Lenguajes y Compiladores
Lenguaje Imperativo: ejemplo y teoremas

Miguel Pagano

12 de abril de 2023

Repaso

Extensión estrictas
Si $f : A \to B$, entonces definimos su ***extensión estricta***, $f_\perp : A_\perp \to B_\perp$

$$f_\perp\, x = \begin{cases} \perp & \text{si } x = \perp \\ f\, a & \text{si } x = a \end{cases}$$

Extensión estrictas
Si $f : A \to B$, entonces definimos su ***extensión estricta***, $f_\perp : A_\perp \to B_\perp$

$$f_\perp\, x = \begin{cases} \perp & \text{si } x = \perp \\ f\, a & \text{si } x = a \end{cases}$$

Extensiones estrictas
Si $f : P \to D$, entonces definimos su ***extensión estricta***, $f_{\perp\!\!\!\perp} : P_\perp \to D$

$$f_{\perp\!\!\!\perp}\, x = \begin{cases} \perp & \text{si } x = \perp \\ f\, a & \text{si } x = a \end{cases}$$

Lenguaje de Programación

Comandos

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

Expresiones

$$\langle natconst \rangle ::= \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \ldots$$

$$\langle boolconst \rangle ::= \mathbf{true} \mid \mathbf{false}$$

$$\begin{array}{rl} \langle intexp \rangle ::= & \langle natconst \rangle \\ \mid & \langle var \rangle \\ \mid & \text{-}\, \langle intexp \rangle \\ \mid & \langle intexp \rangle \oplus \langle intexp \rangle \end{array}$$

$$\oplus \in \{+, -, *, /, \%, \mathbf{rem}\}$$

$$\begin{array}{rl} \langle boolexp \rangle ::= & \langle boolconst \rangle \\ \mid & \neg\, \langle boolexp \rangle \\ \mid & \langle intexp \rangle \otimes \langle intexp \rangle \\ \mid & \langle boolexp \rangle \circledwedge \langle boolexp \rangle \end{array}$$

$$\otimes \in \{<, \leqslant, =, \neq, \geqslant, >\}$$

$$\circledwedge \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$$

Características de LIS

- Hay sólo dos tipos de expresiones: enteras y booleanas

Características de LIS

- Hay sólo dos tipos de expresiones: enteras y booleanas
- Todas las funciones primitivas (incluida la división) son funciones totales.

Características de LIS

- Hay sólo dos tipos de expresiones: enteras y booleanas
- Todas las funciones primitivas (incluida la división) son funciones totales.
- Como en la lógica de predicados, sólo hay variables enteras:

$$\Sigma = \langle var \rangle \to \mathbb{Z}$$

Características de LIS

- Hay sólo dos tipos de expresiones: enteras y booleanas
- Todas las funciones primitivas (incluida la división) son funciones totales.
- Como en la lógica de predicados, sólo hay variables enteras:

$$\Sigma = \langle var \rangle \to \mathbb{Z}$$

- El significado de una expresión entera (booleana), fijado un estado, es un entero (booleano).

Características de LIS

- Hay sólo dos tipos de expresiones: enteras y booleanas
- Todas las funciones primitivas (incluida la división) son funciones totales.
- Como en la lógica de predicados, sólo hay variables enteras:

$$\Sigma = \langle var \rangle \to \mathbb{Z}$$

- El significado de una expresión entera (booleana), fijado un estado, es un entero (booleano).
- El significado de un comando ***que termina***, fijado un estado, es un estado.

Características de LIS

- Hay sólo dos tipos de expresiones: enteras y booleanas
- Todas las funciones primitivas (incluida la división) son funciones totales.
- Como en la lógica de predicados, sólo hay variables enteras:

$$\Sigma = \langle var \rangle \to \mathbb{Z}$$

- El significado de una expresión entera (booleana), fijado un estado, es un entero (booleano).
- El significado de un comando ***que termina***, fijado un estado, es un estado.
- ¿Y el signifcado de un comando que ***NO*** termina?

