangle \Rightarrow^{k_2} s''.$$

Demostración. Si $k = 0$, entonces $\langle S_1; S_2, s\rangle \Rightarrow^0 s''$ implica (vacuamente) el resultado, porque $\langle S_1; S_2, s\rangle$ y s'' son distintos. Supongamos que el resultado se cumple para longitudes menores o iguales que k. Veamos que se sigue para $k+1$. Por tanto, tenemos la premisa $\langle S_1; S_2, s\rangle \Rightarrow^{k+1} s''$, es decir, que existe una configuración γ tal que

$$\langle S_1; S_2, s\rangle \Rightarrow \gamma \Rightarrow^k s''$$

Por tanto, distinguimos dos casos según la regla que hemos aplicado a $\langle S_1; S_2, s\rangle$ para llegar a γ:

(a) Si hemos aplicado $[\mathrm{comp}^1_{\mathrm{sos}}]$, tenemos que

$$[\mathrm{comp}^1_{\mathrm{sos}}]\ \frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s'\rangle = \gamma}$$

luego $\langle S_1'; S_2, s'\rangle \Rightarrow^k s''$. Entonces, como esta derivación es de longitud k, podemos aplicar hipótesis de inducción, esto es, existen $s_0 \in \mathbf{State}$ y $k_1, k_2 \in \mathbb{N}$ con $k = k_1 + k_2$ y

$$\langle S_1', s'\rangle \Rightarrow^{k_1} s_0 \quad \text{y} \quad \langle S_2, s_0\rangle \Rightarrow^{k_2} s''.$$

Ahora bien, como tenemos la premisa $\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle$ y $\langle S_1', s'\rangle \Rightarrow^{k_1} s_0$, entonces tenemos que $\langle S_1, s\rangle \Rightarrow^{k_1+1} s_0$. Por otro lado, también tenemos $\langle S_2, s_0\rangle \Rightarrow^{k_2} s''$ y que $(k_1+1)+k_2 = (k_1+k_2)+1 = k+1$. Es decir, hemos obtenido la conclusión deseada. Por tanto, hemos probado el resultado para este caso.

(b) Si hemos aplicado $[\mathrm{comp}^2_{\mathrm{sos}}]$, tenemos que

$$[\mathrm{comp}^2_{\mathrm{sos}}]\ \frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle = \gamma}$$

Entonces deducimos que $\langle S_2, s'\rangle \Rightarrow^k s''$. Simplemente tomando $k_1 := 1$ y $k_2 := k$ vemos que $k_1 + k_2 = k+1$ y que tenemos el resultado.

□

Ejemplo 2.20. Por otro lado, $\langle S_1; S_2, s\rangle \Rightarrow^* \langle S_2, s'\rangle$ no implica necesariamente que $\langle S_1, s\rangle \Rightarrow^* s'$. Por ejemplo, podemos tomar $S_1 := \texttt{skip}$, $S_2 := \texttt{while}\ \neg(\texttt{x} = 1)\ \texttt{do}\ \texttt{x} := \texttt{x} + 1$ y $s\texttt{x} = 3$, $s'\texttt{x} = s[\texttt{x} \mapsto 2]$.

El siguiente lema viene a decir que la ejecución de una expresión es independiente de cualquier enunciado que se ejecute después:

Lema 2.21. *Si* $\langle S_1, s\rangle \Rightarrow^k s'$*, entonces* $\langle S_1; S_2, s\rangle \Rightarrow^k \langle S_2, s'\rangle$.

Demostración. Por inducción sobre la longitud de las derivaciones. En caso de $k = 0$, la premisa es falsa y el resultado se tiene directamente. Supongamos que se cumple el resultado para longitudes $\leq k$ y veámoslo para $k+1$. Nuestra suposición es que $\langle S_1, s\rangle \Rightarrow^{k+1} s'$. Entonces tenemos que hay cierta configuración γ con

$$\langle S_1, s\rangle \Rightarrow \gamma \Rightarrow^{k} s'$$

y además, notemos que $\gamma = \langle S, s''\rangle$ porque $k \leq 1$. Pero entonces, aplicando la hipótesis de inducción a $\langle S, s''\rangle \Rightarrow^{k} s'$, tenemos que $\langle S; S_2, s''\rangle \Rightarrow^{k} \langle S_2, s'\rangle$.

