SEMÁNTICA OPERACIONAL (O DE TRANSICIONES)

En vez de asociar significado a los programas a través de una función de los programas en los significados, se describe cómo se realiza el cómputo. Por ejemplo:

<x:= x+1;y:= 2*y , s> -> <y:= 2*y , [s | x : s x + 1]> -> [s | x : s x + 1 | y : 2 * s y]

La relación "->" describe un paso de ejecución. Por eso se la llama small-step semantics. En cada etapa, se pasa de una "configuración" a otra.

Γ = conjunto de configuraciones
Γ = Γ_t ∪ Γ_n

donde Γ_t es el conjunto de configuraciones terminales y Γ_n el de las configuraciones no terminales.

Γ_t = Σ
Γ_n = <comm> x Σ

La relación de un paso de ejecución "->", lleva una configuración no terminal en una configuración (terminal o no):

-> ⊆ Γ_n x Γ

La presentación es axiomática

<skip,s> -> s

<v:=e,s> -> [s|v:[[e]]s]

Ambas dan configuraciones terminales.

Ejemplos:

Podemos construir derivaciones para las asignaciones mostradas más arriba:

1)

<x:= x+1 , s> -> [s | x : s x + 1]

2)

<y:= 2*y , s'> -> [s' | y : 2 * s' y]

Secuencia de comandos

    <c0,s> -> s'

<c0;c1,s> -> <c1,s'>

   <c0,s> -> <c0',s'>

<c0;c1,s> -> <c0';c1,s'>

Ejemplo: del ejemplo 1) podemos deducir <x:= x+1;y:= 2*y , s> -> <y:= 2*y , [s | x : s x + 1]> :

          <x:= x+1 , s> -> [s | x : s x + 1]

<x:= x+1;y:= 2*y , s> -> <y:= 2*y , [s | x : s x + 1]>

Condicional:

--- si [[b]]s
<if b then c0 else c1,s> -> <c0,s>

--- si ¬[[b]]s
<if b then c0 else c1,s> -> <c1,s>

Ciclos:

--- si [[b]]s

<while b do c,s> -> <c;while b do c,s>

---------- si ¬[[b]]s
<while b do c,s> -> s

Como se vé, ahora no nos molestó definir el while en términos de sí mismo. Antes queríamos definir una función (que asocie significado a la sintaxis), dirección por sintaxis garantizaba exitencia y unicidad. Ahora sólo nos interesa definir una relación (un paso de ejecución). No cabe duda de que se está definiendo tal relación. Luego puede interesarnos estudiar propiedades de la relación, pero nadie puede dudar de que la relación -> está siendo definida unívocamente.

Variables locales. Podría definirse

---------- n = s v
<newvar v:= e in c,s> -> <c;v:= ñ, [s|v:[[e]]s]>

pero se pierde localía durante la ejecución de c (se notaría en caso de concurrencia ya que si otro comando se ejecuta en paralelo, podría observar el valor de v).

Se prefiere:

      <c, [s|v:[[e]]s]> -> s'

<newvar v:= e in c,s> -> [s'|v:s v]

              <c, [s|v:[[e]]s]> -> <c',s'>
---------- n = s' v
<newvar v:= e in c,s> -> <newvar v:= ñ in c',[s'|v:s v]>

La relación así definida es una función: en efecto, ninguna configuración no terminal puede mover (en un sólo paso) hacia más de una configuración (es determinística) y ninguna configuración no terminal puede mover hacia menos de una configuración (no se traba). Sea <g,s> una configuración no terminal. Entonces, si

g = skip: hay una única transición posible, a s.
g = v:= e: hay una única transición posible, a [s|v:[[e]]s].
g = if b then c0 else c1: hay una única transición posible. Si [[b]]s esa transición es a <c0,s>. En caso contrario, a <c1,s>.
g = c0;c1: En este caso, por hipótesis inductiva desde <c0,s> hay una única transición posible. Si esa transición es a una configuración de la forma s', entonces hay una única transición posible desde <g,s>, es a <c1,s'>. Si en cambio esa transición es a una configuración de la forma <c0',s'>, entonces hay una única transición posible desde <g,s>, es a <c0';c1,s'>.
etc.

Como -> es una función, para toda configuración existe una única ejecución (es decir, secuencia g0 -> g1 -> g2 -> ... maximal, que no puede prolongarse más de lo que está). Dicha ejecución es infinita o termina en una configuración terminal s. Si la ejecución es infinita decimos que g0 diverge y escribimos g0↑. Se puede definir

         ⌜ ⊥   si <c,s>↑
{{c}}s = ⎨
         ⌞ s'   si <c,s> -*-> s'

donde -*-> es la clausura reflexiva y transitiva definida a continuación. Se puede demostrar que {{c}} = [[c]].

Clausura reflexiva y transitiva:

 g -> g'
----
g -*-> g'

g -*-> g'   g' -*-> g''
----
       g -*-> g''

----
g -*-> g

Lema: a) Si <c0,s> -*-> s' entonces <c0;c1,s> -*-> <c1,s'>.
      b) Si <c0,s> -*-> <c0',s'> entonces <c0;c1,s> -*-> <c0';c1,s'>

Lema: a) Si <c,[s|v:[[e]]s]> -*-> s' entonces <newvar v:= e in c,s> -*-> [s'|v:s v].
      b) Si <c,[s|v:[[e]]s]> -*-> <c',s'> entonces <newvar v:= e in c,s> -*-> <newvar v:= ñ in c',[s'|v:s v]> donde n = s v.

FALLAS

Para dar la semántica de transiciones con fallas, modificamos el conjunto de configuraciones terminales:

$\Gamma_t = \Sigma \cup \{\langle abort \rangle\} \times \Sigma$

Además de las reglas que se dieron, agregamos

$$\frac{}{\langle fail,s\rangle \to \langle abort,s\rangle}$$

Es necesario agregar un caso más a la secuencia:

$$\frac{\langle c0,s\rangle \to \langle abort,s'\rangle}{\langle c0;c1,s\rangle \to \langle abort,s'\rangle}$$

El catchin se define por tres reglas

$$\frac{\langle c0,s\rangle \to s'}{\langle catchin\ c0\ with\ c1,s\rangle \to s'}$$

$$\frac{\langle c0,s\rangle \to \langle abort,s'\rangle}{\langle catchin\ c0\ with\ c1,s\rangle \to \langle c1,s'\rangle}$$

$$\frac{\langle c0,s\rangle \to \langle c0',s'\rangle}{\langle catchin\ c0\ with\ c1,s\rangle \to \langle catchin\ c0'\ with\ c1,s'\rangle}$$

Por último, al newvar se le agrega un caso más

$$\frac{\langle c, [s|v:[[e]]s]\rangle \to \langle abort,s'\rangle}{\langle newvar\ v:=\ e\ in\ c,s\rangle \to \langle abort,[s'|v:s\ v]\rangle}$$

El if y el while habían sido definidos con suficiente generalidad para que no requieran revisión.

La relación -> sigue siendo una función, toda configuración g tiene una única ejecución que puede ser infinita (g diverge, $g\uparrow$) o terminar en una configuración terminal que puede ser de la forma s o <abort,s>. Se puede definir

$$\{\{c\}\}s = \begin{cases} \perp & \text{si } \langle c,s\rangle\uparrow \\ s' & \text{si } \langle c,s\rangle \to^* s' \\ \langle abort,s'\rangle & \text{si } \langle c,s\rangle \to^* \langle abort,s'\rangle \end{cases}$$