Lenguajes y Compiladores

2015

Estructura de la materia a grandes rasgos:

Primera Parte: Lenguaje imperativo

Segunda Parte: Lenguaje aplicativo puro, y lenguaje aplicativo con referencias y asignación

Ejes de contenidos de la primer parte

1. Introducción a la sintaxis y la semántica de lenguajes
2. El problema de dar significado a la recursión e iteración
3. Un Lenguaje Imperativo Simple

Un Lenguaje Imperativo Simple

LIS

$$
\begin{array}{lcl}
\langle comm \rangle & ::= & \textbf{skip} \\
 & & \langle var \rangle := \langle intexp \rangle \\
 & & \langle comm \rangle \ ; \ \langle comm \rangle \\
 & & \textbf{if } \langle boolexp \rangle \textbf{ then } \langle comm \rangle \textbf{ else } \langle comm \rangle \\
 & & \textbf{newvar } \langle var \rangle := \langle intexp \rangle \textbf{ in } \langle comm \rangle \\
 & & \textbf{while } \langle boolexp \rangle \textbf{ do } \langle comm \rangle
\end{array}
$$

$$
\begin{array}{lll}
\langle intexp \rangle & ::= & \langle natconst \rangle \\
 & & \langle var \rangle \\
 & & \langle intexp \rangle + \langle intexp \rangle \\
 & & \langle intexp \rangle * \langle intexp \rangle \\
 & & \langle intexp \rangle - \langle intexp \rangle \\
 & & \langle intexp \rangle \,/\, \langle intexp \rangle \\
 & & \langle intexp \rangle \,\%\, \langle intexp \rangle \\
 & & - \langle intexp \rangle
\end{array}
$$

$$
\begin{array}{lll}
\langle boolexp \rangle & ::= & \langle boolconst \rangle \\
 & & \langle intexp \rangle = \langle intexp \rangle \\
 & & \langle intexp \rangle < \langle intexp \rangle \\
 & & \langle intexp \rangle \leq \langle intexp \rangle \\
 & & \langle intexp \rangle > \langle intexp \rangle \\
 & & \langle intexp \rangle \geq \langle intexp \rangle \\
 & & \neg \langle boolexp \rangle \\
 & & \langle boolexp \rangle \vee \langle boolexp \rangle \\
 & & \langle boolexp \rangle \wedge \langle boolexp \rangle
\end{array}
$$

$$
\begin{array}{lll}
\langle natconst \rangle & ::= & 0 \mid 1 \mid 2 \mid .... \\
\langle boolconst \rangle & ::= & \textbf{true} \mid \textbf{false}
\end{array}
$$

Características de LIS

- Hay sólo dos tipos de expresiones: enteras y booleanas
- Las expresiones enteras siempre se pueden evaluar, y su resultado es un entero.Todas las funciones primitivas (incluida la división) son funciones totales.
- Sólo posee variables que adoptan valores enteros. Luego la noción de **estado** se refleja en la siguiente definición:

$$\textbf{Conjunto de estados:} \qquad \Sigma = \langle var \rangle \rightarrow \mathbf{Z}$$

(la memoria posee infinitos lugares que siempre alojan un número entero).

Significado de los comandos de LIS (sin iteración)

Funciones semánticas:

$[\![\_]\!]^{intexp} \in \langle intexp \rangle \to \Sigma \to \mathbf{Z}$

$[\![\_]\!]^{boolexp} \in \langle boolexp \rangle \to \Sigma \to \{V, F\}$

$[\![\_]\!]^{comm} \in \langle comm \rangle \to \Sigma \to \Sigma$

Debemos notar que dado que no estamos considerando la iteración en LIS, todos los comandos terminan en su ejecución, y su “resultado” es un nuevo estado.

Primeras ecuaciones semánticas

$$
\begin{array}{lcl}
[\![\textbf{skip}]\!]\sigma & = & \sigma \\
[\![v := e]\!]\sigma & = & [\sigma | v : [\![e]\!]\sigma] \\
[\![c_0; c_1]\!]\sigma & = & [\![c_1]\!]([\![c_0]\!]\sigma) \\
[\![\textbf{if } e \textbf{ then } c \textbf{ else } c']\!]\sigma & = & \text{if } [\![e]\!]\sigma \text{ then } [\![c]\!]\sigma \text{ else } [\![c']\!]\sigma
\end{array}
$$

Ejemplo:

$$
\begin{array}{lcl}
[\![x := x - 1]\!]_\sigma & = & [\sigma | x : [\![x - 1]\!]_\sigma] \\
 & = & [\sigma | x : [\![x]\!]_\sigma - [\![1]\!]_\sigma] \\
 & = & [\sigma | x : \sigma\ x - 1] \\
[\![y := y + x]\!]_\sigma & = & [\sigma | y : [\![y + x]\!]_\sigma] \\
 & = & [\sigma | y : [\![y]\!]_\sigma + [\![x]\!]_\sigma] \\
 & = & [\sigma | y : \sigma\ y + \sigma\ x]
\end{array}
$$

Semántica de **newvar**

Función "restauración de v según σ":

$$f_{v,\sigma}\sigma' = [\sigma'|v : \sigma v]$$

$$[\![\textbf{newvar } v := e \textbf{ in } c]\!]\sigma \quad = \quad f_{v,\sigma}([\![c]\!][\sigma|v : [\![e]\!]\sigma])$$

Semántica de **newvar**

Función "restauración de v según σ":

$$f_{v,\sigma}\sigma' = [\sigma'|v : \sigma v]$$

$$[\![\textbf{newvar } v := e \textbf{ in } c]\!]\sigma \quad = \quad f_{v,\sigma}([\![c]\!][\sigma|v : [\![e]\!]\sigma])$$

Notación lambda para la función restauración:

$$f_{v,\sigma} \quad = \quad \lambda\sigma' \in \Sigma.\ [\sigma'|v : \sigma v]$$

$$[\![\textbf{newvar } v := e \textbf{ in } c]\!]\sigma \quad = \quad (\lambda\sigma' \in \Sigma.\ [\sigma'|v : \sigma v])\ ([\![c]\!][\sigma|v : [\![e]\!]\sigma])$$

Dificultades para dar significado a la iteración

El comando

$$\textbf{while } b \textbf{ do } c$$

debe satisfacer la propiedad:

$$\begin{aligned} [\![\textbf{while } b \textbf{ do } c]\!]\sigma &= \text{if } [\![b]\!]\sigma \text{ then } [\![c; \textbf{while } b \textbf{ do } c]\!]\sigma \text{ else } [\![\textbf{skip}]\!]\sigma \\ &= \begin{cases} [\![\textbf{while } b \textbf{ do } c]\!]([\![c]\!]\sigma) & \text{si } [\![b]\!]\sigma \\ \sigma & \text{si } \neg[\![b]\!]\sigma \end{cases} \end{aligned}$$

¿Define esta propiedad una función?

