

# 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

$\langle comm \rangle ::= \mathbf{skip}$   
 $\langle var \rangle := \langle intexp \rangle$   
 $\langle comm \rangle ; \langle comm \rangle$   
 $\mathbf{if} \langle boolexp \rangle \mathbf{then} \langle comm \rangle \mathbf{else} \langle comm \rangle$   
 $\mathbf{newvar} \langle var \rangle := \langle intexp \rangle \mathbf{in} \langle comm \rangle$   
 $\mathbf{while} \langle boolexp \rangle \mathbf{do} \langle comm \rangle$

$\langle \text{intexp} \rangle ::= \langle \text{natconst} \rangle$   
 $\langle \text{var} \rangle$   
 $\langle \text{intexp} \rangle + \langle \text{intexp} \rangle$   
 $\langle \text{intexp} \rangle * \langle \text{intexp} \rangle$   
 $\langle \text{intexp} \rangle - \langle \text{intexp} \rangle$   
 $\langle \text{intexp} \rangle / \langle \text{intexp} \rangle$   
 $\langle \text{intexp} \rangle \% \langle \text{intexp} \rangle$   
 $-\langle \text{intexp} \rangle$

$\langle \text{boolexp} \rangle ::= \langle \text{boolconst} \rangle$   
 $\langle \text{intexp} \rangle = \langle \text{intexp} \rangle$   
 $\langle \text{intexp} \rangle < \langle \text{intexp} \rangle$   
 $\langle \text{intexp} \rangle \leq \langle \text{intexp} \rangle$   
 $\langle \text{intexp} \rangle > \langle \text{intexp} \rangle$   
 $\langle \text{intexp} \rangle \geq \langle \text{intexp} \rangle$   
 $\neg \langle \text{boolexp} \rangle$   
 $\langle \text{boolexp} \rangle \vee \langle \text{boolexp} \rangle$   
 $\langle \text{boolexp} \rangle \wedge \langle \text{boolexp} \rangle$

$\langle \text{natconst} \rangle ::= 0 \mid 1 \mid 2 \mid \dots$   
 $\langle \text{boolconst} \rangle ::= \mathbf{true} \mid \mathbf{false}$

## 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:

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

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



## Significado de los comandos de LIS (sin iteración)

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:

$$\llbracket \_ \rrbracket^{intexp} \in \langle intexp \rangle \rightarrow \Sigma \rightarrow \mathbf{Z}$$

$$\llbracket \_ \rrbracket^{boolexp} \in \langle boolexp \rangle \rightarrow \Sigma \rightarrow \{V, F\}$$

$$\llbracket \_ \rrbracket^{comm} \in \langle comm \rangle \rightarrow \Sigma \rightarrow \Sigma$$

## Primeras ecuaciones semánticas

$$[[\text{skip}]]\sigma = \sigma$$

## Primeras ecuaciones semánticas

$$\llbracket \text{skip} \rrbracket \sigma = \sigma$$

$$\llbracket v := e \rrbracket \sigma = [\sigma | v : \llbracket e \rrbracket \sigma]$$

## Primeras ecuaciones semánticas

$$\llbracket \text{skip} \rrbracket \sigma = \sigma$$

$$\llbracket v := e \rrbracket \sigma = [\sigma | v : \llbracket e \rrbracket \sigma]$$

$$\llbracket c_0; c_1 \rrbracket \sigma = \llbracket c_1 \rrbracket (\llbracket c_0 \rrbracket \sigma)$$

## Primeras ecuaciones semánticas

$$\begin{aligned} \llbracket \text{skip} \rrbracket \sigma &= \sigma \\ \llbracket v := e \rrbracket \sigma &= [\sigma | v : \llbracket e \rrbracket \sigma] \\ \llbracket c_0; c_1 \rrbracket \sigma &= \llbracket c_1 \rrbracket (\llbracket c_0 \rrbracket \sigma) \\ \llbracket \text{if } e \text{ then } c \text{ else } c' \rrbracket \sigma &= \text{if } \llbracket e \rrbracket \sigma \text{ then } \llbracket c \rrbracket \sigma \text{ else } \llbracket c' \rrbracket \sigma \end{aligned}$$

## Primeras ecuaciones semánticas

$$\llbracket \text{skip} \rrbracket \sigma = \sigma$$

$$\llbracket v := e \rrbracket \sigma = [\sigma | v : \llbracket e \rrbracket \sigma]$$

$$\llbracket c_0; c_1 \rrbracket \sigma = \llbracket c_1 \rrbracket (\llbracket c_0 \rrbracket \sigma)$$

$$\llbracket \text{if } e \text{ then } c \text{ else } c' \rrbracket \sigma = \text{if } \llbracket e \rrbracket \sigma \text{ then } \llbracket c \rrbracket \sigma \text{ else } \llbracket c' \rrbracket \sigma$$

### Ejemplo:

$$\llbracket x := x - 1 \rrbracket \sigma = [\sigma | x : \llbracket x - 1 \rrbracket \sigma]$$

## Primeras ecuaciones semánticas

$$\llbracket \text{skip} \rrbracket \sigma = \sigma$$

$$\llbracket v := e \rrbracket \sigma = [\sigma | v : \llbracket e \rrbracket \sigma]$$

$$\llbracket c_0; c_1 \rrbracket \sigma = \llbracket c_1 \rrbracket (\llbracket c_0 \rrbracket \sigma)$$

$$\llbracket \text{if } e \text{ then } c \text{ else } c' \rrbracket \sigma = \text{if } \llbracket e \rrbracket \sigma \text{ then } \llbracket c \rrbracket \sigma \text{ else } \llbracket c' \rrbracket \sigma$$

### Ejemplo:

$$\begin{aligned} \llbracket x := x - 1 \rrbracket \sigma &= [\sigma | x : \llbracket x - 1 \rrbracket \sigma] \\ &= [\sigma | x : \llbracket x \rrbracket \sigma - \llbracket 1 \rrbracket \sigma] \end{aligned}$$