Por otro lado, de $\langle S_1, s\rangle \Rightarrow \langle S, s''\rangle$ podemos deducir que:

$$[\text{comp}^1_{\text{sos}}]\ \frac{\langle S_1, s\rangle \Rightarrow \langle S, s''\rangle}{\langle S_1; S_2, s\rangle \Rightarrow \langle S; S_2, s''\rangle}$$

Es decir, sabemos que $\langle S_1; S_2, s\rangle \Rightarrow \langle S; S_2, s''\rangle$ y que $\langle S; S_2, s''\rangle \Rightarrow^{k} \langle S_2, s'\rangle$. Basta componer ambas derivaciones para ver que $\langle S_1; S_2, s\rangle \Rightarrow^{k+1} \langle S_2, s'\rangle$, como queríamos. □

Teorema 2.22. *El sistema de transiciones* $\text{While}_{\text{sos}}$ *es determinista, es decir, para cualesquiera* S, s, γ, γ' *tenemos que*

$$\langle S, s\rangle \Rightarrow \gamma \ \textit{y}\ \langle S, s\rangle \Rightarrow \gamma' \ \textit{implica que}\ \gamma = \gamma'$$

Demostración. Véase la demostración del Teorema 2.10. □

Definición 2.23. Dos expresiones S_1, S_2 se dicen *semánticamente equivalentes* si, para cada $s \in$ **State**,

- Si γ es estado final o bloqueado, entonces $\langle S_1, s\rangle \Rightarrow^* \gamma$ si y solo si $\langle S_2, s\rangle \Rightarrow^* \gamma$. Nótese que las longitudes de las derivaciones no tienen por qué coincidir.
- La[3] secuencia de derivación empezando en $\langle S_1, s\rangle$ es infinita si y solo si lo es la que empieza en $\langle S_2, s\rangle$.

Ejemplo 2.24. Veamos que S y $S;\texttt{skip}$ son semánticamente equivalentes. Supongamos que $\langle S, s\rangle \Rightarrow^* s'$, es decir, existe $k \in \mathbb{N}$ tal que $\langle S, s\rangle \Rightarrow^* s'$. Entonces, el Lema 2.21 nos dice que dado $\langle S, s\rangle \Rightarrow^{k} s'$ se tiene

$$\langle S; \texttt{skip}, s\rangle \Rightarrow^{k} \langle \texttt{skip}, s'\rangle$$

Por la regla $[\text{skip}_{\text{sos}}]$ se deduce $\langle \texttt{skip}, s'\rangle \Rightarrow s'$ y entonces

$$\langle S; \texttt{skip}, s\rangle \Rightarrow^* s'$$

Supongamos ahora que $\langle S; \texttt{skip}, s\rangle \Rightarrow^* s''$, entonces existe $k \in \mathbb{N}$ tal que $\langle S; \texttt{skip}, s\rangle \Rightarrow^{k} s''$. Por el Lema 2.19 se tiene la existencia de $s' \in$ **State** y $k_1, k_2 \in \mathbb{N}$ tal que $k = k_1 + k_2$ y

$$\langle S, s\rangle \Rightarrow^{k_1} s' \quad y \quad \langle \texttt{skip}, s'\rangle \Rightarrow^{k_2} s''$$

La secuencia $\langle \texttt{skip}, s'\rangle \Rightarrow^{k_2} s''$ es cierta si y solo si $k_2 = 1$ y $s' = s''$ pues solo se puede aplicar $[\text{skip}_{\text{sos}}]$, se deduce entonces

$$\langle S, s\rangle \Rightarrow^{k_1} s''$$

es decir, $\langle S, s\rangle \Rightarrow^* s''$.

Ejemplo 2.25. [HACER ESTO]: Demostrar equivalencia entre `repeat` S `until` b y a S; `while` $\neg b$ `do` S

[3] La unicidad viene dada por el determinismo de $\text{While}_{\text{sos}}$.