Surgen varias dificultades al pretender dar significado de manera genérica a las funciones definidas de esta manera.

- el dominio semántico $\Sigma \to \Sigma$ es inadecuado como significado de un comando, ya que al incluir la iteración se incorpora la posibilidad de que el programa no termine. Solucionamos este inconveniente definiendo:

  $\Sigma_\perp = \Sigma \cup \{\perp\}$.

- La expresión

$$[\![\textbf{while } b \textbf{ do } c]\!]([\![c]\!]\sigma) \qquad (2)$$

  tiene el problema de que $[\![c]\!]\sigma$ puede ser $\perp$, que no pertenece al dominio de $[\![\textbf{while } b \textbf{ do } c]\!]$.

Esto se soluciona acudiendo a nuestras funciones auxiliares de transferencia de control. Si $f \in \Sigma \to \Sigma_\perp$, entonces definimos una nueva función $f_{\perp\!\!\perp} \in \Sigma_\perp \to \Sigma_\perp$ de la siguiente manera:

$$f_{\perp\!\!\perp}\sigma = \sigma \qquad\qquad f_{\perp\!\!\perp}\perp = \perp$$

Luego la expresión (2) puede ser corregida poniendo

$$[\![\textbf{while } b \textbf{ do } c]\!]_{\perp\!\!\perp}([\![c]\!]\sigma)$$

Dificultad principal: no es dirigida por sintaxis

La propiedad del **while** no puede ser tomada como definición.
Pero podemos tener la certeza de:

- el significado de $[\![\textbf{while } b \textbf{ do } c]\!]$ es una función $\omega \in \Sigma \to \Sigma_\perp$
- tal función ω satisface la siguiente "ecuación funcional":

$$\omega\sigma \quad = \quad \begin{cases} \omega_{\perp\!\!\perp}([\![c]\!]\sigma) & \text{si } [\![b]\!]\sigma \\ \sigma & \text{si } \neg[\![b]\!]\sigma \end{cases} \qquad (W)$$

Obtenemos entonces una ecuación similar a (ER)

Semántica denotacional de LIS

$$[\![\ ]\!] \in \langle comm \rangle \to \Sigma \to \Sigma_\perp$$

$$
\begin{array}{lcl}
[\![\textbf{skip}]\!]\sigma & = & \sigma \\
[\![v := e]\!]\sigma & = & [\sigma | v : [\![e]\!]\sigma] \\
[\![c_0; c_1]\!]\sigma & = & [\![c_1]\!]_{\perp\!\!\perp}([\![c_0]\!]\sigma) \\
[\![\textbf{if } e \textbf{ then } c \textbf{ else } c']\!]\sigma & = & \text{if } [\![e]\!]\sigma \text{ then } [\![c]\!]\sigma \text{ else } [\![c']\!]\sigma \\
[\![\textbf{newvar v:= e in c}]\!]_\sigma & = & (\lambda\sigma' \in \Sigma.\ [\sigma' | v : \sigma\ v])_{\perp\!\!\perp}([\![c]\!][\sigma | v : [\![e]\!]_\sigma])
\end{array}
$$

Semántica del **while**

Sea

$$F\ w\ \sigma = \begin{cases} w_{\perp\!\!\perp}([\![c]\!]_\sigma) & \text{si } [\![b]\!]_\sigma \\ \sigma & \text{si no} \end{cases}$$

entonces, la ecuación del **while** queda

$$[\![\textbf{while } b \textbf{ do } c]\!]_\sigma = F\ [\![\textbf{while } b \textbf{ do } c]\!]\ \sigma$$

Aquí:

$[\![\textbf{while } b \textbf{ do } c]\!] \in \Sigma \to \Sigma_\perp$

$F \in (\Sigma \to \Sigma_\perp) \to (\Sigma \to \Sigma_\perp)$

Luego la ecuación también puede escribirse

$$[\![\textbf{while } b \textbf{ do } c]\!] = F\ [\![\textbf{while } b \textbf{ do } c]\!]$$

Semántica del **while** usando el TMPF

Para poder utilizar el TMPF, deberíamos asegurarnos de que F es continua. En caso de serlo, la semántica de **while** b **do** c será

$$[\![\textbf{while } b \textbf{ do } c]\!] = \bigsqcup_{i=0}^{\infty} F^i \perp_{\Sigma \to \Sigma_\perp}$$

para

$$F\ w\ \sigma = \begin{cases} w_{\perp\!\!\perp}([\![c]\!]_\sigma) & \text{si } [\![b]\!]_\sigma \\ \sigma & \text{si no} \end{cases}$$

Esta definición sí es dirigida por sintaxis.

¿Cómo calcular el menor punto fijo en $\Sigma \to \Sigma_\perp$?

Si w es la solución buscada (el menor punto fijo), entonces obtenemos el valor de $w\ \sigma$ de la siguiente manera:

- Si la cadena

$$F^0\ \perp_{\Sigma\to\Sigma_\perp} \sigma,\quad F^1\ \perp_{\Sigma\to\Sigma_\perp} \sigma,\quad F^2\ \perp_{\Sigma\to\Sigma_\perp} \sigma\ \ldots$$

  adopta algún valor ditinto de $\perp$, ese será el valor de $w\ \sigma$

- Si la cadena mencionada es siempre $\perp$, entonces $w\ \sigma = \perp$

Ejemplo

while $x \neq 0 \wedge x \neq 1$ **do** $x := x - 2$.

$$[\![\textbf{while } x \neq 0 \wedge x \neq 1 \textbf{ do } x := x - 2]\!]_= \bigsqcup_{i=0}^{\infty} F^i \perp_{\Sigma \to \Sigma_\perp}$$

para

$$F\ w\ \sigma = \begin{cases} w_{\perp\!\!\perp}([\![x := x - 2]\!]_\sigma) & \text{si } [\![x \neq 0 \wedge x \neq 1]\!]_\sigma \\ \sigma & \text{si no} \end{cases}$$

$$= \begin{cases} \sigma & \text{si } \sigma\ x \in \{0, 1\} \\ w\ [\sigma | x : \sigma\ x - 2] & \text{si no} \end{cases}$$