## Primeras ecuaciones semánticas

$$\begin{aligned} \llbracket \text{skip} \rrbracket \sigma &= \sigma \\ \llbracket v := e \rrbracket \sigma &= [\sigma | v : \llbracket e \rrbracket \sigma] \\ \llbracket c_0; c_1 \rrbracket \sigma &= \llbracket c_1 \rrbracket (\llbracket c_0 \rrbracket \sigma) \\ \llbracket \text{if } e \text{ then } c \text{ else } c' \rrbracket \sigma &= \text{if } \llbracket e \rrbracket \sigma \text{ then } \llbracket c \rrbracket \sigma \text{ else } \llbracket c' \rrbracket \sigma \end{aligned}$$

### Ejemplo:

$$\begin{aligned} \llbracket x := x - 1 \rrbracket \sigma &= [\sigma | x : \llbracket x - 1 \rrbracket \sigma] \\ &= [\sigma | x : \llbracket x \rrbracket \sigma - \llbracket 1 \rrbracket \sigma] \\ &= [\sigma | x : \sigma x - 1] \end{aligned}$$

## Semántica de **newvar**

Función “restauración de  $v$  según  $\sigma$ ”:

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

## Semántica de **newvar**

Función “restauración de  $v$  según  $\sigma$ ”:

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

$$\llbracket \text{newvar } v := e \text{ in } c \rrbracket \sigma = f_{v,\sigma}(\llbracket c \rrbracket [\sigma | v : \llbracket e \rrbracket \sigma])$$

## Semántica de **newvar**

Función “restauración de  $v$  según  $\sigma$ ”:

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

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

Notación lambda para la función restauración:

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

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

## ¿Qué función en $\mathbb{Z} \rightarrow \mathbb{Z}_\perp$ 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  $\mathbb{Z} \rightarrow \mathbb{Z}_\perp$  de la ecuación recursiva:

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

La ecuación ER define una familia de funciones.

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

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

El menor punto fijo es:

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

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

## Dificultades para dar significado a la iteración

El comando

**while** *b* **do** *c*

debe satisfacer la propiedad:

$$\llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket \sigma = \text{if } \llbracket b \rrbracket \sigma \text{ then } \llbracket c; \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket \sigma \text{ else } \llbracket \mathbf{skip} \rrbracket \sigma$$

## Dificultades para dar significado a la iteración

El comando

**while**  $b$  **do**  $c$

debe satisfacer la propiedad:

$$\begin{aligned} \llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket \sigma &= \text{if } \llbracket b \rrbracket \sigma \text{ then } \llbracket c; \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket \sigma \text{ else } \llbracket \mathbf{skip} \rrbracket \sigma \\ &= \begin{cases} \llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket (\llbracket c \rrbracket \sigma) & \text{si } \llbracket b \rrbracket \sigma \\ \sigma & \text{si } \neg \llbracket b \rrbracket \sigma \end{cases} \end{aligned}$$



## ¿Define esta propiedad una función en $\Sigma \rightarrow \Sigma_{\perp}$ ?

- el dominio semántico  $\Sigma \rightarrow \Sigma$  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 $\Sigma \rightarrow \Sigma_{\perp}$ ?

- el dominio semántico  $\Sigma \rightarrow \Sigma$  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

$$\llbracket \mathbf{while\ } b \mathbf{ do\ } c \rrbracket (\llbracket c \rrbracket \sigma) \quad (2)$$

tiene el problema de que  $\llbracket c \rrbracket \sigma$  puede ser  $\perp$ , que no pertenece al dominio de  $\llbracket \mathbf{while\ } b \mathbf{ do\ } c \rrbracket$ .

Esto se soluciona acudiendo a nuestras funciones auxiliares de transferencia de control.

Si  $f \in \Sigma \rightarrow \Sigma_{\perp}$ , entonces definimos una nueva función  $f_{\perp} \in \Sigma_{\perp} \rightarrow \Sigma_{\perp}$  de la siguiente manera:

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

Esto se soluciona acudiendo a nuestras funciones auxiliares de transferencia de control.

Si  $f \in \Sigma \rightarrow \Sigma_{\perp}$ , entonces definimos una nueva función  $f_{\perp} \in \Sigma_{\perp} \rightarrow \Sigma_{\perp}$  de la siguiente manera:

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

Luego la expresión (2) puede ser corregida poniendo

$$\llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket_{\perp} (\llbracket c \rrbracket \sigma)$$

## Semántica denotacional de LIS

$$\llbracket \ ] \in \langle comm \rangle \rightarrow \Sigma \rightarrow \Sigma_{\perp}$$

## Semántica denotacional de LIS

$$\llbracket \ ] \in \langle \text{comm} \rangle \rightarrow \Sigma \rightarrow \Sigma_{\perp}$$

$$\llbracket \text{skip} \rrbracket \sigma = \sigma$$

$$\llbracket v := e \rrbracket \sigma = [\sigma | v : \llbracket e \rrbracket \sigma]$$

## Semántica denotacional de LIS

$$\llbracket \cdot \rrbracket \in \langle \text{comm} \rangle \rightarrow \Sigma \rightarrow \Sigma_{\perp}$$

$$\llbracket \text{skip} \rrbracket \sigma = \sigma$$

$$\llbracket v := e \rrbracket \sigma = [\sigma | v : \llbracket e \rrbracket \sigma]$$

$$\llbracket c_0; c_1 \rrbracket \sigma = \llbracket c_1 \rrbracket_{\perp} (\llbracket c_0 \rrbracket \sigma)$$

## Semántica denotacional de LIS

$$\llbracket \ ] \in \langle \text{comm} \rangle \rightarrow \Sigma \rightarrow \Sigma_{\perp}$$

$$\llbracket \text{skip} \rrbracket \sigma = \sigma$$

$$\llbracket v := e \rrbracket \sigma = [\sigma | v : \llbracket e \rrbracket \sigma]$$

$$\llbracket c_0 ; c_1 \rrbracket \sigma = \llbracket c_1 \rrbracket_{\perp} (\llbracket c_0 \rrbracket \sigma)$$

$$\llbracket \text{if } e \text{ then } c \text{ else } c' \rrbracket \sigma = \text{if } \llbracket e \rrbracket \sigma \text{ then } \llbracket c \rrbracket \sigma \text{ else } \llbracket c' \rrbracket \sigma$$



