

# Lenguaje Imperativo con Output e Input

---

Miguel Pagano

10 de mayo de 2023

# Repaso

---

# Un Lenguaje Imperativo (no tan) Simple

## Comandos

$\langle comm \rangle ::= \text{skip}$   
|  $\langle var \rangle := \langle intexp \rangle$   
|  $\langle comm \rangle ; \langle comm \rangle$   
| **if**  $\langle boolexp \rangle$  **then**  $\langle comm \rangle$  **else**  $\langle comm \rangle$   
| **newvar**  $\langle var \rangle := \langle intexp \rangle$  **in**  $\langle comm \rangle$   
| **while**  $\langle boolexp \rangle$  **do**  $\langle comm \rangle$   
| **fail**  
| **catchin**  $\langle comm \rangle$  **with**  $\langle comm \rangle$   
| **!**  $\langle intexp \rangle$

# Un Lenguaje Imperativo (no tan) Simple

## Expresiones

$\langle natconst \rangle ::= 0 \mid 1 \mid 2 \mid \dots$

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

$\langle intexp \rangle ::= \langle natconst \rangle$

$\langle boolexp \rangle ::= \langle boolconst \rangle$

|  $\langle var \rangle$

|  $\neg \langle boolexp \rangle$

|  $- \langle intexp \rangle$

|  $\langle intexp \rangle \ominus \langle intexp \rangle$

|  $\langle intexp \rangle \oplus \langle intexp \rangle$

|  $\langle boolexp \rangle \oslash \langle boolexp \rangle$

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

$\ominus \in \{<, \leq, =, \neq, \geq, >\}$

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

# Output

---

## Posibles comportamientos

1. Cantidad finita de output y un estado final (exitoso o no):  
 $\langle 4, 3, 2, 1, [\sigma \mid x : 0] \rangle$  ó  $\langle 1, 2, \langle \text{abort}, \sigma \rangle \rangle$ .
2. Cantidad finita de output:  $\langle 2 \rangle$ .
3. Cantidad infinita de output:  $\langle -1, -2, -3, -4, \dots \rangle$ .

## de qué programas?

1. `!4; !3; !2; !1` (ó `!4; !3; !2; !1; fail`)
2. `!x; (while  $x = 2$  do skip); !x,`
3. `!x; while  $x \neq 0$  do !x;  $x := x - 1$ .`

$$\begin{aligned}\Omega &= \{\langle n_0, \dots, n_k, \sigma' \rangle \mid n_i \in \mathbb{Z}, \sigma' \in \hat{\Sigma}\} \\ &\cup \{\langle n_0, \dots, n_k \rangle \mid n_i \in \mathbb{Z}\} \\ &\cup \{\langle n_0, n_1, n_2, n_3 \dots, n_k, \dots \rangle \mid n_i \in \mathbb{Z}\} \\ &\text{donde } \hat{\Sigma} = \Sigma \cup (\{\mathbf{abort}\} \times \Sigma)\end{aligned}$$

### Concatenación de tuplas

Podemos “concatenar” ciertos elementos de  $\Omega$  y obtener elementos de  $\Omega$ :

$$\langle n_0, \dots, n_k \rangle ++ \langle m_0, \dots, s \rangle = \langle n_0, \dots, n_k, m_0, \dots, s \rangle$$

Ahí  $s$  puede ser un entero o un estado (fallido o no).

## Estructura de dominio

1. El orden está dado por ser prefijo:  $\langle 4, 3, 2, 1 \rangle \leq \langle 4, 3, 2, 1, \sigma \rangle$ .



## Estructura de dominio

1. El orden está dado por ser prefijo:  $\langle 4, 3, 2, 1 \rangle \leq \langle 4, 3, 2, 1, \sigma \rangle$ .
2. El supremo de una cadena no interesante es el último elemento de la cadena.