Se puede comprobar que

$$F^i \perp_{\Sigma \to \Sigma_\perp} \sigma = \begin{cases} [\sigma | x : \sigma x \,\%\, 2] & \text{si } \sigma\ x \in \{0, 1, \ldots, 2 * i - 1\} \\ \perp & \text{si } \sigma x \notin \{0, 1, \ldots, 2 * i - 1\} \end{cases}$$

Luego

$$\bigsqcup_{i=0}^{\infty} F^i \perp_{\Sigma \to \Sigma_\perp} = \sigma \mapsto \begin{cases} [\sigma | x : \sigma x \,\%\, 2] & \text{si } \sigma\ x \geq 0 \\ \perp & \text{si no} \end{cases}$$

Variables Libres

$$
\begin{aligned}
FV(\textbf{skip}) &= \emptyset \\
FV(v := e) &= \{v\} \cup FV(e) \\
FV(c_0; c_1) &= FV(c_0) \cup FV(c_1) \\
FV(\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1) &= FV(b) \cup FV(c_0) \cup FV(c_1) \\
FV(\textbf{while } b \textbf{ do } c) &= FV(b) \cup FV(c) \\
FV(\textbf{newvar } \text{v} := \text{e} \textbf{ in } \text{c}) &= FV(e) \cup (FV(c) - \{v\})
\end{aligned}
$$

Variables asignables

$$
\begin{aligned}
FA(\textbf{skip}) &= \emptyset \\
FA(v := e) &= \{v\} \\
FA(c_0; c_1) &= FA(c_0) \cup FA(c_1) \\
FA(\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1) &= FA(c_0) \cup FA(c_1) \\
FA(\textbf{while } b \textbf{ do } c) &= FA(c) \\
FA(\textbf{newvar } \text{v:= e } \textbf{in } \text{c}) &= FV(c) - \{v\}
\end{aligned}
$$

Propiedades del LIS

Teorema de Coincidencia (TC)

Si dos estados σ y σ' coinciden en las variables libres de c, entonces da lo mismo evaluar c en σ o σ'.

¿Es correcto enunciarlo como sigue?

$$(\forall w \in FV(c).\ \sigma w = \sigma' w) \Rightarrow [\![c]\!]_\sigma = [\![c]\!]'_\sigma$$

Teorema de Coincidencia (TC)

1. $(\forall w \in FV(c).\ \sigma w = \sigma' w)$ implica,

   o bien $[\![c]\!]_\sigma = \perp = [\![c]\!]'_\sigma$,

   o bien $[\![c]\!]_\sigma \neq \perp \neq [\![c]\!]'_\sigma$ y

   $$\forall w \in FV(c).\ [\![c]\!]_\sigma w = [\![c]\!]'_\sigma w$$

2. Si $[\![c]\!]_\sigma \neq \perp$, entonces $\forall w \notin FA(c).\ [\![c]\!]_\sigma w = \sigma w$.

Teorema de Renombre (TR)

No importa el nombre de las variables utilizadas en las declaraciones de variables locales (o sea las ligadas):

$u \notin FV(c) - \{v\} \Rightarrow$

$$[\![\textbf{newvar } u := e \textbf{ in } c/v \to u]\!] = [\![\textbf{newvar } v := e \textbf{ in } c]\!]$$

Sustituciones

Conjunto de las sustituciones:

$$\Delta = \langle var \rangle \to \langle var \rangle$$

Operador Sustitución:

$$\_/\_ \in \langle comm \rangle \times \Delta \to \langle comm \rangle$$

Sustituciones

Conjunto de las sustituciones:

$$\Delta = \langle var \rangle \to \langle var \rangle$$

Operador Sustitución:

$$\_/\_ \in \langle comm \rangle \times \Delta \to \langle comm \rangle$$

$$\begin{aligned}
\textbf{skip}/\delta &= \textbf{skip} \\
(v := e)/\delta &= (\delta v) := (e/\delta) \\
(c_0; c_1)/\delta &= (c_0/\delta); (c_1/\delta) \\
(\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1)/\delta &= \textbf{if } b/\delta \textbf{ then } c_0/\delta \textbf{ else } c_1/\delta \\
(\textbf{while } b \textbf{ do } c)/\delta &= \textbf{while } b/\delta \textbf{ do } c/\delta
\end{aligned}$$

Operador sustitución para **newvar**

$$(\textbf{newvar}\ v := e\ \textbf{in}\ c)/\delta =$$

$$\textbf{newvar}\ v_{new} := e/\delta\ \textbf{in}\ (c/[\delta | v : v_{new}])$$

donde $v_{new} \notin \{\delta w | w \in FV(c) - \{v\}\}$

¿Cómo enunciar el Teorema de Sustitución para comandos?

¿si aplico la sustitución δ a c y luego evalúo en el estado σ, puedo obtener el mismo resultado a partir de c sin sustituir si evalúo en un estado que hace el trabajo de δ y de σ (en las variables libres de c)?

O sea, vale

$$(\forall w \in FV(c).\ \sigma(\delta w) = \sigma' w) \Rightarrow [\![c/\delta]\!]_\sigma = [\![c]\!]'_\sigma ?$$

Los estados originales adoptarán eventualmente valores distintos en las variables que no ocurren libres en c, y en consecuencia los estados finales diferirán en las mismas.

Debemos comparar $[\![c/\delta]\!]_\sigma(\delta w)$ con $[\![c]\!]'_\sigma w$.

Por ejemplo, si al programa $c = (\mathsf{x} := \mathsf{x} - 1; \mathsf{y} := 2 * \mathsf{y})$ se le sustituye x por u e y por v (sustitución δ), entonces al ejecutar c/δ en un estado σ deberíamos obtener en u y v los valores que se obtienen al ejecutar c en un estado en donde x e y adopten los valores de σu y σv resp.

Problema del alias

Consideremos el programa $\mathsf{x} := \mathsf{x} - 1; \mathsf{y} := 2 * \mathsf{y}$.