## Semántica denotacional de LIS

$$\llbracket \cdot \rrbracket \in \langle \text{comm} \rangle \rightarrow \Sigma \rightarrow \Sigma_{\perp}$$

$$\llbracket \text{skip} \rrbracket \sigma = \sigma$$

$$\llbracket v := e \rrbracket \sigma = [\sigma | v : \llbracket e \rrbracket \sigma]$$

$$\llbracket c_0; c_1 \rrbracket \sigma = \llbracket c_1 \rrbracket_{\perp} (\llbracket c_0 \rrbracket \sigma)$$

$$\llbracket \text{if } e \text{ then } c \text{ else } c' \rrbracket \sigma = \text{if } \llbracket e \rrbracket \sigma \text{ then } \llbracket c \rrbracket \sigma \text{ else } \llbracket c' \rrbracket \sigma$$

$$\llbracket \text{newvar } v := e \text{ in } c \rrbracket \sigma = (\lambda \sigma' \in \Sigma. [\sigma' | v : \sigma v])_{\perp} (\llbracket c \rrbracket [\sigma | v : \llbracket e \rrbracket \sigma])$$

## Semántica del **while**

Dificultad principal:

$$\llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket \sigma = \begin{cases} \llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket \perp \perp (\llbracket c \rrbracket \sigma) & \text{si } \llbracket b \rrbracket \sigma \\ \sigma & \text{si } \neg \llbracket b \rrbracket \sigma \end{cases} \quad (W)$$

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  $\llbracket \text{while } b \text{ do } c \rrbracket$  es una función  
 $\omega \in \Sigma \rightarrow \Sigma_{\perp}$

## 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  $\llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket$  es una función  $\omega \in \Sigma \rightarrow \Sigma_{\perp}$
- tal función  $\omega$  satisface la siguiente "ecuación funcional":

$$\omega \sigma = \begin{cases} \omega_{\perp}(\llbracket c \rrbracket \sigma) & \text{si } \llbracket b \rrbracket \sigma \\ \sigma & \text{si } \neg \llbracket b \rrbracket \sigma \end{cases} \quad (W)$$

Obtenemos entonces una ecuación similar a (ER)

## Semántica del **while**

Sea

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

## Semántica del **while**

Sea

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

entonces, la ecuación del **while** queda

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

## Semántica del **while**

Sea

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

entonces, la ecuación del **while** queda

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

Aquí:

$$\llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket \in \Sigma \rightarrow \Sigma_{\perp}$$

## Semántica del **while**

Sea

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

entonces, la ecuación del **while** queda

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

Aquí:

$$\llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket \in \Sigma \rightarrow \Sigma_{\perp}$$

$$F \in (\Sigma \rightarrow \Sigma_{\perp}) \rightarrow (\Sigma \rightarrow \Sigma_{\perp})$$



## Semántica del **while**

Sea

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

entonces, la ecuación del **while** queda

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

Aquí:

$$\llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket \in \Sigma \rightarrow \Sigma_{\perp}$$

$$F \in (\Sigma \rightarrow \Sigma_{\perp}) \rightarrow (\Sigma \rightarrow \Sigma_{\perp})$$

Luego la ecuación también puede escribirse

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

## 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

$$\llbracket \mathbf{while} \ b \ \mathbf{do} \ c \rrbracket = \bigsqcup_{i=0}^{\infty} F^i \perp_{\Sigma \rightarrow \Sigma} \perp$$

para

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

Esta definición sí es dirigida por sintaxis.

## ¿Cómo calcular el menor punto fijo en $\Sigma \rightarrow \Sigma_{\perp}$ ?

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

- Si la cadena

$$F^0 \perp_{\Sigma \rightarrow \Sigma_{\perp}} \sigma, F^1 \perp_{\Sigma \rightarrow \Sigma_{\perp}} \sigma, F^2 \perp_{\Sigma \rightarrow \Sigma_{\perp}} \sigma \dots$$

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

## ¿Cómo calcular el menor punto fijo en $\Sigma \rightarrow \Sigma_{\perp}$ ?

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

- Si la cadena

$$F^0 \perp_{\Sigma \rightarrow \Sigma_{\perp}} \sigma, F^1 \perp_{\Sigma \rightarrow \Sigma_{\perp}} \sigma, F^2 \perp_{\Sigma \rightarrow \Sigma_{\perp}} \sigma \dots$$

adopta algún valor distinto 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**  $x := x - 2$ .

$$\llbracket \text{while } x \neq 0 \wedge x \neq 1 \text{ do } x := x - 2 \rrbracket = \bigsqcup_{i=0}^{\infty} F^i \perp_{\Sigma \rightarrow \Sigma} \perp$$

para

$$\begin{aligned} F w \sigma &= \begin{cases} w \perp (\llbracket x := x - 2 \rrbracket \sigma) & \text{si } \llbracket x \neq 0 \wedge x \neq 1 \rrbracket \sigma \\ \sigma & \text{si no} \end{cases} \\ &= \begin{cases} \sigma & \text{si } \sigma x \in \{0, 1\} \\ w [\sigma | x : \sigma x - 2] & \text{si no} \end{cases} \end{aligned}$$

Se puede comprobar que

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

Luego

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

## Variables Libres

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



## Variables asignables

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

## Propiedades del LIS

### Teorema de Coincidencia (TC)

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

¿Es correcto enunciarlo como sigue?

$$(\forall w \in FV(c). \sigma w = \sigma' w) \Rightarrow \llbracket c \rrbracket \sigma = \llbracket c \rrbracket \sigma'$$

## Teorema de Coincidencia (TC)

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

o bien  $\llbracket c \rrbracket \sigma = \perp = \llbracket c \rrbracket \sigma'$ ,

o bien  $\llbracket c \rrbracket \sigma \neq \perp \neq \llbracket c \rrbracket \sigma'$  y

$$\forall w \in FV(c). \llbracket c \rrbracket \sigma w = \llbracket c \rrbracket \sigma' w$$

2 Si  $\llbracket c \rrbracket \sigma \neq \perp$ , entonces  $\forall w \notin FA(c). \llbracket c \rrbracket \sigma w = \sigma w$ .

