SEMÁNTICA OPERACIONAL (O DE TRANSICIONES)
---

FALLAS
---

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

Γ_t = Σ ∪ {<abort>} x Σ

Además de las reglas que se dieron, agregamos

```
---------------------
<fail,s> -> <abort,s>
```

Es necesario agregar un caso más a la secuencia:

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

El catchin se define por tres reglas

```
        <c0,s> -> s'
----------------------------
<catchin c0 with c1,s> -> s'

      <c0,s> -> <abort,s'>
---------------------------------
<catchin c0 with c1,s> -> <c1,s'>

               <c0,s> -> <c0',s'>
-------------------------------------------------
<catchin c0 with c1,s> -> <catchin c0' with c1,s'>
```

Por último, al newvar se le agrega un caso más

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

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↑) o terminar en una configuración terminal que puede ser de la forma s o <abort,s>. Se puede definir

$$\{\{c\}\}s = \begin{cases} \bot & \text{si } \langle c,s\rangle\uparrow \\ s' & \text{si } \langle c,s\rangle \text{ -*-> } s' \\ \langle abort,s'\rangle & \text{si } \langle c,s\rangle \text{ -*-> } \langle abort,s'\rangle \end{cases}$$

ENTRADA Y SALIDA
---

Γ = conjunto de configuraciones
Γ = Γ_t ∪ Γ_n
Γ_t = Σ ∪ {<abort>} x Σ
Γ_n = <comm> x Σ

Para incorporar entrada salida, la relación de transición cambia. En vez de ser

-> ⊆ Γ_n x Γ

pasa a ser ternaria

-> ⊆ Γ_n x Δ x Γ

donde Δ = {ə} ∪ {?n | n ∈ N} ∪ {!n | n ∈ N}

Es decir, las transiciones tienen rótulo que indican si mientras se cambiaba de una configuración a otra se emitió output (rótulo !n) o se leyó input (rótulo ?n) o no ocurrió ninguna de las dos cosas (rótulo ə, que en general se omite).

Todas las reglas que hemos dado son correctas, dado que las podemos interpretar como teniendo el rótulo ə, que, como dijimos, se omite.

Sin embargo algunas pueden escribirse con mayor generalidad:

$$\frac{<c0,s> \xrightarrow{\lambda} s'}{<c0;c1,s> \xrightarrow{\lambda} <c1,s'>}$$

$$\frac{<c0,s> \xrightarrow{\lambda} <c0',s'>}{<c0;c1,s> \xrightarrow{\lambda} <c0';c1,s'>}$$

$$\frac{<c0,s> \xrightarrow{\lambda} <abort,s'>}{<c0;c1,s> \xrightarrow{\lambda} <abort,s'>}$$

Esta nueva notación indica que si para realizarse el cambio de configuración de la premisa se produjo un cierto efecto (input, output, o silencio), el mismo efecto se produce al realizarse el cambio de configuración de la conclusión. El mismo tratamiento se hace al catchin y al newvar.

$$\frac{<c0,s> \xrightarrow{\lambda} s'}{<\text{catchin } c0 \text{ with } c1,s> \xrightarrow{\lambda} s'}$$

$$\frac{<c0,s> \xrightarrow{\lambda} <abort,s'>}{<\text{catchin } c0 \text{ with } c1,s> \xrightarrow{\lambda} <c1,s'>}$$

$$\frac{<c0,s> \xrightarrow{\lambda} <c0',s'>}{<\text{catchin } c0 \text{ with } c1,s> \xrightarrow{\lambda} <\text{catchin } c0' \text{ with } c1,s'>}$$

$$\frac{<c, [s|v:[[e]]s]> \xrightarrow{\lambda} s'}{<\text{newvar } v:= e \text{ in } c,s> \xrightarrow{\lambda} [s'|v:s\ v]}$$

$$<c, [s|v:[[e]]s]> \xrightarrow{\lambda} <c',s'>$$

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


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

Los nuevos casos son los corresondientes al input y output:

```
----------- n = [[e]]s
       !n
<!e,s> -> s


----------------- n ∈ N
       ?n
<?v,s> -> [s|v:n]
```