$\delta\ x = \delta\ y = z$

El programa resultante será $\mathsf{z} := \mathsf{z} - 1; \mathsf{z} := 2 * \mathsf{z}$

Aunque $\sigma' x = \sigma z = \sigma' y$, se tiene

$$[\![\mathsf{z} := \mathsf{z} - 1; \mathsf{z} := 2 * \mathsf{z}]\!]_\sigma z \neq [\![\mathsf{x} := \mathsf{x} - 1; \mathsf{y} := 2 * \mathsf{y}]\!]'_\sigma x$$

$$[\![\mathsf{z} := \mathsf{z} - 1; \mathsf{z} := 2 * \mathsf{z}]\!]_\sigma z \neq [\![\mathsf{x} := \mathsf{x} - 1; \mathsf{y} := 2 * \mathsf{y}]\!]'_\sigma y$$

El problema surge porque δ no es inyectiva.

Teorema de Sustitución

Si δ es inyectiva sobre $FV(c)$ y

$\forall w \in FV(c).\ \sigma(\delta w) = \sigma' w$, entonces

o bien $[\![c/\delta]\!]_\sigma = \perp = [\![c]\!]'_\sigma$,

o bien $[\![c/\delta]\!]_\sigma \neq \perp \neq [\![c]\!]'_\sigma$ y

$$\forall w \in FV(c).\quad [\![c/\delta]\!]_\sigma(\delta w) = [\![c]\!]'_\sigma w.$$

Fallas en el Lenguaje Imperativo

Ahora un programa tiene 3 comportamientos posibles:

1. da un estado final
2. aborta y da un estado final
3. no termina

Fallas en el Lenguaje Imperativo

Para la incorporación de la posibilidad de transferencia de control por fallas, se agregan excepciones al lenguaje:

$$\langle comm \rangle ::= \textbf{fail} \mid \textbf{catchin}\ \langle comm \rangle\ \textbf{with}\ \langle comm \rangle$$

Fallas en el Lenguaje Imperativo

Para la incorporación de la posibilidad de transferencia de control por fallas, se agregan excepciones al lenguaje:

$$\langle comm \rangle ::= \textbf{fail} \mid \textbf{catchin}\ \langle comm \rangle\ \textbf{with}\ \langle comm \rangle$$

Dominio de resultados posibles:

$\Sigma' = \Sigma \cup \{\textbf{abort}\} \times \Sigma$ (con el orden discreto)

Fallas en el Lenguaje Imperativo

Para la incorporación de la posibilidad de transferencia de control por fallas, se agregan excepciones al lenguaje:

$$\langle comm \rangle ::= \textbf{fail} \mid \textbf{catchin}\ \langle comm \rangle\ \textbf{with}\ \langle comm \rangle$$

Dominio de resultados posibles:

$\Sigma' = \Sigma \cup \{\textbf{abort}\} \times \Sigma$ (con el orden discreto)

Función semántica:

$$[\![\_]\!] \in \langle comm \rangle \to \Sigma \to \Sigma'_\perp$$

Ecuaciones semánticas

$$
\begin{aligned}
[\![\textbf{skip}]\!]_\sigma &= \sigma \\
[\![\textbf{fail}]\!]_\sigma &= \langle \textbf{abort}, \sigma \rangle \\
[\![v := e]\!]_\sigma &= [\sigma | v : [\![e]\!]_\sigma] \\
[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]_\sigma &= \begin{cases} [\![c_0]\!]_\sigma & \text{si } [\![b]\!]_\sigma \\ [\![c_1]\!]_\sigma & \text{si no} \end{cases}
\end{aligned}
$$

Operadores de transferencia de control: $*$

Dada $f \in \Sigma \to \Sigma'_\perp$, denotamos por f_* la siguiente extensión de f a $\Sigma'_\perp$:

$$f_* \in \Sigma'_\perp \to \Sigma'_\perp$$

$$f_* x = \begin{cases} f\sigma & \text{si } x = \sigma \in \Sigma \\ x & \text{si no} \end{cases}$$

En este caso, la presencia de una situación abortiva determina que no se transfiere el control a f. Servirá para describir el significado de $c_0; c_1$, ya que si ocurre una situación de excepción al ejecutar c_0, el control no es tranferido a c_1.

Operadores de transferencia de control: $+$

Dado $f \in \Sigma \to \Sigma'_\perp$, denotamos por f_+ la siguiente extensión de f a $\Sigma'_\perp$:

$$f_+ \in \Sigma'_\perp \to \Sigma'_\perp$$

$$f_+ x = \begin{cases} f\sigma & \text{si } x = \langle \textbf{abort}, \sigma \rangle \in \{\textbf{abort}\} \times \Sigma \\ x & \text{si no} \end{cases}$$

En una clara dualidad con la definición de f_*, la definición de f_+ determina lo contrario: se tranfiere el control a f sólo en caso de excepción. Esto corresponderá a **catchin** c **with** c'.

Operadores de transferencia de control: †

Dado $f \in \Sigma \to \Sigma$, denotamos por $f_\dagger$ la siguiente extensión de f a $\Sigma'_\perp$:

$$f_\dagger \in \Sigma'_\perp \to \Sigma'_\perp$$

$$f_\dagger x = \begin{cases} \langle \textbf{abort}, f\sigma \rangle & x = \langle \textbf{abort}, \sigma \rangle \\ fx & x \in \Sigma \\ \perp & x = \perp \end{cases}$$

Note que aquí hay una transferencia de control a f en cualquier situación (abortiva o no). Servirá para restaurar el valor de las variables locales.

Restantes ecuaciones semánticas

$$
\begin{aligned}
[\![c_0; c_1]\!]\sigma &= [\![c_1]\!]_*([\![c_0]\!]\sigma) \\
[\![\textbf{catchin } c_0 \textbf{ with } c_1]\!]\sigma &= [\![c_1]\!]_+([\![c_0]\!]\sigma) \\
[\![\textbf{newvar v:= e in c}]\!]\sigma &= (\lambda\sigma' \in \Sigma.\ [\sigma' | v : \sigma\ v])_\dagger([\![c]\!][\sigma | v : [\![e]\!]\sigma]) \\
[\![\textbf{while } b \textbf{ do } c]\!] &= \bigsqcup_{i=0}^{\infty} F^i\ \bot_{\Sigma\to\Sigma_\bot}
\end{aligned}
$$

donde

$$
F\ w\ \sigma = \begin{cases} w_*([\![c]\!]\sigma) & \text{si } [\![b]\!]\sigma \\ \sigma & \text{si no} \end{cases}
$$

Output

Los comportamientos posibles de un programa en un estado dado son ahora los siguientes:

- se genera una cantidad finita de output y luego "se cuelga"
- se genera una cantidad finita de output y luego termina
- se genera una cantidad finita de output y luego falla
- se genera una cantidad infinita de output

Output: sintaxis

Agregamos el comando

$$\langle comm \rangle ::= ! \langle intexp \rangle$$

Dominio Ω

Sea Ω = conjunto de estos comportamientos. El mismo será la unión de las siguientes familias. Las mismas corresponden en orden a las situaciones de arriba.