Funciones semánticas

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

Significado de los comandos de LIS

Funciones semánticas

$$\llbracket \_ \rrbracket^{intexp} \in \langle intexp \rangle \rightarrow (\Sigma \rightarrow \mathbb{Z})$$
$$\llbracket \_ \rrbracket^{boolexp} \in \langle boolexp \rangle \rightarrow (\Sigma \rightarrow \{V, F\})$$

Significado de los comandos de LIS

Funciones semánticas

$$
\begin{aligned}
[\![\_]\!]^{intexp} &\in \langle intexp \rangle \to (\Sigma \to \mathbb{Z}) \\
[\![\_]\!]^{boolexp} &\in \langle boolexp \rangle \to (\Sigma \to \{V, F\}) \\
[\![\_]\!]^{comm} &\in \langle comm \rangle \to (\Sigma \to \Sigma_{\perp})
\end{aligned}
$$

Funciones semánticas

$$\llbracket \_ \rrbracket^{intexp} \in \langle intexp \rangle \to (\Sigma \to \mathbb{Z})$$
$$\llbracket \_ \rrbracket^{boolexp} \in \langle boolexp \rangle \to (\Sigma \to \{V, F\})$$
$$\llbracket \_ \rrbracket^{comm} \in \langle comm \rangle \to (\Sigma \to \Sigma_\perp)$$

Observación: en el codominio de $\llbracket \_ \rrbracket^{comm}$ aparece $\Sigma_\perp$ para dar cuenta de la no terminación.

Equivalencia de términos

Dados e, e' en la misma categoría sintáctica, se dice que e es equivalente a e', que escribimos como $e \equiv e'$, si y solo si

$$[\![e]\!] = [\![e']\!]$$

Equivalencia de Expresiones y Comandos

Equivalencia de términos
Dados e, e' en la misma categoría sintáctica, se dice que e es equivalente a e', que escribimos como $e \equiv e'$, si y solo si

$$[\![e]\!] = [\![e']\!]$$

En nuestro caso, es equivalente a decir que para todo $\sigma \in \Sigma$,

$$[\![e]\!]\sigma = [\![e']\!]\sigma$$

Equivalencia de Expresiones y Comandos

Equivalencia de términos
Dados e, e' en la misma categoría sintáctica, se dice que e es equivalente a e', que escribimos como $e \equiv e'$, si y solo si

$$[\![e]\!] = [\![e']\!]$$

En nuestro caso, es equivalente a decir que para todo $\sigma \in \Sigma$,

$$[\![e]\!]\sigma = [\![e']\!]\sigma$$

Por ejemplo:

Equivalencia de términos
Dados e, e' en la misma categoría sintáctica, se dice que e es equivalente a e', que escribimos como $e \equiv e'$, si y solo si

$$[\![e]\!] = [\![e']\!]$$

En nuestro caso, es equivalente a decir que para todo $\sigma \in \Sigma$,

$$[\![e]\!]\sigma = [\![e']\!]\sigma$$

Por ejemplo:

$$\mathbf{true} \equiv x = x$$

Equivalencia de Expresiones y Comandos

Equivalencia de términos
Dados e, e' en la misma categoría sintáctica, se dice que e es equivalente a e', que escribimos como $e \equiv e'$, si y solo si

$$[\![e]\!] = [\![e']\!]$$

En nuestro caso, es equivalente a decir que para todo $\sigma \in \Sigma$,

$$[\![e]\!]\sigma = [\![e']\!]\sigma$$

Por ejemplo:

$$\textbf{true} \equiv x = x$$

$$2 * x \equiv x + x$$

Equivalencia de términos
Dados e, e' en la misma categoría sintáctica, se dice que e es equivalente a e', que escribimos como $e \equiv e'$, si y solo si

$$[\![e]\!] = [\![e']\!]$$

En nuestro caso, es equivalente a decir que para todo $\sigma \in \Sigma$,

$$[\![e]\!]\sigma = [\![e']\!]\sigma$$

Por ejemplo:

$$\textbf{true} \equiv x = x$$

$$2 * x \equiv x + x$$

$$x := 0; y := 1 \equiv y := 1; x := 0$$

Ecuaciones Semánticas de Comandos

Comandos sencillos

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

Comandos sencillos

$$[\![\mathbf{skip}]\!]\,\sigma = \sigma$$

$$[\![v := e]\!]\,\sigma = [\sigma \,|\, v : [\![e]\!]\,\sigma]$$

Comandos sencillos

$$
\begin{aligned}
[\![\mathbf{skip}]\!]\,\sigma &= \sigma \\
[\![v := e]\!]\,\sigma &= [\sigma \,|\, v : [\![e]\!]\,\sigma] \\
[\![\mathbf{if}\; b \;\mathbf{then}\; c \;\mathbf{else}\; c']\!]\,\sigma &= \begin{cases} [\![c]\!]\,\sigma & \text{si } [\![b]\!]\,\sigma \\ [\![c']\!]\,\sigma & \text{si } \neg[\![b]\!]\,\sigma \end{cases}
\end{aligned}
$$

Ecuaciones Semánticas de Comandos

Comandos sencillos

$$
\begin{aligned}
[\![\mathbf{skip}]\!]\,\sigma &= \sigma \\
[\![v := e]\!]\,\sigma &= [\sigma \,|\, v : [\![e]\!]\,\sigma] \\
[\![\mathbf{if}\; b\; \mathbf{then}\; c\; \mathbf{else}\; c']\!]\,\sigma &= \begin{cases} [\![c]\!]\,\sigma & \text{si } [\![b]\!]\,\sigma \\ [\![c']\!]\,\sigma & \text{si } \neg[\![b]\!]\,\sigma \end{cases}
\end{aligned}
$$

Claramente **if** ... es un caso recursivo, pero no problemático.

Composición

La ecuación obvia

$$[\![c_0; c_1]\!]\,\sigma = [\![c_1]\!]\,([\![c_0]\!]\,\sigma)$$

no tiene en cuenta que c_0 se puede colgar.

Composición

La ecuación obvia

$$[\![c_0; c_1]\!]\,\sigma = [\![c_1]\!]\,([\![c_0]\!]\,\sigma)$$

no tiene en cuenta que c_0 se puede colgar.

De hecho, nuestro dominio semántico está bien pensado porque hay un error de tipos.

Composición

La ecuación obvia

$$[\![c_0; c_1]\!]\,\sigma = [\![c_1]\!]\,([\![c_0]\!]\,\sigma)$$

no tiene en cuenta que c_0 se puede colgar.

De hecho, nuestro dominio semántico está bien pensado porque hay un error de tipos.

$$[\![c_0; c_1]\!]\,\sigma = [\![c_1]\!]_{\perp\!\!\!\perp}([\![c_0]\!]\,\sigma)$$

Bloque local

Después de ejecutar el cuerpo hay que restaurar el valor de la variable.

$$\begin{array}{rcl} rest_{v,\sigma} & : & \Sigma \to \Sigma \\ rest_{v,\sigma}(\sigma') & = & [\sigma' \,|\, v : \sigma v] \end{array}$$

Acá σ y v son parámetros fijos; el argumento es σ'.

Bloque local

Después de ejecutar el cuerpo hay que restaurar el valor de la variable.

$$\begin{array}{rcl} rest_{v,\sigma} & : & \Sigma \to \Sigma \\ rest_{v,\sigma}(\sigma') & = & [\sigma' \,|\, v : \sigma v] \end{array}$$

Acá σ y v son parámetros fijos; el argumento es σ'.

$$[\![\mathbf{newvar}\, v \;:=\; e \;\mathbf{in}\, c]\!]\, \sigma = (rest_{v,\sigma})_{\perp\!\!\!\perp}([\![c]\!]\, [\sigma|v : [\![e]\!]\, \sigma])$$

Bloque local

Después de ejecutar el cuerpo hay que restaurar el valor de la variable.

$$
\begin{array}{rcl}
rest_{v,\sigma} & : & \Sigma \to \Sigma \\
rest_{v,\sigma}(\sigma') & = & [\sigma' \,|\, v : \sigma v]
\end{array}
$$

Acá σ y v son parámetros fijos; el argumento es σ'.

$$[\![\mathbf{newvar}\, v \; := \; e \; \mathbf{in}\, c]\!]\, \sigma = (rest_{v,\sigma})_{\perp\!\!\!\perp}([\![c]\!]\, [\sigma|v : [\![e]\!]\, \sigma])$$

Con notación lambda:

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

Ecuaciones Semánticas de Comandos

Ciclo

Evaluamos la guarda. Si es falsa, se terminó el programa.