2.2.3. Expresiones

Análogamente a como hicimos en la semántica de paso largo, podemos definir el valor semántico de las expresiones mediante una función parcial $\mathcal{S}_{\text{sos}} : \textbf{Stm} \to (\textbf{State} \hookrightarrow \textbf{State})$, donde

$$\mathcal{S}_{\text{sos}}[[S]] : \textbf{State} \hookrightarrow \textbf{State}$$
$$s \mapsto \begin{cases} s', \text{ si } \langle S, s\rangle \Rightarrow^* s' \\ \texttt{indefinido}, \text{ en otro caso} \end{cases}$$

Notemos que esta función está bien definida precisamente por el determinismo que vimos en el anterior apartado. La ejecución de $\langle S, s\rangle$ para una expresión S dada una configuración inicial $s \in \textbf{State}$ puede dar lugar a tres casos

- Que termine, y entonces existe $s' \in \textbf{State}$ tal que

  $$s \in \textbf{State} \Rightarrow^* s'$$

  y entonces $\mathcal{S}_{\text{sos}}[[S]]s = s'$
- Que se quede bloqueada, y entonces no queda otra que $\mathcal{S}_{\text{sos}}[[S]]s = \texttt{indefinido}$.
- Que cicle, ocurriendo nuevamente que $\mathcal{S}_{\text{sos}}[[S]]s = \texttt{indefinido}$.

La equivalencia semántica coincide $\mathcal{S}_{\text{sos}}$ en el sentido de la siguiente proposición:

Proposición 2.26. *Si S_1 y S_2 son semánticamente equivalente entonces*

$$\mathcal{S}_{\text{sos}}[[S_1]] = \mathcal{S}_{\text{sos}}[[S_2]]$$

Demostración. Hay que probarlo particularizando en cada $s \in \textbf{State}$

Supongamos que S_1 y S_2 son semánticamente equivalentes. Se distinguen dos casos

- Existe un estado final $s' \in \textbf{State}$ tal que

  $$\langle S_1, s\rangle \Rightarrow^* s' \quad y \quad \langle S_2, s\rangle \Rightarrow^* s'$$

  y trivialmente

  $$\mathcal{S}_{\text{sos}}[[S_1]]s = s' = \mathcal{S}_{\text{sos}}[[S_2]]s$$
- Ambas secuencias son infinitas, es decir

  $$\langle S_1, s\rangle \text{ es infinita si y solo si } \langle S_2, s\rangle \text{ es infinita}$$

  y entonces

  $$\mathcal{S}_{\text{sos}}[[S_1]]s = \texttt{indefinido} = \mathcal{S}_{\text{sos}}[[S_2]]s$$

□

La implicación contraria no es cierta, pues si S_1 es una sentencia que cicla y S_2 es una sentencia que se bloquea, $\mathcal{S}_{\text{sos}}[[S_1]] = \texttt{indefinido} = \mathcal{S}_{\text{sos}}[[S_2]]$ pero S_1 y S_2 no son semánticamente equivalentes.

2.3. Teorema de equivalencia

Hasta ahora hemos presentado por separado los dos sistemas de transiciones para While, y hemos visto que tienen comportamientos, en general, distintos. Sin embargo, si vemos la semántica de paso largo como una extensión de la de paso corto, podemos convencernos que son esencialmente los mismo. De hecho, tienen el mismo poder expresivo. Más precisamente, el resultado clave de esta sección es:

Teorema 2.27. *Consideremos el lenguaje* While. *Para cada* $S \in \mathbf{Stm}$, $\mathcal{S}_{\text{ns}}[[S]] = \mathcal{S}_{\text{sos}}[[S]]$.

Demostración. La demostración se divide en dos pasos, cada uno consiste en demostrar uno de los siguientes lemas:

- Para cada $S \in \mathbf{Stm}$, dados $s, s' \in \mathbf{State}$, $\langle S, s\rangle \to s'$ implica que $\langle S, s\rangle \Rightarrow^* s'$.

  Se demuestra por inducción en los árboles de derivación.

- Para cada $S \in \mathbf{Stm}$, dados $s, s' \in \mathbf{State}$ y $k \in \mathbb{N}$, $\langle S, s\rangle \Rightarrow^k s'$ implica que $\langle S, s\rangle \to s'$.

  Se demuestra por inducción en la longitud de las secuencias de derivación.