- $\{\langle n_1, .., n_k \rangle : \ n_i \in \mathbf{Z}\}$
- $\{\langle n_1, .., n_k, \sigma \rangle : \ n_i \in \mathbf{Z}, \ \sigma \in \Sigma\}$
- $\{\langle n_1, .., n_k, \langle \mathbf{abort}, \sigma \rangle \rangle : \ n_i \in \mathbf{Z}, \ \sigma \in \Sigma\}$
- $\{\langle n_1, .., n_k, ... \rangle : \ n_i \in \mathbf{Z}\}$

¿Cómo se define el orden en ω?

En Ω se define la relación $\omega \leq \omega'$ cuando ω es segmento inicial de ω'. Con esta relación Ω es un dominio, donde el mínimo es la secuencia vacía $\langle\rangle$. Las cadenas interesantes tienen supremo de la forma $\{\langle n_1, .., n_k, ...\rangle\}$.

Función semántica:

$$[\![\_]\!] \in \langle comm \rangle \to \Sigma \to \Omega$$

Ecuación semántica de la nueva construcción:

$$[\![!e]\!]_\sigma = \langle [\![e]\!]_\sigma, \sigma \rangle$$

Restantes Ecuaciones Semánticas

$$
\begin{aligned}
[\![\textbf{skip}]\!]_\sigma &= \sigma \\
[\![\textbf{fail}]\!]_\sigma &= \langle \textbf{abort}, \sigma \rangle \\
[\![v := e]\!]_\sigma &= [\sigma | v : [\![e]\!]_\sigma] \\
[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]_\sigma &= \begin{cases} [\![c_0]\!]_\sigma & \text{si } [\![b]\!]_\sigma \\ [\![c_1]\!]_\sigma & \text{si no} \end{cases} \\
[\![c_0; c_1]\!]_\sigma &= [\![c_1]\!]_*([\![c_0]\!]_\sigma) \\
[\![\textbf{catchin } c_0 \textbf{ with } c_1]\!]_\sigma &= [\![c_1]\!]_+([\![c_0]\!]_\sigma)
\end{aligned}
$$

Restantes Ecuaciones Semánticas

$$[\![\textbf{newvar v:= e in c}]\!]_\sigma = (\lambda\sigma' \in \Sigma.\ [\sigma' | v : \sigma\ v])_\dagger([\![c]\!][\sigma | v : [\![e]\!]_\sigma])$$

$$[\![\textbf{while } b \textbf{ do } c]\!] = \bigsqcup_{i=0}^{\infty} F^i\ \bot_{\Sigma\to\Omega}$$

donde

$$F\ w\ \sigma = \begin{cases} w_*([\![c]\!]_\sigma) & \text{si } [\![b]\!]_\sigma \\ \sigma & \text{si no} \end{cases}$$

Operadores

Sea $f \in \Sigma \to \Omega$, el operador $(\_)_*$ transfiere el control a la función sólo en caso de terminación normal:

$$f_* x = \begin{cases} \langle n_1, ..., n_k \rangle & x = \langle n_1, ..., n_k \rangle \\ \langle n_1, ..., n_k, f\sigma \rangle & x = \langle n_1, ..., n_k, \sigma \rangle \\ \langle n_1, ..., n_k, \langle \textbf{abort}, \sigma \rangle \rangle & x = \langle n_1, ..., n_k, \langle \textbf{abort}, \sigma \rangle \rangle \\ \langle n_1, ..., n_k, ... \rangle & x = \langle n_1, ..., n_k, ... \rangle \end{cases}$$

Operadores

Sea $f \in \Sigma \to \Omega$, el operador $(\_)_*$ transfiere el control a la función sólo en caso de terminación normal:

$$f_* x = \begin{cases} \langle n_1, ..., n_k \rangle & x = \langle n_1, ..., n_k \rangle \\ \langle n_1, ..., n_k, f\sigma \rangle & x = \langle n_1, ..., n_k, \sigma \rangle \\ \langle n_1, ..., n_k, \langle \textbf{abort}, \sigma \rangle \rangle & x = \langle n_1, ..., n_k, \langle \textbf{abort}, \sigma \rangle \rangle \\ \langle n_1, ..., n_k, ... \rangle & x = \langle n_1, ..., n_k, ... \rangle \end{cases}$$

Más compacto:

$$f_* x = \begin{cases} \langle\rangle & x = \langle\rangle \\ f\sigma & x = \langle \sigma \rangle \\ \langle \textbf{abort}, \sigma \rangle & x = \langle \textbf{abort}, \sigma \rangle \\ \langle n \rangle +\!\!+ f_* \omega & x = \langle n \rangle +\!\!+ \omega \end{cases}$$

Operadores

Sea $f \in \Sigma \to \Omega$, el operador $(\_)_+$ transfiere el control a la función sólo en caso de terminación abortiva:

$$f_+ x = \begin{cases} \langle\rangle & x = \langle\rangle \\ \langle\sigma\rangle & x = \langle\sigma\rangle \\ f\sigma & x = \langle \textbf{abort}, \sigma\rangle \\ \langle n\rangle +\!\!+ f_+\omega & x = \langle n\rangle +\!\!+ \omega \end{cases}$$

Operadores

Sea $f \in \Sigma \to \Sigma$, entonces la extensión

$$f_\dagger \in \Omega \to \Omega$$

se define

$$f_\dagger x = \begin{cases} \langle\rangle & x = \langle\rangle \\ \langle f\sigma \rangle & x = \langle \sigma \rangle \\ \langle \textbf{abort}, f\sigma \rangle & x = \langle \textbf{abort}, \sigma \rangle \\ \langle n \rangle +\!\!+ f_\dagger \omega & x = \langle n \rangle +\!\!+ \omega \end{cases}$$

Más sobre dominios: Producto de posets

Si $P_0, P_1, ..., P_{n-1}$ son órdenes parciales, entonces $P_0 \times P_1, ... \times P_{n-1}$ también lo es, donde el orden entre tuplas se define componente a componente.