## Teorema de Renombre (TR)

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

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

$$\llbracket \mathbf{newvar} \ u := e \ \mathbf{in} \ c/v \rightarrow u \rrbracket = \llbracket \mathbf{newvar} \ v := e \ \mathbf{in} \ c \rrbracket$$

# Sustituciones

## Conjunto de las sustituciones:

$$\Delta = \langle var \rangle \rightarrow \langle var \rangle$$

## Operador Sustitución:

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

# Sustituciones

## Conjunto de las sustituciones:

$$\Delta = \langle var \rangle \rightarrow \langle var \rangle$$

## Operador Sustitución:

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

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

$$(v := e)/\delta = (\delta v) := (e/\delta)$$

$$(c_0; c_1)/\delta = (c_0/\delta); (c_1/\delta)$$

$$(\mathbf{if } b \mathbf{ then } c_0 \mathbf{ else } c_1)/\delta = \mathbf{if } b/\delta \mathbf{ then } c_0/\delta \mathbf{ else } c_1/\delta$$

$$(\mathbf{while } b \mathbf{ do } c)/\delta = \mathbf{while } b/\delta \mathbf{ do } c/\delta$$

## Operador sustitución para **newvar**

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

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

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

## ¿Cómo enunciar el Teorema de Sustitución para comandos?

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

O sea, vale

$$(\forall w \in FV(c). \sigma(\delta w) = \sigma'w) \Rightarrow \llbracket c/\delta \rrbracket \sigma = \llbracket c \rrbracket \sigma'?$$



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  $\llbracket c/\delta \rrbracket \sigma(\delta w)$  con  $\llbracket c \rrbracket \sigma'$ .

Por ejemplo, si al programa  $c = (x := x - 1; y := 2 * y)$  se le sustituye  $x$  por  $u$  e  $y$  por  $v$  (sustitución  $\delta$ ), entonces al ejecutar  $c/\delta$  en un estado  $\sigma$  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  $\sigma u$  y  $\sigma v$  resp.

## Problema del alias

Consideremos el programa  $x := x - 1; y := 2 * y$ .

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

El programa resultante será  $z := z - 1; z := 2 * z$

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

$$\llbracket z := z - 1; z := 2 * z \rrbracket_{\sigma z} \neq \llbracket x := x - 1; y := 2 * y \rrbracket'_{\sigma} x$$

$$\llbracket z := z - 1; z := 2 * z \rrbracket_{\sigma z} \neq \llbracket x := x - 1; y := 2 * y \rrbracket'_{\sigma} y$$

El problema surge porque  $\delta$  no es inyectiva.

## Teorema de Sustitución

Si  $\delta$  es inyectiva sobre  $FV(c)$  y

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

o bien  $\llbracket c/\delta \rrbracket \sigma = \perp = \llbracket c \rrbracket \sigma'$ ,

o bien  $\llbracket c/\delta \rrbracket \sigma \neq \perp \neq \llbracket c \rrbracket \sigma'$  y

$$\forall w \in FV(c). \llbracket c/\delta \rrbracket \sigma(\delta w) = \llbracket c \rrbracket \sigma' w.$$

## 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:

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

## Fallas en el Lenguaje Imperativo

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

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

**Dominio de resultados posibles:**

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

## Fallas en el Lenguaje Imperativo

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

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

**Dominio de resultados posibles:**

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

**Función semántica:**

$$\llbracket \_ \rrbracket \in \langle comm \rangle \rightarrow \Sigma \rightarrow \Sigma'_{\perp}$$

## Ecuaciones semánticas

$$\llbracket \text{skip} \rrbracket_{\sigma} = \sigma$$

$$\llbracket \text{fail} \rrbracket_{\sigma} = \langle \text{abort}, \sigma \rangle$$

$$\llbracket v := e \rrbracket_{\sigma} = [\sigma | v : \llbracket e \rrbracket_{\sigma}]$$

$$\llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \rrbracket_{\sigma} = \begin{cases} \llbracket c_0 \rrbracket_{\sigma} & \text{si } \llbracket b \rrbracket_{\sigma} \\ \llbracket c_1 \rrbracket_{\sigma} & \text{si no} \end{cases}$$



## Operadores de transferencia de control: \*

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

$$f_* \in \Sigma'_{\perp} \rightarrow \Sigma'_{\perp}$$

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

En este caso, la presencia de una situación abortiva determina que no se transfiera 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 transferido a  $c_1$ .

## Operadores de transferencia de control: +

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

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

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

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

## Operadores de transferencia de control: †

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

$$f_{\dagger} \in \Sigma'_{\perp} \rightarrow \Sigma'_{\perp}$$

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

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

$$\llbracket c_0; c_1 \rrbracket_\sigma = \llbracket c_1 \rrbracket_* (\llbracket c_0 \rrbracket_\sigma)$$

$$\llbracket \text{catchin } c_0 \text{ with } c_1 \rrbracket_\sigma = \llbracket c_1 \rrbracket_+ (\llbracket c_0 \rrbracket_\sigma)$$

$$\llbracket \text{newvar } v := e \text{ in } c \rrbracket_\sigma = (\lambda \sigma' \in \Sigma. [\sigma' | v : \sigma \ v])_+ (\llbracket c \rrbracket [\sigma | v : \llbracket e \rrbracket_\sigma])$$

$$\llbracket \text{while } b \text{ do } c \rrbracket = \bigsqcup_{i=0}^{\infty} F^i \perp_{\Sigma \rightarrow \Sigma \perp}$$

donde

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

## Semántica de transiciones

Describe cómo se realiza el cómputo.

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

## Semántica de transiciones

Describe cómo se realiza el cómputo.

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

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

## Semántica de transiciones

Describe cómo se realiza el cómputo.