Si es verdadera, ejecutamos el cuerpo y luego el ciclo entero.

Ciclo

Evaluamos la guarda. Si es falsa, se terminó el programa.
Si es verdadera, ejecutamos el cuerpo y luego el ciclo entero.

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

Ciclo

Evaluamos la guarda. Si es falsa, se terminó el programa.
Si es verdadera, ejecutamos el cuerpo y luego el ciclo entero.

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

Problema: no es dirigida por sintaxis. Solución: definir

$$F_{b,c} : (\Sigma \to \Sigma_\perp) \to (\Sigma \to \Sigma_\perp)$$

y encontrar su menor punto fijo.

Ciclo

Evaluamos la guarda. Si es falsa, se terminó el programa.
Si es verdadera, ejecutamos el cuerpo y luego el ciclo entero.

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

Problema: no es dirigida por sintaxis. Solución: definir

$$F_{b,c} \colon (\Sigma \to \Sigma_\perp) \to (\Sigma \to \Sigma_\perp)$$

y encontrar su menor punto fijo.

$$F_{b,c}\, f\, \sigma = \begin{cases} \sigma & \text{si } \neg[\![b]\!]\sigma \\ f_{\perp\!\!\!\perp}([\![c]\!]\, \sigma) & \text{si } [\![b]\!]\sigma \end{cases}$$

Ejemplo: Exponenciación

Especificación
Quiero un programa c que calcule x^y, asumiendo que $y \geqslant 0$.

Exponenciación: Enunciado

Especificación
Quiero un programa c que calcule x^y, asumiendo que $y \geqslant 0$.

Volviendo a primer año:

$$\{Y \geqslant 0\}$$
$$c$$
$$\{e = X^Y\}$$

Exponenciación: Enunciado

Especificación
Quiero un programa c que calcule x^y, asumiendo que $y \geqslant 0$.

Volviendo a primer año:

$$\{Y \geqslant 0\}$$
$$c$$
$$\{e = X^Y\}$$

Volviendo a quinto $[\![c]\!]\sigma\, e = \sigma\, x^{\sigma\, y}$ si $\sigma\, y \geqslant 0$.

Solución obvia

$$\mathtt{e} := 1\,; \textbf{while}\ \underbrace{\mathtt{y} > 0}_{b}\ \textbf{do}\ \underbrace{\mathtt{e} := \mathtt{e} * \mathtt{x}\,; \mathtt{y} := \mathtt{y} - 1}_{body}$$

Exponenciación: Programa

Solución obvia

$$\mathtt{e} := 1\,; \textbf{while}\, \underbrace{\mathtt{y} > 0}_{b}\; \textbf{do}\, \underbrace{\mathtt{e} := \mathtt{e} * \mathtt{x}\,; \mathtt{y} := \mathtt{y} - 1}_{body}$$

Cálculo de semántica

- $[\![\mathtt{e} := 1]\!]\sigma$.

Exponenciación: Programa

Solución obvia

$$\mathtt{e} := 1\,;\mathbf{while}\,\underbrace{\mathtt{y} > 0}_{b}\ \mathbf{do}\,\underbrace{\mathtt{e} := \mathtt{e} * \mathtt{x}\,;\,\mathtt{y} := \mathtt{y} - 1}_{body}$$

Cálculo de semántica

- $[\![\mathtt{e} := 1]\!]\sigma$.
- $[\![body]\!]\sigma$.

Solución obvia

$$\texttt{e := 1}\,;\,\textbf{while}\,\underbrace{\texttt{y} > 0}_{b}\ \textbf{do}\,\underbrace{\texttt{e := e} * \texttt{x}\,;\,\texttt{y := y} - 1}_{body}$$

Cálculo de semántica

- $[\![\texttt{e := 1}]\!]\sigma$.
- $[\![body]\!]\sigma$.
- $F^{i}_{b,body} \perp \sigma$.