Cadenas

$$\langle p_0^1, p_1^1, ..., p_{n-1}^1 \rangle \leq \langle p_0^2, p_1^2, ..., p_{n-1}^2 \rangle \leq \cdots \langle p_0^k, p_1^k, ..., p_{n-1}^k \rangle \leq \cdots$$

Componente a componente forman cadenas en los respectivos órdenes:

$$p_0^1 \leq p_0^2 \leq ... \leq p_0^k \leq ...$$
$$p_1^1 \leq p_1^2 \leq ... \leq p_1^k \leq ...$$
$$\vdots$$

Producto de dominios

Si $P_0, P_1, ..., P_{n-1}$ son dominios, entonces $P_0 \times P_1, ... \times P_{n-1}$ también lo es, donde el mínimo es la tupla que consiste del mínimo de cada uno de los dominios.

Todas las funciones sencillas usuales (proyecciones, constructor de tuplas, etc) son trivialmente continuas.

Uniones Disjuntas

Dados conjuntos $P_0, P_1, ..., P_{n-1}$ se define la unión disjunta

$$P_0 + P_1 + ... + P_{n-1} = \{\langle i, p\rangle : p \in P_i\}.$$

Se definen las inyecciones $\iota_i \in P_i \to P_0 + P1 + ... + P_{n-1}$ mediante $\iota_i p = \langle i, p\rangle$

Uniones Disjuntas de cpos

Dados órdenes parciales $P_0, P_1, ..., P_{n-1}$, entonces $P_0 + P_1 + ... + P_{n-1}$ es un órden parcial, donde el orden "no mezcla" los órdenes de los diferentes conjuntos dados:

$$\langle i, p\rangle \leq \langle j, q\rangle \iff i = j \ \wedge \ p \leq_i q$$

Cadenas Una cadena de $P_0 + P_1 + ... + P_{n-1}$ es una que proviene enteramente de uno de los P_i:

$$\langle i, p_1 \rangle \leq \langle i, p_2 \rangle \leq ... \leq \langle i, p_n \rangle \leq ...$$

donde los p_j son todos elementos de P_i.

Unión de dominios

Dados dominios $P_0, P_1, ..., P_{n-1}$, entonces $P_0 + P_1 + ... + P_{n-1}$ en general **no** es un dominio (sólo lo es en el caso trivial $n = 1$).

Todas las funciones sencillas usuales (inyecciones, análisis por casos, etc) son trivialmente continuas.

Dominios recursivos

Domino semántico para LIS con fallas y output

$$\Omega \quad \approx \quad (\Sigma' + \mathbf{Z} \times \Omega)_{\perp}$$

Dominios recursivos

Domino semántico para LIS con fallas y output

$$\Omega \quad \approx \quad (\Sigma' + \mathbf{Z} \times \Omega)_\perp$$

El símbolo $\approx$ significa isomorfismo, el cual está dado por las funciones continuas (una inversa de la otra):

$$\phi \in \Omega \to (\Sigma' + \mathbf{Z} \times \Omega)_\perp \qquad \psi \in (\Sigma' + \mathbf{Z} \times \Omega)_\perp \to \Omega$$

$$\phi x = \begin{cases} \bot & x = \langle\rangle \\ \iota_\bot(\iota_0 y) & x = \langle y\rangle \text{ con } y \in \Sigma' \\ \iota_\bot(\iota_1\langle n, \phi\omega\rangle) & x = \langle n\rangle +\!\!+ \omega \end{cases}$$

$$\psi x = \begin{cases} \langle\rangle & x = \bot \\ \langle y\rangle & x = \iota_\bot(\iota_0 y) \text{ con } y \in \Sigma' \\ \langle n\rangle +\!\!+ \psi\omega & x = \iota_\bot(\iota_1\langle n, \omega\rangle) \end{cases}$$

Notación

Para expresar las ecuaciones semánticas en adelante serán útiles las siguientes composiciones.

$$
\begin{aligned}
\iota_{term} &= \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{norm} \in \Sigma \to \Omega \\
\iota_{abort} &= \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{abnorm} \in \Sigma \to \Omega \\
\iota_{out} &= \psi \cdot \iota_{\perp} \cdot \iota_1 \in \mathbf{Z} \times \Sigma \to \Omega \\
\perp_{\Omega} &= \psi(\perp) \in \Omega
\end{aligned}
$$

$$\begin{array}{lll} & \overset{\iota_{norm}}{\longrightarrow} & \\ \Sigma & & \overset{\iota_0}{\longrightarrow} \\ & & \Sigma' \quad \longrightarrow \\ \Sigma & \longrightarrow & \end{array} \qquad \begin{array}{lll} & & \overset{\iota_\perp}{\longrightarrow} \\ \Sigma' + \mathbf{Z} \times \Omega & \longrightarrow & (\Sigma' + \mathbf{Z} \times \Omega)_\perp \end{array} \qquad \overset{\psi}{\longrightarrow} \Omega$$

$$\begin{array}{lll} & \overset{\iota_{abnorm}}{\longrightarrow} & \\ & & \overset{\iota_1}{\longrightarrow} \\ & & \mathbf{Z} \times \Omega \quad \longrightarrow \end{array}$$

Ecuaciones reformuladas

$$
\begin{aligned}
[\![\textbf{skip}]\!]_\sigma &= \iota_{term}\sigma \\
[\![\textbf{fail}]\!]_\sigma &= \iota_{abort}\sigma \\
[\![v := e]\!]_\sigma &= \iota_{term}[\sigma | v : [\![e]\!]_\sigma] \\
[\![!e]\!]_\sigma &= \iota_{out}([\![e]\!]_\sigma, \iota_{term}\sigma)
\end{aligned}
$$

Ecuaciones reformuladas

$$
\begin{aligned}
[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]_\sigma &= \begin{cases} [\![c_0]\!]_\sigma & \text{si } [\![b]\!]_\sigma \\ [\![c_1]\!]_\sigma & \text{si no} \end{cases} \\
[\![c_0; c_1]\!]_\sigma &= [\![c_1]\!]_*([\![c_0]\!]_\sigma) \\
[\![\textbf{catchin } c_0 \textbf{ with } c_1]\!]_\sigma &= [\![c_1]\!]_+([\![c_0]\!]_\sigma) \\
[\![\textbf{newvar v:= e in c}]\!]_\sigma &= (\lambda\sigma' \in \Sigma.\ [\sigma' | v : \sigma\ v])_\dagger([\![c]\!][\sigma | v : [\![e]\!]_\sigma]) \\
[\![\textbf{while } b \textbf{ do } c]\!] &= \bigsqcup_{i=0}^{\infty} F^i\ \bot_{\Sigma\to\Omega}
\end{aligned}
$$

donde

$$
F\ w\ \sigma = \begin{cases} w_*([\![c]\!]_\sigma) & \text{si } [\![b]\!]_\sigma \\ \iota_{term}\sigma & \text{si no} \end{cases}
$$

