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

Un Lenguaje Imperativo Simple

LIS

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

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

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

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

Características de LIS

- Hay sólo dos tipos de expresiones: enteras y booleanas

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.

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:

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

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

Significado de los comandos de LIS (sin iteración)

Todos los comandos terminan su ejecución (no necesitamos $\perp_\Sigma$), y el único resultado posible es un estado.

Significado de los comandos de LIS (sin iteración)

Todos los comandos terminan su ejecución (no necesitamos $\perp_\Sigma$), y el único resultado posible es un estado.

Funciones semánticas:

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

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

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

Primeras ecuaciones semánticas

$$[\![\textbf{skip}]\!]\sigma \quad = \quad \sigma$$

Primeras ecuaciones semánticas

$$
\begin{aligned}
[\![\textbf{skip}]\!]\sigma &= \sigma \\
[\![v := e]\!]\sigma &= [\sigma | v : [\![e]\!]\sigma]
\end{aligned}
$$

Primeras ecuaciones semánticas

$$
\begin{aligned}
[\![\mathbf{skip}]\!]\sigma &= \sigma \\
[\![v := e]\!]\sigma &= [\sigma | v : [\![e]\!]\sigma] \\
[\![c_0 ; c_1]\!]\sigma &= [\![c_1]\!]([\![c_0]\!]\sigma)
\end{aligned}
$$

Primeras ecuaciones semánticas

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

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:

$$[\![x := x - 1]\!]_\sigma \quad = \quad [\sigma | x : [\![x - 1]\!]_\sigma]$$

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:

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

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:

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

Semántica de **newvar**

Función "restauración de v según σ":

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

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”:

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

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

Notación lambda para la función restauración:

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

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

Revisión: ¿Qué función computa haskell?

```
g ::  Int -> Int
g n = if n == 0 then 0
       else if n == 1 then 1
            else g (n-2)
```

Revisión: ¿Qué función computa haskell?

```
g ::  Int -> Int
g n = if n == 0 then 0
       else if n == 1 then 1
            else g (n-2)
```

Es la menor solución en $\mathbb{Z} \to \mathbb{Z}_\perp$ de la ecuación recursiva:

$$f\ n = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ f(n-2) & \text{caso contrario (c.c.)} \end{cases} \qquad (ER)$$

Revisión: ¿Qué función computa haskell?

```
g ::  Int -> Int
g n = if n == 0 then 0
       else if n == 1 then 1
            else g (n-2)
```

Es la menor solución en $\mathbb{Z} \to \mathbb{Z}_\perp$ de la ecuación recursiva:

$$f\ n = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ f(n-2) & \text{caso contrario (c.c.)} \end{cases} \qquad (ER)$$

La ecuación ER define una familia de funciones.

Una función es solución de ER si y sólo si es punto fijo de:

$$F\ f\ n = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ f\ (n-2) & \text{si } n \notin \{0,1\} \end{cases}$$

Una función es solución de ER si y sólo si es punto fijo de:

$$F\ f\ n = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ f\ (n-2) & \text{si } n \notin \{0,1\} \end{cases}$$

El menor punto fijo es:

$$\sup_{\mathbb{Z}\to\mathbb{Z}_\perp} (\{F^i\ \perp_{\mathbb{Z}\to\mathbb{Z}_\perp} | i \geq 0\})$$

donde $\perp_{\mathbb{Z}\to\mathbb{Z}_\perp}$ es el elemento mínimo de $\mathbb{Z} \to \mathbb{Z}_\perp$, es decir, la función que devuelve siempre $\perp$.

Una función es solución de ER si y sólo si es punto fijo de:

$$F\ f\ n = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ f\ (n-2) & \text{si } n \notin \{0,1\} \end{cases}$$

El menor punto fijo es:

$$\sup_{\mathbb{Z}\to\mathbb{Z}_\perp} (\{F^i\ \perp_{\mathbb{Z}\to\mathbb{Z}_\perp} | i \geq 0\})$$

donde $\perp_{\mathbb{Z}\to\mathbb{Z}_\perp}$ es el elemento mínimo de $\mathbb{Z} \to \mathbb{Z}_\perp$, es decir, la función que devuelve siempre $\perp$.