Exponenciación: Programa

Solución obvia

$$\mathtt{e} := 1\,;\,\mathbf{while}\,\underbrace{\mathtt{y} > 0}_{b}\;\mathbf{do}\,\underbrace{\mathtt{e} := \mathtt{e} * \mathtt{x}\,;\,\mathtt{y} := \mathtt{y} - 1}_{body}$$

Cálculo de semántica

- $[\![\mathtt{e} := 1]\!]\sigma$.
- $[\![body]\!]\sigma$.
- $F^{i}_{b,body}\perp\sigma$.
- $[\![\mathbf{while}\, b\;\mathbf{do}\; body]\!]\sigma$.

Cálculo de semántica de ciclos

1. Hallar la expresión más sencilla posible para *body*.

Cálculo de semántica de ciclos

1. Hallar la expresión más sencilla posible para $body$.
2. Hallar la expresión más sencilla posible para $F_{b,body}\ f\ \sigma$.

Cálculo de semántica de ciclos

1. Hallar la expresión más sencilla posible para $body$.
2. Hallar la expresión más sencilla posible para $F_{b,body}\ f\ \sigma$.
3. Calcular los primeros elementos de la cadena $F^i \perp \sigma$.

Cálculo de semántica de ciclos

1. Hallar la expresión más sencilla posible para $body$.
2. Hallar la expresión más sencilla posible para $F_{b,body}\ f\ \sigma$.
3. Calcular los primeros elementos de la cadena $F^i \perp \sigma$.
4. Hallar una expresión general para el supremo.

Teorema de Coincidencia

Expresiones y comandos

Para expresiones

Si $\sigma\, w = \sigma'\, w$ para toda $w \in FV(e)$, entonces $[\![e]\!]\sigma = [\![e]\!]\sigma'$.

Para comandos

¿Si $\sigma\, w = \sigma'\, w$ para toda $w \in FV(c)$, entonces $[\![c]\!]\sigma = [\![c]\!]\sigma'$?

Para expresiones

Si $\sigma\, w = \sigma'\, w$ para toda $w \in FV(e)$, entonces $[\![e]\!]\sigma = [\![e]\!]\sigma'$.

Para comandos

¿Si $\sigma\, w = \sigma'\, w$ para toda $w \in FV(c)$, entonces $[\![c]\!]\sigma = [\![c]\!]\sigma'$?

Sea $c \doteq \mathsf{x} := 5 + \mathsf{w}$.

Sea $\sigma_0\, u = 0$. Ahora consideremos
$\sigma = [\sigma_0 \mid \mathsf{x} : 3 \mid \mathsf{w} : 4 \mid \mathsf{z} : 3]$ y
$\sigma' = [\sigma_0 \mid \mathsf{x} : 3 \mid \mathsf{w} : 4]$.

Variables libres y Variables asignables

Variables libres

$$FV(\mathbf{skip}) = \emptyset$$

Variables libres y Variables asignables

Variables libres

$$
\begin{aligned}
FV(\mathbf{skip}) &= \emptyset \\
FV(v := e) &= \{v\} \cup FV(e)
\end{aligned}
$$

Variables libres y Variables asignables

Variables libres

$$\begin{array}{lcl} FV(\mathbf{skip}) & = & \emptyset \\ FV(v := e) & = & \{v\} \cup FV(e) \\ FV(c_0; c_1) & = & FV(c_0) \cup FV(c_1) \end{array}$$

Variables libres y Variables asignables

Variables libres

$$
\begin{array}{lcl}
FV(\mathbf{skip}) & = & \emptyset \\
FV(v := e) & = & \{v\} \cup FV(e) \\
FV(c_0; c_1) & = & FV(c_0) \cup FV(c_1) \\
FV(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1) & = & FV(b) \cup FV(c_0) \cup FV(c_1)
\end{array}
$$

Variables libres y Variables asignables

Variables libres

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

Variables libres y Variables asignables

Variables libres

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

Variables libres y Variables asignables

Variables Asignables

$$FA(\textbf{skip}) = \emptyset$$

Variables libres y Variables asignables

Variables Asignables

$$\begin{array}{lcl} FA(\mathbf{skip}) & = & \emptyset \\ FA(v := e) & = & \{v\} \end{array}$$

