Apuntes de Teoría de la Programación

4 de marzo de 2021

Contenidos

1. Introducción **1**
1.1. Un ejemplo de la lógica . . . 1
1.2. Métodos de descripción semántica . . . 2
1.2.1. Semántica operacional . . . 2
1.2.2. Semántica denotacional . . . 2
1.2.3. Semántica axiomática . . . 3
1.3. El lenguaje While . . . 3
1.4. Semántica para expresiones . . . 4
1.5. Propiedades semánticas . . . 6
1.5.1. Variables libres . . . 7
1.5.2. Sustitución . . . 8

2. Semántica operacional **10**
2.1. Semántica operacional natural . . . 10
2.1.1. Sistema de transiciones . . . 10
2.1.2. Propiedades . . . 12

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 están sintácticamente bien escritos, como en este ejemplo. La asignación de un valor semántico 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.

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 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 absracta*, 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:

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

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 $x \in \mathrm{FV}(a)$, $s\ x = s'$. 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 $x \in \mathrm{FV}(b)$, $s\ x = s'$. Entonces $\mathcal{B}[\![b]\!]s = \mathcal{B}[\![b]\!]s'$.*

Demostración. Casos base:

- Si $b := \texttt{true}$, $\mathcal{B}[\![b]\!]s := V =: \mathcal{B}[\![b]\!]s'$ y análogamente con $\texttt{false}$.
- 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 $b' \in \text{Bexp}$, sabemos que entonces $\text{FV}(b) = \text{FV}(b')$. Como suponemos que, para cada $x \in \text{FV}(b)$, $s\ x = s'\ x$, 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 $b_1, b_2 \in \text{Bexp}$, sabemos que entonces $\text{FV}(b) = \text{FV}(b_1) \cup \text{FV}(b_2)$ 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 $x \in \text{FV}(a)$, 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:

$$
\begin{aligned}
n[xa_0] &= n \\
y[x \mapsto a_0] &= \begin{cases} a_0, \text{ si } x = y \\ y, \text{ si } x \neq y \end{cases} \\
(a_1 + a_2)[x \mapsto a_0] &= a_1[x \mapsto a_0] + a_2[x \mapsto a_0] \\
(a_1 \times a_2)[x \mapsto a_0] &= a_1[x \mapsto a_0] \times a_2[x \mapsto a_0] \\
(a_1 - a_2)[x \mapsto a_0] &= a_1[x \mapsto a_0] - a_2[x \mapsto a_0]
\end{aligned}
$$

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 $a, a_0 \in \mathbf{Aexp}$, para todo $s \in \mathbf{State}$ 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.

□

Todo lo anterior justifica una noción que será importante a lo largo del curso. Dadas 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

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

2 | Semántica operacional

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

En este capítulo queremos estudiar cada enfoque para luego comprobar que, en cierto sentido, son ambos 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:

$[\text{ass}_{\text{ns}}]$

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

$[\text{skip}_{\text{ns}}]$

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

$[\text{comp}_{\text{ns}}]$

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

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

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

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

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

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

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

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

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

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:

(a) $\texttt{while}\, \neg(\texttt{x} = 1)\, \texttt{do}\, (\texttt{y} := \texttt{y} \times \texttt{x}; \texttt{x} := \texttt{x} - 1)$.

(b) $\texttt{while}\, 1 \leq \texttt{x}\, \texttt{do}\, (\texttt{y} := \texttt{y} \times \texttt{x}; \texttt{x} := \texttt{x} - 1)$.

(b) `while true do skip`.

HACER ESTO

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. `while` b `do` S *es semánticamente equivalente a* `if` b `then` $(S;$`while` b `do` $S)$ `else skip`.

Demostración. Dividimos la prueba en dos implicaciones:
Parte 1. Supongamos que se cumple $\langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \to s''$. 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:

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

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

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

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

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

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

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

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

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

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

Pero entonces,

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

y 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:

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

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:

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

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

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

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. Podríamos extender el lenguaje While con la regla `repeat` S `until` b.