Entonces, dados $S \in \mathbf{Stm}$, $s \in \mathbf{State}$, por los dos lemas se tiene que $\mathcal{S}_{\text{ns}}[[S]] = s'$ si y solo si $\mathcal{S}_{\text{sos}}[[S]] = s'$. Pero entonces obtenemos que, si una de las dos funciones está definida en un estado o no, la otra lo estará también y coincidirá con ella o no estará definida, respectivamente. Es decir, ambas funciones coinciden. □

3 | Más semántica operacional

La elección de una semántica operacional u otra para el lenguaje While es, por el Teorema de equivalencia, arbitraria. En cambio, para otros lenguajes puede ocurrir que una aproximación sea natural y la otra completamente impracticable. En esta sección estudiaremos una serie de conceptos que mostrarán la debilidad de las semánticas que hemos visto hasta ahora.

3.1. Construcciones no secuenciales

3.1.1. `abort`

La intención que tenemos al introducir la expresión `abort` es la siguiente: queremos que, cuando se ejecute, pare la ejecución de todo el programa. Notemos que no podríamos, a simple vista, construir una expresión con el mismo comportamiento en While. A saber, `while true do skip` solo consigue hacer un bucle, y `skip` permite que un programa se pueda ejecutar más tarde. Entonces, la sintaxis de las expresiones (o sentencias) queda del siguiente modo:

$$S ::= x := a \mid \texttt{skip} \mid S_1; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \mid \texttt{while } b \texttt{ do } S \mid \texttt{abort}$$

Por otro lado, anque debemos modificar ahora el comportamiento de las relaciones de transición que vimos en el anterior capítulo, no es necesario alterar los sistemas de transiciones asociados, porque tan solo queremos que cualquier configuración de la forma $\langle \texttt{abort}, s\rangle$ quede atascada, y esto no modifica el comportamiento de (las reglas asociadas a) las otras expresiones. Es decir, la expresión `abort` no necesita de ninguna regla que determine su interpretación.

Ahora bien, no es cierto que While_{ns} y $\text{While}_{\text{sos}}$ se comporten, en general, del mismo modo. De hecho, `abort` distingue entre ambos sistemas de semántica operacional. Esto se debe a que en While_{ns} solo nos interesan las ejecuciones que terminan correctamente y por tanto no distinguimos entre bucles o configuraciones atascadas, mientras que en $\text{While}_{\text{sos}}$ podemos definir bucles (secuencias de derivación infinitas) y ejecuciones que terminan incorrectamente (secuencias de derivación finitas que terminan en una configuración atascada). Así, aunque `abort` no sea semánticamente equivalente a `skip` en While_{ns}, sí que lo es `while true do skip`. Por otro lado, vemos que `abort` no puede ser semánticamente equivalente en $\text{While}_{\text{sos}}$ a `while true do skip`, porque a partir de $\langle \texttt{while true do skip}, s\rangle$ hay una secuencia de derivación infinita y a partir de $\langle \texttt{abort}, s\rangle$ no. Tampoco puede serlo `skip`, porque $\langle \texttt{skip}, s\rangle \Rightarrow s$ es la única secuencia de derivación posible empezando en `skip` y $\langle \texttt{abort}, s\rangle$ es la correspondiente a `abort`.

Ejemplo 3.1. Podemos extender While con la expresión `assert` b `before` S. La idea es que, si b es cierto, entonces ejecutamos S y, si es falso, entonces la ejecución del programa se aborta. [HACER ESTO]

3.1.2. `or`

Otra posible extensión de las expresiones de While consiste en añadir nuevas posibilidades de ejecución, es decir, forzar un tipo de no determinismo. Para ello incluimos la expresión `or`:

$$S ::= x := a \mid \texttt{skip} \mid S_1; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \mid \texttt{while } b \texttt{ do } S \mid S_1 \texttt{ or } S_2$$

Así, si por ejemplos ejecutamos `x := 1 or x := 2`, entonces generamos dos caminos distintos: uno en el que `x` pasa a tener el valor 1 y otro en el que toma el valor 2. Como siempre, en caso de que sea preciso, emplearemos paréntesis para indicar de manera precisa las distintas opciones.