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

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

**Conjunto de configuraciones:**

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

$$\Gamma_t = \Sigma \quad (\text{configuraciones terminales})$$

$$\Gamma_n = \langle comm \rangle \times \Sigma \quad (\text{configuraciones no terminales})$$

## Definición axiomática de $\rightarrow$

$$\overline{\langle \mathbf{skip}, \sigma \rangle \rightarrow \sigma}$$

$$\overline{\langle v := e, \sigma \rangle \rightarrow [\sigma | v : \llbracket e \rrbracket \sigma]}$$



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

$$\frac{\langle c_0, \sigma \rangle \rightarrow \langle c'_0, \sigma' \rangle}{\langle c_0; c_1, \sigma \rangle \rightarrow \langle c'_0; c_1, \sigma' \rangle}$$

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

$$\frac{\langle c_0, \sigma \rangle \rightarrow \langle c'_0, \sigma' \rangle}{\langle c_0; c_1, \sigma \rangle \rightarrow \langle c'_0; c_1, \sigma' \rangle}$$

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

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

$$\frac{(\llbracket e \rrbracket \sigma = V)}{\langle \text{if } e \text{ then } c \text{ else } c', \sigma \rangle \rightarrow \langle c, \sigma \rangle}$$

$$\frac{(\llbracket e \rrbracket \sigma = F)}{\langle \text{if } e \text{ then } c \text{ else } c', \sigma \rangle \rightarrow \langle c', \sigma \rangle}$$

$$\frac{(\llbracket e \rrbracket \sigma = V)}{\langle \text{if } e \text{ then } c \text{ else } c', \sigma \rangle \rightarrow \langle c, \sigma \rangle}$$

$$\frac{(\llbracket e \rrbracket \sigma = F)}{\langle \text{if } e \text{ then } c \text{ else } c', \sigma \rangle \rightarrow \langle c', \sigma \rangle}$$

$$\frac{(\llbracket e \rrbracket \sigma = F)}{\langle \text{while } e \text{ do } c, \sigma \rangle \rightarrow \sigma}$$

$$\frac{(\llbracket e \rrbracket \sigma = T)}{\langle \text{while } e \text{ do } c, \sigma \rangle \rightarrow \langle c : \text{while } e \text{ do } c, \sigma \rangle}$$

$$\frac{\langle c, [\sigma | v : \llbracket e \rrbracket_{\sigma}] \rangle \rightarrow \sigma'}{\langle \text{newvar } v := e \text{ in } c, \sigma \rangle \rightarrow [\sigma' | v : \sigma v]}$$

$$\frac{\langle c, [\sigma | v : \llbracket e \rrbracket_{\sigma}] \rangle \rightarrow \langle c', \sigma' \rangle}{\langle \text{newvar } v := e \text{ in } c, \sigma \rangle \rightarrow \langle \text{newvar } v := \sigma' v \text{ in } c', [\sigma' | v : \sigma v] \rangle}$$

## Determinismo y continuación

**Determinismo:** → 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  $c_0 \rightarrow c_1 \rightarrow c_2 \rightarrow \dots$  maximal, esto es, que no puede prolongarse más de lo que está. Dicha ejecución es infinita o termina en una configuración terminal  $\sigma$ .

## Ejecución

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

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

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



## Corrección de la semántica operacional

### Lema 1

1 Si  $\langle c_0, \sigma \rangle \rightarrow^* \sigma'$ , entonces  $\langle c_0; c_1, \sigma \rangle \rightarrow^* \langle c_1, \sigma' \rangle$

## Corrección de la semántica operacional

### Lema 1

- 1 Si  $\langle c_0, \sigma \rangle \rightarrow^* \sigma'$ , entonces  $\langle c_0; c_1, \sigma \rangle \rightarrow^* \langle c_1, \sigma' \rangle$
- 2 Si  $\langle c, [\sigma | v : \llbracket e \rrbracket \sigma] \rangle \rightarrow^* \sigma'$ , entonces

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

## Corrección de la semántica operacional

### Lema 1

1 Si  $\langle c_0, \sigma \rangle \rightarrow^* \sigma'$ , entonces  $\langle c_0; c_1, \sigma \rangle \rightarrow^* \langle c_1, \sigma' \rangle$

2 Si  $\langle c, [\sigma | v : \llbracket e \rrbracket \sigma] \rangle \rightarrow^* \sigma'$ , entonces

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

3 Si  $\langle c, [\sigma | v : \llbracket e \rrbracket \sigma] \rangle \rightarrow^* \langle c', \sigma' \rangle$ , entonces

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

## Corrección de la semántica operacional

### Lema 2

$$\textcircled{1} \quad \langle c, \sigma \rangle \rightarrow \sigma' \implies \llbracket c \rrbracket \sigma = \sigma'.$$

$$\textcircled{2} \quad \langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle \implies \llbracket c \rrbracket \sigma = \llbracket c' \rrbracket \sigma'.$$

## Corrección de la semántica operacional

### Lema 2

$$\textcircled{1} \quad \langle c, \sigma \rangle \rightarrow \sigma' \implies \llbracket c \rrbracket \sigma = \sigma'.$$

$$\textcircled{2} \quad \langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle \implies \llbracket c \rrbracket \sigma = \llbracket c' \rrbracket \sigma'.$$

$$\textbf{Lema 3} \quad \llbracket c \rrbracket \sigma = \sigma' \implies \langle c, \sigma \rangle \rightarrow^* \sigma'$$

## Corrección de la semántica operacional

### Lema 2

- 1  $\langle c, \sigma \rangle \rightarrow \sigma' \implies \llbracket c \rrbracket \sigma = \sigma'$ .
- 2  $\langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle \implies \llbracket c \rrbracket \sigma = \llbracket c' \rrbracket \sigma'$ .

**Lema 3**  $\llbracket c \rrbracket \sigma = \sigma' \implies \langle c, \sigma \rangle \rightarrow^* \sigma'$

**Teorema** Para todo comando  $c$  se tiene  $\{\{c\}\} = \llbracket c \rrbracket$ .

## Semántica operacional de las fallas

Configuraciones terminales:

$$\Gamma_t = \Sigma \cup \{\mathbf{abort}\} \times \Sigma$$

## Semántica operacional de las fallas

Configuraciones terminales:

$$\Gamma_t = \Sigma \cup \{\mathbf{abort}\} \times \Sigma$$

$$\overline{\langle \mathbf{fail}, \sigma \rangle} \rightarrow \langle \mathbf{abort}, \sigma \rangle$$



## Semántica operacional de las fallas

Configuraciones terminales:

$$\Gamma_t = \Sigma \cup \{\mathbf{abort}\} \times \Sigma$$

$$\overline{\langle \mathbf{fail}, \sigma \rangle} \rightarrow \langle \mathbf{abort}, \sigma \rangle$$