## Estructura de dominio

1. El orden está dado por ser prefijo:  $\langle 4, 3, 2, 1 \rangle \leq \langle 4, 3, 2, 1, \sigma \rangle$ .
2. El supremo de una cadena no interesante es el último elemento de la cadena.
3. El supremo de una cadena interesante es **la** tupla que coincide en todas las posiciones de cada tupla de la cadena.

## Estructura de dominio

1. El orden está dado por ser prefijo:  $\langle 4, 3, 2, 1 \rangle \leq \langle 4, 3, 2, 1, \sigma \rangle$ .
2. El supremo de una cadena no interesante es el último elemento de la cadena.
3. El supremo de una cadena interesante es **la** tupla que coincide en todas las posiciones de cada tupla de la cadena.

## Estructura de dominio

1. El orden está dado por ser prefijo:  $\langle 4, 3, 2, 1 \rangle \leq \langle 4, 3, 2, 1, \sigma \rangle$ .
2. El supremo de una cadena no interesante es el último elemento de la cadena.
3. El supremo de una cadena interesante es **la** tupla que coincide en todas las posiciones de cada tupla de la cadena.

## Observaciones

1. El menor elemento está dado por  $\langle \rangle$ .

## Estructura de dominio

1. El orden está dado por ser prefijo:  $\langle 4, 3, 2, 1 \rangle \leq \langle 4, 3, 2, 1, \sigma \rangle$ .
2. El supremo de una cadena no interesante es el último elemento de la cadena.
3. El supremo de una cadena interesante es **la** tupla que coincide en todas las posiciones de cada tupla de la cadena.

## Observaciones

1. El menor elemento está dado por  $\langle \rangle$ .
2. En una cadena interesante de tuplas hay tuplas de cada longitud,

## Estructura de dominio

1. El orden está dado por ser prefijo:  $\langle 4, 3, 2, 1 \rangle \leq \langle 4, 3, 2, 1, \sigma \rangle$ .
2. El supremo de una cadena no interesante es el último elemento de la cadena.
3. El supremo de una cadena interesante es **la** tupla que coincide en todas las posiciones de cada tupla de la cadena.

## Observaciones

1. El menor elemento está dado por  $\langle \rangle$ .
2. En una cadena interesante de tuplas hay tuplas de cada longitud,
3. por lo tanto el supremo es efectivamente único.

## Semántica

Nueva semántica de programas:

$$\llbracket \_ \rrbracket : \langle comm \rangle \rightarrow (\Sigma \rightarrow \Omega)$$

## Semántica

Nueva semántica de programas:

$$\begin{aligned}\llbracket \_ \rrbracket &: \langle comm \rangle \rightarrow (\Sigma \rightarrow \Omega) \\ \llbracket !e \rrbracket \sigma &= \langle \llbracket e \rrbracket \sigma, \sigma \rangle\end{aligned}$$



## Comandos sencillos

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

## Comandos sencillos

$$\llbracket \mathbf{skip} \rrbracket \sigma = \langle \sigma \rangle$$

$$\llbracket v := e \rrbracket \sigma = \langle [\sigma \mid v : \llbracket e \rrbracket \sigma] \rangle$$

## Comandos sencillos

$$\llbracket \mathbf{skip} \rrbracket \sigma = \langle \sigma \rangle$$

$$\llbracket v := e \rrbracket \sigma = \langle [\sigma \mid v : \llbracket e \rrbracket \sigma] \rangle$$

$$\llbracket \mathbf{if } b \mathbf{ then } c \mathbf{ else } c' \rrbracket \sigma = \begin{cases} \llbracket c \rrbracket \sigma & \text{si } \llbracket b \rrbracket \sigma \\ \llbracket c' \rrbracket \sigma & \text{si } \neg \llbracket b \rrbracket \sigma \end{cases}$$