A diferencia de la expresión `abort`, debemos distinguir la nueva ampliación de While en función de cada semántica operacional:

Sistema (While_{ns} con `op`)**.** A las reglas de While_{ns} añadimos:

$$[\text{or}^1_{\text{ns}}]\ \frac{\langle S_1, s\rangle \to s'}{\langle S_1 \texttt{ or } S_2, s\rangle \to s'}$$

$$[\text{or}^2_{\text{ns}}]\ \frac{\langle S_2, s\rangle \to s'}{\langle S_1 \texttt{ or } S_2, s\rangle \to s'}$$

Sistema ($\text{While}_{\text{sos}}$ con `op`)**.** A las reglas de $\text{While}_{\text{sos}}$ añadimos:

$$[\text{or}^1_{\text{sos}}]\ \frac{}{\langle S_1 \texttt{ or } S_2, s\rangle \Rightarrow \langle S_1, s\rangle}$$

$$[\text{or}^2_{\text{sos}}]\ \frac{}{\langle S_1 \texttt{ or } S_2, s\rangle \Rightarrow \langle S_2, s\rangle}$$

De nuevo, como cabía esperar, ambos sistemas de transiciones funcionan de maneras distintas respecto del no determinismo que hemos introducido. En el caso de While_{ns} tenemos que el no determinismo omite la posibilidad de que haya bucles infinitos, mientras que en $\text{While}_{\text{sos}}$ esto no ocurre. Para ilustrar esto vamos a recurrir al ejemplo de antes.

En While_{ns}, a la configuración $\langle \texttt{x} := 1 \texttt{ or } \texttt{x} := 2, s\rangle$ corresponden dos árboles de derivación, cada uno asociado a las siguientes transiciones:

$$\langle \texttt{x} := 1 \texttt{ or } \texttt{x} := 2, s\rangle \to s[\texttt{x} \mapsto 1]$$

$$\langle \texttt{x} := 1 \texttt{ or } \texttt{x} := 2, s\rangle \to s[\texttt{x} \mapsto 2]$$

Además, si tuviésemos $\langle(\texttt{while true do skip}) \texttt{ or } \texttt{x} := 2, s\rangle$, entonces solo habría un árbol de derivación posible, a saber, el que no corresponde al bucle, es decir,

$$\langle(\texttt{while true do skip}) \texttt{ or } \texttt{x} := 2, s\rangle \to s[\texttt{x} \mapsto 2]$$

En cambio, en $\text{While}_{\text{sos}}$, las secuencias de derivación para $\langle \texttt{x} := 1 \texttt{ or } \texttt{x} := 2, s\rangle$ serían

$$\langle \texttt{x} := 1 \texttt{ or } \texttt{x} := 2, s\rangle \Rightarrow^* s[\texttt{x} \mapsto 1]$$

$$\langle \texttt{x} := 1 \texttt{ or } \texttt{x} := 2, s\rangle \Rightarrow^* s[\texttt{x} \mapsto 2]$$

y para $\langle(\texttt{while true do skip}) \texttt{ or } \texttt{x} := 2, s\rangle$,

$$\langle(\texttt{while true do skip}) \texttt{ or } \texttt{x} := 2, s\rangle \Rightarrow^* s[\texttt{x} \mapsto 2]$$

$$\langle(\texttt{while true do skip}) \texttt{ or } \texttt{x} := 2, s\rangle \Rightarrow \langle \texttt{while true do skip}, s\rangle \Rightarrow \dots$$

de donde se deduce lo que dijimos arriba.

Ejemplo 3.2. Podemos extender While con la expresión `random(x)`. [HACER ESTO]

3.1.3. `par`

Otra manera de ampliar el lenguaje While consiste en que, por ejemplo, se dé la posibilidad de ejecutar dos expresiones, pero no de forma excluyente, es decir, que ambas ejecuciones se puedan ejecutar intercalándose. Esta es la idea detrás de la expresión `par`:

$$S ::= x := a \mid \texttt{skip} \mid S_1; S_2 \mid \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \mid \texttt{while } b \texttt{ do } S \mid S_1 \texttt{ par } S_2$$

En un caso sencillo como $\texttt{x} := 1$ `par` $\texttt{x} := 2$, tenemos solo los resultados posibles 1 y 2, correspondientes a ejecutar primero $\texttt{x} := 1$ y después $\texttt{x} := 2$ y viceversa. Si por ejemplo tuviésemos $\texttt{x} := 1$ `par` $(\texttt{x} := 2; \texttt{x} := \texttt{x} + 2)$, entonces habría tres posibles resultados:

- Si ejecutamos $\texttt{x} := 1$ y luego $\texttt{x} := 2; \texttt{x} := \texttt{x} + 2$, obtenemos 4.
- Si ejecutamos $\texttt{x} := 2; \texttt{x} := \texttt{x} + 2$ y luego $\texttt{x} := 1$, obtenemos 1.
- Si ejecutamos $\texttt{x} := 2$, después $\texttt{x} := 1$ y por último $\texttt{x} := \texttt{x} + 2$, obtenemos 3.

Veamos cómo queda el lenguaje While ampliado con esta nueva expresión. En el caso de $\text{While}_{\text{sos}}$ tenemos:

Sistema ($\text{While}_{\text{sos}}$ con `par`)**.** A las reglas de $\text{While}_{\text{sos}}$ añadimos[1]:

$$[\text{par}^1_{\text{sos}}] \frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1 \texttt{ par } S_2, s\rangle \Rightarrow \langle S_1' \texttt{ par } S_2, s'\rangle}$$

$$[\text{par}^2_{\text{sos}}] \frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1 \texttt{ par } S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

$$[\text{par}^3_{\text{sos}}] \frac{\langle S_2, s\rangle \Rightarrow \langle S_2', s'\rangle}{\langle S_1 \texttt{ par } S_2, s\rangle \Rightarrow \langle S_1 \texttt{ par } S_2', s'\rangle}$$

$$[\text{par}^4_{\text{sos}}] \frac{\langle S_2, s\rangle \Rightarrow s'}{\langle S_1 \texttt{ par } S_2, s\rangle \Rightarrow \langle S_1, s'\rangle}$$

Si intentásemos ampliar While_{ns} de la misma manera, podríamos comenzar escribiendo las siguientes reglas:

$$\frac{\langle S_2, s'\rangle \to s''}{\langle S_1 \texttt{ par } S_2, s\rangle \to \langle S_1, s''\rangle}$$

$$\frac{\langle S_1, s'\rangle \to s''}{\langle S_1 \texttt{ par } S_2, s\rangle \to \langle S_1, s''\rangle}$$

Pero entonces observamos que estas reglas únicamente expresan que S_1 se ejecuta antes o después de S_2, es decir, no parecen traducir correctamente el valor semántico que tenemos en mente para `par`. Esto se debe a que en While_{ns} nos interesa solo el desarrollo general de la ejecución y, por tanto, la ejecución de una expresión no se puede descomponer en ejecuciones intermedias.

Ejemplo 3.3 (EJEMPLO DEL protect ?????)**.**

3.2. Bloques y declaración de variables

Si bien en la anterior sección introdujimos nuevas expresiones que capturaran nuevos tipos de ejecuciones, lo que nos interesa ahora es añadir entornos para variables y procedimientos locales.

[1] Se hace notar que las reglas no son mutuamente excluyentes, es decir, podemos aplicar distintas reglas indistintamente, como hemos visto en los ejemplos anteriores.

3.2.1. Bloques y declaraciones simples

La primera extensión de While que consideraremos es el lenguaje Block. Éste consiste en la siguiente extensión de la sintaxis de las expresiones de While:

$$S ::= \mathtt{x} := a \mid \mathtt{skip} \mid S_1; S_2 \mid \mathtt{if}\ b\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2 \mid \mathtt{while}\ b\ \mathtt{do}\ S \mid \mathtt{begin}\ \ D_V\ \ S\ \ \mathtt{end}$$

donde D_V es una metavariable de la nueva categoría sintáctica $\mathbf{Dec}_V$ de *declaraciones de variables*:

$$D_V ::= \mathtt{var}\ \ \mathtt{x} := a; D_V \mid \varepsilon$$

siendo ε la *declaración vacía*. El conjunto de todas las variables declaradas en D_V, $DVar(D_V)$, se define como sigue:

$$DVar(\varepsilon) := \emptyset, \qquad DVar(\mathtt{var}\ \ \mathtt{x} := a; D_V) := \{\mathtt{x}\} \cup DVar(D_V)$$

Queremos que esta nueva clase de expresiones se interprete como la ejecución de otra expresión cualquiera bajo la declaración de variables de D_V, y que éstas variables vuelvan al estado anterior después de finalizar. Es decir, la idea es que las variables declaradas dentro del *bloque* $\mathtt{begin}\ \ D_V\ \ S\ \ \mathtt{end}$ sean *locales*, a diferencia de las que quedan fuera, que llamamos *globales*.