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

## Semántica operacional de las fallas

$$\frac{\langle c_0, \sigma \rangle \rightarrow \sigma'}{\langle \text{catchin } c_0 \text{ with } c_1, \sigma \rangle \rightarrow \sigma'}$$

## Semántica operacional de las fallas

$$\frac{\langle c_0, \sigma \rangle \rightarrow \sigma'}{\langle \mathbf{catchin} \ c_0 \ \mathbf{with} \ c_1, \sigma \rangle \rightarrow \sigma'}$$

$$\frac{\langle c_0, \sigma \rangle \rightarrow \langle \mathbf{abort}, \sigma' \rangle}{\langle \mathbf{catchin} \ c_0 \ \mathbf{with} \ c_1, \sigma \rangle \rightarrow \langle c_1, \sigma' \rangle}$$

## Semántica operacional de las fallas

$$\frac{\langle c_0, \sigma \rangle \rightarrow \sigma'}{\langle \mathbf{catchin} \ c_0 \ \mathbf{with} \ c_1, \sigma \rangle \rightarrow \sigma'}$$

$$\frac{\langle c_0, \sigma \rangle \rightarrow \langle \mathbf{abort}, \sigma' \rangle}{\langle \mathbf{catchin} \ c_0 \ \mathbf{with} \ c_1, \sigma \rangle \rightarrow \langle c_1, \sigma' \rangle}$$

$$\frac{\langle c_0, \sigma \rangle \rightarrow \langle c'_0, \sigma' \rangle}{\langle \mathbf{catchin} \ c_0 \ \mathbf{with} \ c_1, \sigma \rangle \rightarrow \langle \mathbf{catchin} \ c'_0 \ \mathbf{with} \ c_1, \sigma' \rangle}$$

## Semántica operacional de las fallas

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

## Semántica operacional de las fallas

$$\frac{\langle c, [\sigma | v : \llbracket e \rrbracket \sigma] \rangle \rightarrow \langle \mathbf{abort}, \sigma' \rangle}{\langle \mathbf{newvar} \ v := e \ \mathbf{in} \ c, \sigma \rangle \rightarrow \langle \mathbf{abort}, [\sigma' | v : \sigma v] \rangle}$$

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  $\gamma$  tiene una única ejecución que puede ser infinita ( $\gamma$  diverge) o terminar en una configuración terminal que puede ser de la forma  $\sigma$  o  $\langle \mathbf{abort}, \sigma \rangle$ .

## Corrección de la semántica operacional de las fallas

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

Se puede definir:

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

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



## 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 $\Omega$

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

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

¿Cómo se define el orden en  $\omega$ ?

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

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

### **Función semántica:**

$$\llbracket \_ \rrbracket \in \langle comm \rangle \rightarrow \Sigma \rightarrow \Omega$$

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

### **Función semántica:**

$$\llbracket \_ \rrbracket \in \langle comm \rangle \rightarrow \Sigma \rightarrow \Omega$$

### **Ecuación semántica de la nueva construcción:**

$$\llbracket !e \rrbracket_\sigma = \langle \llbracket e \rrbracket_\sigma, \sigma \rangle$$

## Restantes Ecuaciones Semánticas

$$\llbracket \text{skip} \rrbracket_{\sigma} = \langle \sigma \rangle$$

## Restantes Ecuaciones Semánticas

$$\llbracket \text{skip} \rrbracket_{\sigma} = \langle \sigma \rangle$$

$$\llbracket \text{fail} \rrbracket_{\sigma} = \langle \langle \text{abort}, \sigma \rangle \rangle$$



## Restantes Ecuaciones Semánticas

$$\llbracket \text{skip} \rrbracket_{\sigma} = \langle \sigma \rangle$$

$$\llbracket \text{fail} \rrbracket_{\sigma} = \langle \langle \text{abort}, \sigma \rangle \rangle$$

$$\llbracket v := e \rrbracket_{\sigma} = \langle [\sigma | v : \llbracket e \rrbracket_{\sigma}] \rangle$$

## Restantes Ecuaciones Semánticas

$$\llbracket \text{skip} \rrbracket_{\sigma} = \langle \sigma \rangle$$

$$\llbracket \text{fail} \rrbracket_{\sigma} = \langle \langle \text{abort}, \sigma \rangle \rangle$$

$$\llbracket v := e \rrbracket_{\sigma} = \langle [\sigma | v : \llbracket e \rrbracket_{\sigma}] \rangle$$

$$\llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \rrbracket_{\sigma} = \begin{cases} \llbracket c_0 \rrbracket_{\sigma} & \text{si } \llbracket b \rrbracket_{\sigma} \\ \llbracket c_1 \rrbracket_{\sigma} & \text{si no} \end{cases}$$

## Restantes Ecuaciones Semánticas

$$\llbracket \text{skip} \rrbracket_{\sigma} = \langle \sigma \rangle$$

$$\llbracket \text{fail} \rrbracket_{\sigma} = \langle \langle \text{abort}, \sigma \rangle \rangle$$

$$\llbracket v := e \rrbracket_{\sigma} = \langle [\sigma | v : \llbracket e \rrbracket_{\sigma}] \rangle$$

$$\llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \rrbracket_{\sigma} = \begin{cases} \llbracket c_0 \rrbracket_{\sigma} & \text{si } \llbracket b \rrbracket_{\sigma} \\ \llbracket c_1 \rrbracket_{\sigma} & \text{si no} \end{cases}$$

$$\llbracket c_0; c_1 \rrbracket_{\sigma} = \llbracket c_1 \rrbracket_{*}(\llbracket c_0 \rrbracket_{\sigma})$$

## Restantes Ecuaciones Semánticas

$$\llbracket \text{skip} \rrbracket_{\sigma} = \langle \sigma \rangle$$

$$\llbracket \text{fail} \rrbracket_{\sigma} = \langle \langle \text{abort}, \sigma \rangle \rangle$$

$$\llbracket v := e \rrbracket_{\sigma} = \langle [\sigma | v : \llbracket e \rrbracket_{\sigma}] \rangle$$

$$\llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \rrbracket_{\sigma} = \begin{cases} \llbracket c_0 \rrbracket_{\sigma} & \text{si } \llbracket b \rrbracket_{\sigma} \\ \llbracket c_1 \rrbracket_{\sigma} & \text{si no} \end{cases}$$

$$\llbracket c_0; c_1 \rrbracket_{\sigma} = \llbracket c_1 \rrbracket_{*}(\llbracket c_0 \rrbracket_{\sigma})$$

$$\llbracket \text{catchin } c_0 \text{ with } c_1 \rrbracket_{\sigma} = \llbracket c_1 \rrbracket_{+}(\llbracket c_0 \rrbracket_{\sigma})$$