Operadores redefinidos

$$f_* x = \begin{cases} \bot_\Omega & x = \bot_\Omega \\ f\sigma & x = \iota_{term}\sigma \\ \iota_{abort}\sigma & x = \iota_{abort}\sigma \\ \iota_{out}(n, f_*\omega) & x = \iota_{out}(n, \omega) \end{cases}$$

$$f_+ x = \begin{cases} \bot_\Omega & x = \bot_\Omega \\ \iota_{term}\sigma & x = \iota_{term}\sigma \\ f\sigma & x = \iota_{abort}\sigma \\ \iota_{out}(n, f_+\omega) & x = \iota_{out}(n, \omega) \end{cases}$$

Operadores redefinidos

$$f_\dagger x = \begin{cases} \perp_\Omega & x = \perp_\Omega \\ \iota_{term}(f\sigma) & x = \iota_{term}\sigma \\ \iota_{abort}(f\sigma) & x = \iota_{abort}\sigma \\ \iota_{out}(n, f_\dagger\omega) & x = \iota_{out}(n, \omega) \end{cases}$$

Input

Sintaxis abstracta:

$$\langle comm \rangle \ ::= ? \ \langle var \rangle$$

Domino semántico:

$$\Omega \quad \approx \quad (\Sigma' + \mathbf{Z} \times \Omega + \mathbf{Z} \to \Omega)_\perp$$

Isomorfismos:

$$\phi \ \in \ \Omega \to (\Sigma' + \mathbf{Z} \times \Omega + \mathbf{Z} \to \Omega)_\perp \qquad \psi \ \in \ (\Sigma' + \mathbf{Z} \times \Omega + \mathbf{Z} \to \Omega)_\perp \to \Omega$$

Composiciones útiles

$$
\begin{aligned}
\iota_{term} &= \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{norm} \in \Sigma \to \Omega \\
\iota_{abort} &= \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{abnorm} \in \Sigma \to \Omega \\
\iota_{out} &= \psi \cdot \iota_{\perp} \cdot \iota_1 \in \mathbf{Z} \times \Sigma \to \Omega \\
\iota_{in} &= \psi \cdot \iota_{\perp} \cdot \iota_2 \in (\mathbf{Z} \to \Sigma) \to \Omega \\
\perp_{\Omega} &= \psi(\perp) \in \Omega
\end{aligned}
$$

$$\Sigma \xrightarrow{\iota_{norm}} \Sigma' \xrightarrow{\iota_0}$$

$$\Sigma \xrightarrow{\iota_{abnorm}} \mathbf{Z} \times \Omega \xrightarrow{\iota_1}$$

$$\mathbf{Z} \to \Omega \xrightarrow{\iota_2}$$

$$\Sigma' + \mathbf{Z} \times \Omega \xrightarrow{\iota_\perp} (\Sigma' + \mathbf{Z} \times \Omega)_\perp \xrightarrow{\psi} \Omega$$

Ecuación semántica para el input:

$$[\![?v]\!]_\sigma \quad = \quad \iota_{in}(\lambda n \in \mathbf{Z}.\ \iota_{term}[\sigma | v : n])$$

Observación: Las restantes ecuaciones semánticas no se alteran, sólo es necesario actualizar las funciones de transferencia de control.

$$f_* x = \begin{cases} \bot_\Omega & x = \bot_\Omega \\ f\sigma & x = \iota_{term}\sigma \\ \iota_{abort}\sigma & x = \iota_{abort}\sigma \\ \iota_{out}(n, f_*\omega) & x = \iota_{out}(n, \omega) \\ \iota_{in}(f_* \cdot g) & x = \iota_{in} g \end{cases}$$

$$f_+ x = \begin{cases} \bot_\Omega & x = \bot_\Omega \\ \iota_{term}\sigma & x = \iota_{term}\sigma \\ f\sigma & x = \iota_{abort}\sigma \\ \iota_{out}(n, f_+\omega) & x = \iota_{out}(n, \omega) \\ \iota_{in}(f_+ \cdot g) & x = \iota_{in} g \end{cases}$$

$$f_\dagger x = \begin{cases} \perp_\Omega & x = \perp_\Omega \\ \iota_{term}(f\sigma) & x = \iota_{term}\sigma \\ \iota_{abort}(f\sigma) & x = \iota_{abort}\sigma \\ \iota_{out}(n, f_\dagger\omega) & x = \iota_{out}(n, \omega) \\ \iota_{in}(f_\dagger \cdot g) & x = \iota_{in}g \end{cases}$$

Semántica de transiciones

Describe cómo se realiza el cómputo.

$$\langle \mathsf{x} := 1; \mathsf{y} := 2 * \mathsf{y}, \sigma \rangle \rightarrow \langle \mathsf{y} := 2 * \mathsf{y}, [\sigma | \mathsf{x} : 1] \rangle \rightarrow [\sigma | \mathsf{x} : 1 | \mathsf{y} : 2 * \sigma \mathsf{y}]$$

La relación $\rightarrow$ describe un paso de ejecución (*small-step semantics*). En cada paso se pasa de una *configuración* a otra.

Conjunto de configuraciones:

$$\Gamma = \Gamma_t \cup \Gamma_n$$

$$\begin{array}{lcll} \Gamma_t & = & \Sigma & \text{(configuraciones terminales)} \\ \Gamma_n & = & \langle comm \rangle \times \Sigma & \text{(configuraciones no terminales)} \end{array}$$

Definición axiomática de $\rightarrow$

$$\frac{}{\langle \textbf{skip}, \sigma \rangle \rightarrow \sigma}$$

$$\frac{}{\langle v := e, \sigma \rangle \rightarrow [\sigma | v : [\![e]\!]\sigma]}$$

$$\frac{\langle c_0, \sigma\rangle \to \sigma'}{\langle c_0; c_1, \sigma\rangle \to \langle c_1, \sigma'\rangle}$$

$$\frac{\langle c_0, \sigma\rangle \to \langle c_0', \sigma'\rangle}{\langle c_0; c_1, \sigma\rangle \to \langle c_0'; c_1, \sigma'\rangle}$$