Variables libres y Variables asignables

Variables Asignables

$$
\begin{array}{lcl}
FA(\mathbf{skip}) & = & \emptyset \\
FA(v := e) & = & \{v\} \\
FA(c_0; c_1) & = & FA(c_0) \cup FA(c_1)
\end{array}
$$

Variables libres y Variables asignables

Variables Asignables

$$
\begin{array}{lcl}
FA(\mathbf{skip}) & = & \emptyset \\
FA(v := e) & = & \{v\} \\
FA(c_0; c_1) & = & FA(c_0) \cup FA(c_1) \\
FA(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1) & = & FA(c_0) \cup FA(c_1)
\end{array}
$$

Variables libres y Variables asignables

Variables Asignables

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

Variables libres y Variables asignables

Variables Asignables

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

Variables Asignables

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

Enunciado correcto

Lema de intangibilidad
Si $[\![c]\!]\sigma = \sigma_1 \neq \perp$, entonces $\sigma_1\, v = \sigma\, v$ para toda $v \notin FA(c)$.

Enunciado correcto

Lema de intangibilidad
Si $[\![c]\!]\sigma = \sigma_1 \neq \perp$, entonces $\sigma_1\, v = \sigma\, v$ para toda $v \notin FA(c)$.

Teorema de coincidencia

Si $\sigma\, w = \sigma'\, w$ para toda $w \in FV(c)$, entonces:

1. o bien $[\![c]\!]\sigma = \perp = [\![c]\!]\sigma'$,
2. o bien $[\![c]\!]\sigma = \sigma_1 \neq \perp$ y $[\![c]\!]\sigma' = \sigma_2 \neq \perp$, y para toda $w \in FV(c)$, $\sigma_1\, w = \sigma_2\, w$.

Esquema de prueba

1. Inducción estructural en c.

Esquema de prueba

1. Inducción estructural en c.
2. Los casos que no son ciclos son fáciles.

Esquema de prueba

1. Inducción estructural en c.
2. Los casos que no son ciclos son fáciles.
3. Para el ciclo **while** b **do** cw enunciamos un lema auxiliar para $F^i_{b,cw} \perp$ que probamos por inducción en i.

Esquema de prueba

1. Inducción estructural en c.
2. Los casos que no son ciclos son fáciles.
3. Para el ciclo **while** b **do** cw enunciamos un lema auxiliar para $F^i_{b,cw} \perp$ que probamos por inducción en i.
4. Como es una inducción en los naturales dentro de la inducción de comandos podemos usar la hipótesis inductiva para cw y también tenemos la hipótesis inductiva para el natural.

Sustituciones

Expresiones y comandos

Para expresiones

Si $\sigma\, w = [\![\delta\, w]\!]\sigma'$ para toda $w \in FV(e)$, entonces $[\![e]\!]\sigma = [\![e/\delta]\!]\sigma'$.

Para comandos

¿ Si $\sigma\, w = [\![\delta\, w]\!]\sigma'$ para toda $w \in FV(c)$, entonces $[\![c]\!]\sigma = [\![c/\delta]\!]\sigma'$?

Para expresiones

Si $\sigma\, w = [\![\delta\, w]\!]\sigma'$ para toda $w \in FV(e)$, entonces $[\![e]\!]\sigma = [\![e/\delta]\!]\sigma'$.

Para comandos

¿ Si $\sigma\, w = [\![\delta\, w]\!]\sigma'$ para toda $w \in FV(c)$, entonces $[\![c]\!]\sigma = [\![c/\delta]\!]\sigma'$?

Sea $c \doteq \mathsf{x} := 5 + \mathsf{w}$.

Consideremos la siguiente sustitución: $\delta\, w = \begin{cases} \mathsf{y} + 1 & \text{si } w = \mathsf{x} \\ w & \text{si } w \neq \mathsf{x} \end{cases}$

Expresiones y comandos

Para expresiones

Si $\sigma\, w = [\![\delta\, w]\!]\sigma'$ para toda $w \in FV(e)$, entonces $[\![e]\!]\sigma = [\![e/\delta]\!]\sigma'$.