## Restantes Ecuaciones Semánticas

$$\llbracket \text{newvar } v := e \text{ in } c \rrbracket_{\sigma} = (\lambda \sigma' \in \Sigma. [\sigma' | v : \sigma \ v])_{\dagger} (\llbracket c \rrbracket [\sigma | v : \llbracket e \rrbracket_{\sigma}] )$$

## Restantes Ecuaciones Semánticas

$$\llbracket \text{newvar } v := e \text{ in } c \rrbracket_{\sigma} = (\lambda \sigma' \in \Sigma. [\sigma' | v : \sigma \ v])_{\dagger}(\llbracket c \rrbracket[\sigma | v : \llbracket e \rrbracket_{\sigma}] )$$

$$\llbracket \text{while } b \text{ do } c \rrbracket = \bigsqcup_{i=0}^{\infty} F^i \perp_{\Sigma \rightarrow \Omega}$$

donde

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

# Operadores

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

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

# Operadores

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

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

Más compacto:

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



## Operadores

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

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

## Operadores

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

$$f_{\dagger} \in \Omega \rightarrow \Omega$$

se define

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

## Más sobre dominios: Producto de posets

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

## Más sobre dominios: Producto de posets

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

### Cadenas

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

Componente a componente forman cadenas en los respectivos órdenes:

$p_0^1 \leq p_0^2 \leq \dots \leq p_0^k \leq \dots$

$p_1^1 \leq p_1^2 \leq \dots \leq p_1^k \leq \dots$

$\vdots$

## Producto de dominios

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

## Producto de dominios

Si  $P_0, P_1, \dots, P_{n-1}$  son dominios, entonces  $P_0 \times P_1, \dots \times P_{n-1}$  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  $P_0, P_1, \dots, P_{n-1}$  se define la unión disjunta

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

Se definen las inyecciones  $\iota_i \in P_i \rightarrow P_0 + P_1 + \dots + P_{n-1}$   
mediante  $\iota_i p = \langle i, p \rangle$

## Uniones Disjuntas

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

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

Se definen las inyecciones  $\iota_i \in P_i \rightarrow P_0 + P_1 + \dots + P_{n-1}$   
mediante  $\iota_i p = \langle i, p \rangle$

### Uniones Disjuntas de cpos

Dados órdenes parciales  $P_0, P_1, \dots, P_{n-1}$ , entonces  
 $P_0 + P_1 + \dots + P_{n-1}$  es un orden 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  $P_0 + P_1 + \dots + P_{n-1}$  es una que proviene enteramente de uno de los  $P_i$ :

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

donde los  $p_j$  son todos elementos de  $P_i$ .

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

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

donde los  $p_j$  son todos elementos de  $P_i$ .

## Unión de dominios

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

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

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

donde los  $p_j$  son todos elementos de  $P_i$ .

## Unión de dominios

Dados dominios  $P_0, P_1, \dots, P_{n-1}$ , entonces  $P_0 + P_1 + \dots + P_{n-1}$  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

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

## Dominios recursivos

### Domino semántico para LIS con fallas y output

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

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

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

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

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

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

## Notación

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

$$\iota_{term} = \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{norm} \in \Sigma \rightarrow \Omega$$



## Notación

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

$$\iota_{term} = \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{norm} \in \Sigma \rightarrow \Omega$$

$$\iota_{abort} = \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{abnorm} \in \Sigma \rightarrow \Omega$$

## Notación

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

$$\iota_{term} = \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{norm} \in \Sigma \rightarrow \Omega$$

$$\iota_{abort} = \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{abnorm} \in \Sigma \rightarrow \Omega$$

$$\iota_{out} = \psi \cdot \iota_{\perp} \cdot \iota_1 \in \mathbf{Z} \times \Sigma \rightarrow \Omega$$

## Notación

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

$$\iota_{term} = \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{norm} \in \Sigma \rightarrow \Omega$$

$$\iota_{abort} = \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{abnorm} \in \Sigma \rightarrow \Omega$$

$$\iota_{out} = \psi \cdot \iota_{\perp} \cdot \iota_1 \in \mathbf{Z} \times \Sigma \rightarrow \Omega$$

$$\perp_{\Omega} = \psi(\perp) \in \Omega$$

$$\begin{array}{c}
 \Sigma \xrightarrow{\iota_{norm}} \Sigma' \xrightarrow{\iota_0} \Sigma' + \mathbf{Z} \times \Omega \xrightarrow{\iota_{\perp}} (\Sigma' + \mathbf{Z} \times \Omega)_{\perp} \xrightarrow{\psi} \Omega \\
 \Sigma \xrightarrow{\iota_{abnorm}} \mathbf{Z} \times \Omega \xrightarrow{\iota_1} \Sigma' + \mathbf{Z} \times \Omega \xrightarrow{\iota_{\perp}} (\Sigma' + \mathbf{Z} \times \Omega)_{\perp} \xrightarrow{\psi} \Omega
 \end{array}$$

## Ecuaciones reformuladas

$$\llbracket \text{skip} \rrbracket_{\sigma} = \iota_{\text{term}} \sigma$$

$$\llbracket \text{fail} \rrbracket_{\sigma} = \iota_{\text{abort}} \sigma$$

## Ecuaciones reformuladas

$$\llbracket \text{skip} \rrbracket_{\sigma} = \iota_{\text{term}} \sigma$$

$$\llbracket \text{fail} \rrbracket_{\sigma} = \iota_{\text{abort}} \sigma$$

$$\llbracket v := e \rrbracket_{\sigma} = \iota_{\text{term}} [\sigma | v : \llbracket e \rrbracket_{\sigma}]$$