Ejemplo 3.4. Consideremos la expresión

$$\mathtt{begin\ var\ y} := 1; (\mathtt{x} := 1; \mathtt{begin\ var\ x} := 2; \mathtt{y} := \mathtt{x} + 1\ \ \mathtt{end}; \mathtt{x} := \mathtt{x} + \mathtt{y})\ \ \mathtt{end}$$

El símbolo $\mathtt{x}$ en $\mathtt{y} := \mathtt{x} + 1$ se refiere a la variable local introducida en $\mathtt{var}\ \ \mathtt{x} := 2$, mientras que en $\mathtt{x} := \mathtt{x} + \mathtt{y}$ se refiere a la variable global $\mathtt{x}$ en $\mathtt{x} := 1$. La variable $\mathtt{y}$ es global. Por tanto, $\mathtt{y} := \mathtt{x} + 1$ nos da el valor $2 + 1 = 3$ para $\mathtt{y}$ y, en $\mathtt{x} := \mathtt{x} + \mathtt{y}$, $\mathtt{x}$ toma el valor $1 + 3 = 4$.

Nos podemos centrar en ampliar While_{ns} (hacerlo con $\text{While}_{\text{sos}}$ es considerablemente más difícil). Tendremos que distinguir un sistema de transiciones para cada una de las dos categorías sintácticas que hemos introducido antes. Las configuraciones asociadas a $\mathbf{Dec}_V$ son del tipo $\langle D_V, s\rangle$ o s, con el significado usual, y denotamos por $\langle D_V, s\rangle \to_D s'$ a la función de transición asociada. Entonces tenemos:

Sistema (Block_{ns})**.** A las reglas de While_{ns} añadimos:

$$[\text{var}_{\text{ns}}]\ \frac{\langle D_V, s[x \to \mathcal{A}[\![a]\!]s]]\rangle \to_D s'}{\langle \mathtt{var}\ \ x := a; D_V, s\rangle \to_D s'}$$

$$[\text{none}_{\text{ns}}]\ \frac{}{\langle \varepsilon, s\rangle \to_D s}$$

$$[\text{block}_{\text{ns}}]\ \frac{\langle D_V, s\rangle \to_D s' \qquad \langle S, s'\rangle \to s''}{\langle \mathtt{begin}\ \ D_V\ \ S\ \ \mathtt{end}, s\rangle \to s''[\text{DV}(D_v) \mapsto s]}$$

Fíjemonos en que el objetivo de la última regla es ejecutar primero la sentencia S con la declaración de variables local D_V y que, después, aquellas variables dentro de D_V vuelvan al estado previo al $\mathtt{begin}$[2].

Ejemplo 3.5. Podríamos crear una función (sintáctica) que permita sustituir las expresiones de $\mathbf{Dec}_V$ por otras de **Stm**:

$$\begin{array}{rcl} \delta : \mathbf{Dec}_V & \longrightarrow & \mathbf{Stm} \\ \mathtt{var}\ \ \mathtt{x} := a; Dv & \mapsto & \mathtt{x} := a; \delta(D_V) \\ \varepsilon & \mapsto & \mathtt{skip} \end{array}$$

Entonces podríamos reinterpretar las reglas empleando la semántica que ya conocemos para While_{ns}. Por ejemplo, la regla block_{bs} podría expresarse como:

$$\frac{\langle \delta(D_V), s\rangle \to s' \qquad \langle S, s'\rangle \to s''}{\langle \mathtt{begin}\ \ D_V\ \ S\ \ \mathtt{end}, s\rangle \to s''[\text{DVar}(D_V) \mapsto s]}$$

es decir, sin emplear la transición $\to_D$. Las otras dos reglas se seguirían de las que ya vimos en While_{ns}.

3.2.2. Procedimientos

[2] Véase la sección 1.5.2 para la definición de sustitución múltiple.