Para comandos

¿ Si $\sigma\, w = [\![\delta\, w]\!]\sigma'$ para toda $w \in FV(c)$, entonces $[\![c]\!]\sigma = [\![c/\delta]\!]\sigma'$?

Sea $c \doteq \mathsf{x} := 5 + \mathsf{w}$.

Consideremos la siguiente sustitución: $\delta\, w = \begin{cases} \mathsf{y} + 1 & \text{si } w = \mathsf{x} \\ w & \text{si } w \neq \mathsf{x} \end{cases}$

Entonces $c/\delta = \mathsf{y} + 1 := 5 + \mathsf{w}$!

Sustituciones más razonables

Renombres

Una sustitución $\delta\colon \langle var\rangle \to \langle intexp\rangle$ es un ***renombre*** si para toda $v \in \langle var\rangle$, $\delta\, v = w$ para alguna $w \in \langle var\rangle$.

Es decir, un renombre es una función $\delta\colon \langle var\rangle \to \langle var\rangle$.

Enunciado del teorema

Sean $\delta\colon \langle var\rangle \to \langle var\rangle$ y $\sigma, \sigma' : \Sigma$. Si para todo $w \in FV(c)$, $\sigma'\,(\delta\, w) = \sigma\, w$, entonces

1. o bien $[\![c]\!]\sigma = \perp = [\![c\,/\,\delta]\!]\sigma'$.
2. o bien $[\![c]\!]\sigma = \sigma_1$, $[\![c\,/\,\delta]\!]\sigma' = \sigma_2$ y para toda $w \in FV(c)$, $\sigma_1\, w = \sigma_2\, \Box$.

¿Qué debe ir en el espacio en blanco?

Renombres

Una sustitución $\delta \colon \langle var\rangle \to \langle intexp\rangle$ es un ***renombre*** si para toda $v \in \langle var\rangle$, $\delta\, v = w$ para alguna $w \in \langle var\rangle$.

Es decir, un renombre es una función $\delta \colon \langle var\rangle \to \langle var\rangle$.

Enunciado del teorema

Sean $\delta \colon \langle var\rangle \to \langle var\rangle$ y $\sigma, \sigma' : \Sigma$. Si para todo $w \in FV(c)$, $\sigma'\,(\delta\, w) = \sigma\, w$, entonces

1. o bien $[\![c]\!]\sigma = \perp = [\![c \,/\, \delta]\!]\sigma'$.
2. o bien $[\![c]\!]\sigma = \sigma_1$, $[\![c \,/\, \delta]\!]\sigma' = \sigma_2$ y
   para toda $w \in FV(c)$, $\sigma_1\, w = \sigma_2\,(\delta\, w)$.

Sustitución en Comandos

Definición de sustitución

$$\mathbf{skip}\ /\ \delta \quad = \quad \mathbf{skip}$$

Sustitución en Comandos

Definición de sustitución

$$\begin{array}{lcl} \textbf{skip} \,/\, \delta & = & \textbf{skip} \\ v := e \,/\, \delta & = & \delta\, v := e \,/\, \delta \end{array}$$

Definición de sustitución

$$
\begin{array}{lcl}
\mathbf{skip} \,/\, \delta & = & \mathbf{skip} \\
v := e \,/\, \delta & = & \delta\, v := e \,/\, \delta \\
(c_0; c_1) \,/\, \delta & = & c_0 \,/\, \delta \,;\, c_1 \,/\, \delta
\end{array}
$$

Definición de sustitución

$$
\begin{array}{lcl}
\mathbf{skip} \,/\, \delta & = & \mathbf{skip} \\
v := e \,/\, \delta & = & \delta\, v := e \,/\, \delta \\
(c_0; c_1) \,/\, \delta & = & c_0 \,/\, \delta \,;\, c_1 \,/\, \delta \\
(\mathbf{if}\; b\; \mathbf{then}\; c_0\; \mathbf{else}\; c_1) \,/\, \delta & = & \mathbf{if}\; b \,/\, \delta\; \mathbf{then}\; c_0 \,/\, \delta\; \mathbf{else}\; c_1 \,/\, \delta
\end{array}
$$