## Ecuaciones reformuladas

$$\llbracket \text{skip} \rrbracket_{\sigma} = \iota_{\text{term}} \sigma$$

$$\llbracket \text{fail} \rrbracket_{\sigma} = \iota_{\text{abort}} \sigma$$

$$\llbracket v := e \rrbracket_{\sigma} = \iota_{\text{term}} [\sigma | v : \llbracket e \rrbracket_{\sigma}]$$

$$\llbracket !e \rrbracket_{\sigma} = \iota_{\text{out}} (\llbracket e \rrbracket_{\sigma}, \iota_{\text{term}} \sigma)$$

## Ecuaciones reformuladas

$$\llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \rrbracket_\sigma = \begin{cases} \llbracket c_0 \rrbracket_\sigma & \text{si } \llbracket b \rrbracket_\sigma \\ \llbracket c_1 \rrbracket_\sigma & \text{si no} \end{cases}$$
$$\llbracket c_0; c_1 \rrbracket_\sigma = \llbracket c_1 \rrbracket_\sigma * (\llbracket c_0 \rrbracket_\sigma)$$



## Ecuaciones reformuladas

$$\begin{aligned} \llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \rrbracket_\sigma &= \begin{cases} \llbracket c_0 \rrbracket_\sigma & \text{si } \llbracket b \rrbracket_\sigma \\ \llbracket c_1 \rrbracket_\sigma & \text{si no} \end{cases} \\ \llbracket c_0; c_1 \rrbracket_\sigma &= \llbracket c_1 \rrbracket_* (\llbracket c_0 \rrbracket_\sigma) \\ \llbracket \text{catchin } c_0 \text{ with } c_1 \rrbracket_\sigma &= \llbracket c_1 \rrbracket_+ (\llbracket c_0 \rrbracket_\sigma) \end{aligned}$$

## Ecuaciones reformuladas

$$\llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \rrbracket_{\sigma} = \begin{cases} \llbracket c_0 \rrbracket_{\sigma} & \text{si } \llbracket b \rrbracket_{\sigma} \\ \llbracket c_1 \rrbracket_{\sigma} & \text{si no} \end{cases}$$

$$\llbracket c_0; c_1 \rrbracket_{\sigma} = \llbracket c_1 \rrbracket_{*}(\llbracket c_0 \rrbracket_{\sigma})$$

$$\llbracket \text{catchin } c_0 \text{ with } c_1 \rrbracket_{\sigma} = \llbracket c_1 \rrbracket_{+}(\llbracket c_0 \rrbracket_{\sigma})$$

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

## Ecuaciones reformuladas

$$\llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \rrbracket_\sigma = \begin{cases} \llbracket c_0 \rrbracket_\sigma & \text{si } \llbracket b \rrbracket_\sigma \\ \llbracket c_1 \rrbracket_\sigma & \text{si no} \end{cases}$$

$$\llbracket c_0; c_1 \rrbracket_\sigma = \llbracket c_1 \rrbracket_* (\llbracket c_0 \rrbracket_\sigma)$$

$$\llbracket \text{catchin } c_0 \text{ with } c_1 \rrbracket_\sigma = \llbracket c_1 \rrbracket_+ (\llbracket c_0 \rrbracket_\sigma)$$

$$\llbracket \text{newvar } v := e \text{ in } c \rrbracket_\sigma = (\lambda \sigma' \in \Sigma. [\sigma' | v : \sigma \ v])_{\dagger} (\llbracket c \rrbracket [\sigma | v : \llbracket e \rrbracket_\sigma])$$

$$\llbracket \text{while } b \text{ do } c \rrbracket = \bigsqcup_{i=0}^{\infty} F^i \perp_{\Sigma \rightarrow \Omega}$$

donde

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

## Operadores redefinidos

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

## Operadores redefinidos

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

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

## 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:

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

# Input

## Sintaxis abstracta:

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

## Domino semántico:

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



# Input

## Sintaxis abstracta:

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

## Domino semántico:

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

## Isomorfismos:

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

## Composiciones útiles

$$\iota_{term} = \psi \cdot \iota_{\perp} \cdot \iota_0 \cdot \iota_{norm} \in \Sigma \rightarrow \Omega$$

## Composiciones útiles

$$l_{term} = \psi \cdot l_{\perp} \cdot l_0 \cdot l_{norm} \in \Sigma \rightarrow \Omega$$

$$l_{abort} = \psi \cdot l_{\perp} \cdot l_0 \cdot l_{abnorm} \in \Sigma \rightarrow \Omega$$

## Composiciones útiles

$$l_{term} = \psi \cdot l_{\perp} \cdot l_0 \cdot l_{norm} \in \Sigma \rightarrow \Omega$$

$$l_{abort} = \psi \cdot l_{\perp} \cdot l_0 \cdot l_{abnorm} \in \Sigma \rightarrow \Omega$$

$$l_{out} = \psi \cdot l_{\perp} \cdot l_1 \in \mathbf{Z} \times \Sigma \rightarrow \Omega$$

## Composiciones útiles

$$l_{term} = \psi \cdot l_{\perp} \cdot l_0 \cdot l_{norm} \in \Sigma \rightarrow \Omega$$

$$l_{abort} = \psi \cdot l_{\perp} \cdot l_0 \cdot l_{abnorm} \in \Sigma \rightarrow \Omega$$

$$l_{out} = \psi \cdot l_{\perp} \cdot l_1 \in \mathbf{Z} \times \Sigma \rightarrow \Omega$$

$$l_{in} = \psi \cdot l_{\perp} \cdot l_2 \in (\mathbf{Z} \rightarrow \Sigma) \rightarrow \Omega$$

## Composiciones útiles

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

$$\begin{array}{rcl}
 \Sigma & \xrightarrow{\iota_{norm}} & \Sigma' \\
 \Sigma & \xrightarrow{\iota_{abnorm}} & \mathbf{Z} \times \Omega \\
 & & \mathbf{Z} \rightarrow \Omega \\
 & & \mathbf{Z} \times \Omega \xrightarrow{\iota_1} \Sigma' + \mathbf{Z} \times \Omega \xrightarrow{\iota_{\perp}} (\Sigma' + \mathbf{Z} \times \Omega)_{\perp} \xrightarrow{\psi} \Omega
 \end{array}$$

## Ecuación semántica para el input:

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



## Ecuación semántica para el input:

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

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

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

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

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