Ahora -> no es función, ya que en una configuración dada no siempre hay una sola transición posible. En general se da una y sólo una de las siguientes posibilidades en una configuración no terminal g:
1) existe una única transición y es silenciosa g -> g'
2) existe una única transición y lleva rótulo !n con n ∈ Z
3) existen infinitas transiciones y todas llevan rótulo ?n, es más, existe exactamente una de estas transiciones para cada rótulo n ∈ Z.

Podemos seguir usando la notación -*-> para secuencias de transiciones silenciosas.

El análisis realizado implica que a partir de una configuración no terminal g existe una única secuencia maximal de cero o más (tal vez infinitas) transiciones silenciosas. Esta secuencia puede:
a) ser infinita, en cuyo caso g↑
b) ser finita y terminar en una configuración terminal s ∈ Σ
c) ser finita y terminar en una configuración terminal <abort,s>
d) ser finita y terminar en una configuración no terminal g' desde donde existe una única transición y lleva rótulo !n con n ∈ Z
e) ser finita y terminar en una configuración no terminal g' desde donde existe exactamente una transición de la forma ?n para cada n ∈ Z

Esto permite definir F ∈ Γ -> Ω

```
      ⎧⊥              si g↑
      ⎪Lterm s           si g -*-> s
F g = ⎨Labort s          si g -*-> <abort,s>
      ⎪Lout <n,F g">     si g -*-> g' y g' -> g", está ultima transición con rótulo !n
      ⎩Lin (λn∈Z.F gn)  si g -*-> g' y para cada n∈Z existe una transición con rótulo ?n de g' -> gn
```

Nuevamente puede demostrarse que [[c]]s = F <c,s>.

CÁLCULO LAMBDA
--------------

Motivación: notación para no necesitar nombrar funciones. Por ejemplo, x+y puede ser:

```
f(x) = x+y
g(y) = x+y
h(x,y) = x+y
```

La notación lambda permite expresar sin dar nombre:

```
λx.x+y
λy.x+y
λ(x,y).x+y
```

e incluso

λx.λy.x+y
λy.λx.x+y

En algunos contextos, los matemáticos usan notaciones similares:

En integrales, ∫x+y.dx enfatiza que x varía e y está fija. En ∑_{i∈{0..10}} i+j, i varía y j está fija.

El cálculo lambda, en principio es una notación para no tener que nombrar funciones. Pero además, el cálculo lambda puro, sólo contiene variables, aplicaciones y la notación lambda (llamada abstracción).

```
<exp> ::=                {término lambda, o expresión}
        <var>            {variable}
      | <exp> <exp>      {aplicación, el primero es el operador y el segundo el operando}
      | λ<var>.<exp>     {abstracción o expresión lambda}
```

La aplicación asocia a izquierda. En λv.e, la primer ocurrencia de v es ligadora y su alcance es e.

Por ejemplo, λx.(λy.xyx)x es lo mismo que λx.((λy.((xy)x))x)

Se hacen exactamente las mismas definiciones de ocurrencia ligadora, ocurrencia libre y variable libre. El conjunto de variables libres se define también por inducción en la estructura de los términos:

```
FV(v) = {v}
FV(e0 e1) = FV(e0) ∪ FV(e1)
FV(λv.e) = FV(e) - {v}
```

También se define sustitución:

```
Δ = <var> -> <exp>
_/_ ∈ <exp> x Δ -> <exp>

v/d = d v
(e0 e1)/d = (e0/d) (e1/d)
(λv.e)/d = λv'.(e/[d|v:v'])
```

donde v' ∉ ∪{FV(d w) | w ∈ FV(e) - {v}}

La sustitución en el cálculo lambda es fundamental: permite definir la semántica operacional.