Sustitución en Comandos

Definición de sustitución

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

Sustitución en Comandos

Definición de sustitución

$$
\begin{array}{lcl}
\textbf{skip} \,/\, \delta & = & \textbf{skip} \\
v := e \,/\, \delta & = & \delta\, v := e \,/\, \delta \\
(c_0; c_1) \,/\, \delta & = & c_0 \,/\, \delta \,;\, c_1 \,/\, \delta \\
(\textbf{if}\, b\, \textbf{then}\, c_0\, \textbf{else}\, c_1) \,/\, \delta & = & \textbf{if}\, b \,/\, \delta\, \textbf{then}\, c_0 \,/\, \delta\, \textbf{else}\, c_1 \,/\, \delta \\
(\textbf{while}\, b\, \textbf{do}\, c) \,/\, \delta & = & \textbf{while}\, (b \,/\, \delta)\, \textbf{do}\, (c_0 \,/\, \delta) \\
(\textbf{newvar}\, v \;:=\; e \;\, \textbf{in}\, c) \,/\, \delta & = & \textbf{newvar}\, v_{new} \;:=\; e \,/\, \delta \;\, \textbf{in}\, c \,/\, [\delta | v : v_{new}]
\end{array}
$$

$$\text{donde}\; v_{new} \notin \{\delta\, w \mid w \in FV(c) - \{v\}\}$$

Problema de alias

Consideremos el programa

$$c \doteq \mathsf{x} := \mathsf{x} + 1; \mathsf{y} := \mathsf{y} * 2$$

Y el renombre $\delta\,\mathsf{x} = \delta\,\mathsf{y} = \mathsf{z}$. Entonces

$$c \,/\, \delta = \mathsf{z} := \mathsf{z} + 1; \mathsf{z} := \mathsf{z} * 2$$

Problema de alias

Consideremos el programa

$$c \doteq \mathsf{x} := \mathsf{x} + 1; \mathsf{y} := \mathsf{y} * 2$$

Y el renombre $\delta\,\mathsf{x} = \delta\,\mathsf{y} = \mathsf{z}$. Entonces

$$c\,/\,\delta = \mathsf{z} := \mathsf{z} + 1; \mathsf{z} := \mathsf{z} * 2$$

Sea σ' un estado tal que $\sigma'\,z = 2$ y sea $\sigma\,x = \sigma\,y = 2$.

Problema de alias

Consideremos el programa

$$c \doteq \mathsf{x} := \mathsf{x} + 1; \mathsf{y} := \mathsf{y} * 2$$

Y el renombre $\delta\,\mathsf{x} = \delta\,\mathsf{y} = \mathsf{z}$. Entonces

$$c \,/\, \delta = \mathsf{z} := \mathsf{z} + 1; \mathsf{z} := \mathsf{z} * 2$$

Sea σ' un estado tal que $\sigma'\,z = 2$ y sea $\sigma\,x = \sigma\,y = 2$.

Pero ni siquiera vale lo que esperamos que valga.

Teorema de Sustitución

Sean $\delta \colon \langle var \rangle \to \langle var \rangle$ y $\sigma, \sigma' : \Sigma$. Si δ es ***inyectiva*** y para todo $w \in FV(c)$, $\sigma' \, (\delta \, w) = \sigma \, w$,

1. o bien $[\![c]\!]\sigma = \perp = [\![c \, / \, \delta]\!]\sigma'$.
2. o bien $[\![c]\!]\sigma = \sigma_1$, $[\![c \, / \, \delta]\!]\sigma' = \sigma_2$ y
   para toda $w \in FV(c)$, $\sigma_1 \, w = \sigma_2 \, (\delta \, w)$.

Teorema de Sustitución. Demostración.

La misma receta que el teorema de coincidencia.

Agregando cosas al lenguaje

¿Cómo añadir un comando que nos permita ejecutar un cuerpo una cierta cantidad de veces?

Python

```
v,w=10,5
for i in range(v+5,w*4):
    c = c + i
```

Javascript

```
var w,v,i;
[v,w] = [10,5];
for (i = v+5; i < w*4; i++) {
  c = c + i;
}
```

Repetición acotada

1. Construcción sintáctica: nuevo comando real o syntactic-sugar?
2. Qué versión queremos?
3. Ecuación semántica