Note como estas reglas permiten construir un paso intermedio de ejecución, como por ejemplo el primer paso de la ejecución de arriba:

$$\frac{\langle \mathsf{x} := 1, \sigma\rangle \to [\sigma|\mathsf{x} : 1]}{\langle \mathsf{x} := 1; \mathsf{y} := 2 * \mathsf{y}, \sigma\rangle \to \langle \mathsf{y} := 2 * \mathsf{y}, [\sigma|\mathsf{x} : 1]\rangle}$$

$$\frac{([\![e]\!]\sigma = V)}{\langle \textbf{if } e \textbf{ then } c \textbf{ else } c', \sigma\rangle \to \langle c, \sigma\rangle}$$

$$\frac{([\![e]\!]\sigma = F)}{\langle \textbf{if } e \textbf{ then } c \textbf{ else } c', \sigma\rangle \to \langle c', \sigma\rangle}$$

$$\frac{([\![e]\!]\sigma = F)}{\langle \textbf{while } e \textbf{ do } c, \sigma\rangle \to \sigma}$$

$$\frac{([\![e]\!]\sigma = T)}{\langle \textbf{while } e \textbf{ do } c, \sigma\rangle \to \langle c : \textbf{ while } e \textbf{ do } c, \sigma\rangle}$$

$$\frac{\langle c, [\sigma | v : [\![e]\!]_\sigma] \rangle \to \sigma'}{\langle \textbf{newvar } v := e \textbf{ in } c, \sigma \rangle \to [\sigma' | v : \sigma v]}$$

$$\frac{\langle c, [\sigma | v : [\![e]\!]_\sigma] \rangle \to \langle c', \sigma' \rangle}{\langle \textbf{newvar } v := e \textbf{ in } c, \sigma \rangle \to \langle \textbf{newvar } v := \sigma' v \textbf{ in } c', [\sigma' | v : \sigma v] \rangle}$$

Determinismo y continuación

Determinismo: $\rightarrow$ define una función: ninguna configuración no terminal puede mover (en un sólo paso) hacia más de una configuración

Continuación: ninguna configuración no terminal puede mover hacia menos de una configuración (no se traba).

Ejecución

Por *ejecución* entendemos una secuencia $c_0 \to c_1 \to c_2 \to \ldots$ maximal, esto es, que no puede prolongarse más de lo que está. Dicha ejecución es infinita o termina en una configuración terminal σ.

Si la ejecución es infinita decimos que c_0 *diverge* y escribimos $c_0 \uparrow$.

$$\{\!\{c\}\!\}\sigma = \begin{cases} \bot & \text{si } \langle c, \sigma \rangle \uparrow \\ \sigma' & \text{si existe } \sigma' \text{ tal que } \langle c, \sigma \rangle \to^* \sigma' \end{cases}$$

Corrección de la semántica operacional

Lema 1

1. Si $\langle c_0, \sigma\rangle \to^* \sigma'$, entonces $\langle c_0; c_1, \sigma\rangle \to^* \langle c_1, \sigma'\rangle$
2. Si $\langle c, [\sigma | v : [\![e]\!]\sigma]\rangle \to^* \sigma'$, entonces

$$\langle \textbf{newvar } v := e \textbf{ in } c, \sigma\rangle \to^* [\sigma' | v : \sigma v].$$

3. Si $\langle c, [\sigma | v : [\![e]\!]\sigma]\rangle \to^* \langle c', \sigma'\rangle$, entonces

$$\langle \textbf{newvar } v := e \textbf{ in } c, \sigma\rangle \to^* \langle \textbf{newvar } v := \sigma' x \textbf{ in } c', [\sigma' | v : \sigma v]\rangle$$

Corrección de la semántica operacional

Lema 2

1. $\langle c, \sigma \rangle \to \sigma' \Longrightarrow [\![c]\!]\sigma = \sigma'$.
2. $\langle c, \sigma \rangle \to \langle c', \sigma' \rangle \Longrightarrow [\![c]\!]\sigma = [\![c']\!]\sigma'$.

Lema 3 $[\![c]\!]\sigma = \sigma' \Longrightarrow \langle c, \sigma \rangle \to^* \sigma'$

Teorema Para todo comando c se tiene $\{\!\{c\}\!\} = [\![c]\!]$.

Semántica operacional de las fallas

Configuraciones terminales:

$$\Gamma_t \quad = \quad \Sigma \cup \{\textbf{abort}\} \times \Sigma$$

$$\frac{}{\langle \textbf{fail}, \sigma \rangle \rightarrow \langle \textbf{abort}, \sigma \rangle}$$

$$\frac{\langle c_0, \sigma \rangle \rightarrow \langle \textbf{abort}, \sigma' \rangle}{\langle c_0 ; c_1, \sigma \rangle \rightarrow \langle \textbf{abort}, \sigma' \rangle}$$

Semántica operacional de las fallas

$$\frac{\langle c_0, \sigma\rangle \to \sigma'}{\langle \textbf{catchin } c_0 \textbf{ with } c_1, \sigma\rangle \to \sigma'}$$

$$\frac{\langle c_0, \sigma\rangle \to \langle \textbf{abort}, \sigma'\rangle}{\langle \textbf{catchin } c_0 \textbf{ with } c_1, \sigma\rangle \to \langle c_1, \sigma'\rangle}$$

$$\frac{\langle c_0, \sigma\rangle \to \langle c_0', \sigma'\rangle}{\langle \textbf{catchin } c_0 \textbf{ with } c_1, \sigma\rangle \to \langle \textbf{catchin } c_0' \textbf{ with } c_1, \sigma'\rangle}$$

Semántica operacional de las fallas

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

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

Corrección de la semántica operacional de las fallas

La relación $\rightarrow$ sigue siendo una función en el lenguaje con fallas, toda configuración γ tiene una única ejecución que puede ser infinita (γ diverge) o terminar en una configuración terminal que puede ser de la forma σ o $\langle \textbf{abort}, \sigma \rangle$.

Se puede definir:

$$\{\!\{c\}\!\}\sigma = \begin{cases} \bot & \text{si } \langle c, \sigma \rangle \uparrow \\ \sigma' & \text{si existe } \sigma' \text{ tal que } \langle c, \sigma \rangle \rightarrow^* \sigma' \\ \langle \textbf{abort}, \sigma' \rangle & \text{si existe } \sigma' \text{ tal que } \langle c, \sigma \rangle \rightarrow^* \langle \textbf{abort}, \sigma' \rangle \end{cases}$$

y obtendremos de manera similar a LIS que para todo comando c se tiene $\{\!\{c\}\!\} = [\![c]\!]$.