Fin de la revisión

Dificultades para dar significado a la iteración

El comando

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

debe satisfacer la propiedad:

$$[\![\textbf{while } b \textbf{ do } c]\!]\sigma \quad = \quad \text{if } [\![b]\!]\sigma \text{ then } [\![c; \textbf{while } b \textbf{ do } c]\!]\sigma \text{ else } [\![\textbf{skip}]\!]\sigma$$

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 en $\Sigma \to \Sigma_\perp$?

- 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:

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

¿Define esta propiedad una función en $\Sigma \to \Sigma_\perp$?

- 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:

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

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:

$$f_{\perp\!\!\!\perp}\sigma \;=\; f\,\sigma \qquad\qquad f_{\perp\!\!\!\perp}\,\perp = \perp$$

Luego la expresión (2) puede ser corregida poniendo

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

Semántica del **while**

Dificultad principal:

$$[\![\textbf{while}\ b\ \textbf{do}\ c]\!]\sigma = \begin{cases} [\![\textbf{while}\ b\ \textbf{do}\ c]\!]_{\perp\!\!\!\perp}([\![c]\!]\sigma) & \text{si } [\![b]\!]\sigma \\ \sigma & \text{si } \neg[\![b]\!]\sigma \end{cases} \qquad (W)$$

no es dirigida por sintaxis

La propiedad W no es dirigida por sintaxis

La propiedad del **while** no puede ser tomada como definición.

Pero podemos tener la certeza de:

- el significado de $[\![$**while** b **do** $c]\!]$ es una función $\omega \in \Sigma \to \Sigma_{\perp}$

La propiedad W 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 del **while**

Sea

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

Semántica del **while**

Sea

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

entonces, la ecuación del **while** queda

$$\llbracket \textbf{while}\ b\ \textbf{do}\ c \rrbracket \sigma = F\ \llbracket \textbf{while}\ b\ \textbf{do}\ c \rrbracket\ \sigma$$

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:

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

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:

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

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

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.

Semántica denotacional de LIS

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

Semántica denotacional de LIS

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

$$\begin{array}{rcl} [\![\textbf{skip}]\!]\sigma & = & \sigma \\ [\![v := e]\!]\sigma & = & [\sigma | v : [\![e]\!]\sigma] \end{array}$$

Semántica denotacional de LIS

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

$$\begin{aligned}
[\![\mathbf{skip}]\!]\sigma &= \sigma \\
[\![v := e]\!]\sigma &= [\sigma | v : [\![e]\!]\sigma] \\
[\![c_0 ; c_1]\!]\sigma &= [\![c_1]\!]_{\perp\!\!\!\perp}([\![c_0]\!]\sigma)
\end{aligned}$$

Semántica denotacional de LIS

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

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

Semántica denotacional de LIS

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

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

Semántica denotacional de LIS

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

donde

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

¿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 $w\ \sigma$

¿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** $x := x - 2$.

Ejemplo

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

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

donde

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

Ejemplo

$\textbf{while } x \neq 0 \wedge x \neq 1 \textbf{ do } x := x - 2.$

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

donde

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

Entonces:

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

Se puede comprobar que

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

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

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

Variables asignables

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

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):

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

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

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:

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

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

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 $[\![c]\!]\sigma'$.

Por ejemplo, si al programa $c = (\mathrm{x} := \mathrm{x} - 1; \mathrm{y} := 2 * \mathrm{y})$ 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 $\mathrm{x} := \mathrm{x} - 1; \mathrm{y} := 2 * \mathrm{y}$.

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

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

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

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

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

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

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