## Comandos sencillos

$$\llbracket \mathbf{skip} \rrbracket \sigma = \langle \sigma \rangle$$

$$\llbracket v := e \rrbracket \sigma = \langle [\sigma \mid v : \llbracket e \rrbracket \sigma] \rangle$$

$$\llbracket \mathbf{if } b \mathbf{ then } c \mathbf{ else } c' \rrbracket \sigma = \begin{cases} \llbracket c \rrbracket \sigma & \text{si } \llbracket b \rrbracket \sigma \\ \llbracket c' \rrbracket \sigma & \text{si } \neg \llbracket b \rrbracket \sigma \end{cases}$$

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

## Comandos sencillos

$$\llbracket \mathbf{skip} \rrbracket \sigma = \langle \sigma \rangle$$

$$\llbracket v := e \rrbracket \sigma = \langle [\sigma \mid v : \llbracket e \rrbracket \sigma] \rangle$$

$$\llbracket \mathbf{if } b \mathbf{ then } c \mathbf{ else } c' \rrbracket \sigma = \begin{cases} \llbracket c \rrbracket \sigma & \text{si } \llbracket b \rrbracket \sigma \\ \llbracket c' \rrbracket \sigma & \text{si } \neg \llbracket b \rrbracket \sigma \end{cases}$$

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

$$\llbracket \mathbf{newvar } v := e \mathbf{ in } c \rrbracket \sigma = (rest_{v, \sigma})_{\dagger} (\llbracket c \rrbracket [\sigma \mid v : \llbracket e \rrbracket \sigma])$$

## La ecuación

Sea  $b \in \langle boolexp \rangle$  y  $c \in \langle comm \rangle$ .

$$F_{b,c} : (\Sigma \rightarrow \Sigma') \rightarrow (\Sigma \rightarrow \Sigma')$$
$$F_{b,c}(f)(\sigma) = \begin{cases} \sigma & \text{si } \neg \llbracket b \rrbracket \sigma \\ f_{*}(\llbracket c \rrbracket \sigma) & \text{si } \llbracket b \rrbracket \sigma \end{cases}$$

## La ecuación

Sea  $b \in \langle boolexp \rangle$  y  $c \in \langle comm \rangle$ .

$$F_{b,c} : (\Sigma \rightarrow \Sigma') \rightarrow (\Sigma \rightarrow \Sigma')$$
$$F_{b,c}(f)(\sigma) = \begin{cases} \sigma & \text{si } \neg \llbracket b \rrbracket \sigma \\ f_{*}(\llbracket c \rrbracket \sigma) & \text{si } \llbracket b \rrbracket \sigma \end{cases}$$

$$\llbracket \textbf{while } b \textbf{ do } c \rrbracket \sigma = \left( \bigsqcup_{i \in \mathbb{N}} F^i \perp \right) \sigma$$

No hay que cambiar nada, sólo redefinir los operadores.

# Funciones de propagación de errores

## Propagación de errores

$$(\_)_*: (\Sigma \rightarrow \Sigma') \rightarrow (\Sigma' \rightarrow \Sigma')$$

$$f_* (\perp) = \perp$$

$$f_* (\langle \mathbf{abort}, \sigma \rangle) = \langle \mathbf{abort}, \sigma \rangle$$

$$f_* (\sigma) = f(\sigma)$$

## Como *map* para listas

$$(\_)_{\dagger}: (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma' \rightarrow \Sigma')$$

$$f_{\dagger} (\perp) = \perp$$

$$f_{\dagger} (\langle \mathbf{abort}, \sigma \rangle) = \langle \mathbf{abort}, f(\sigma) \rangle$$

$$f_{\dagger} (\sigma) = f(\sigma)$$



## Propagación de errores

$$(\_)_*: (\Sigma \rightarrow \Omega) \rightarrow (\Omega \rightarrow \Omega)$$

$$f_* (\langle n_0, \dots, n_{k-1} \rangle) = \langle n_0, \dots, n_{k-1} \rangle$$

$$f_* (\langle n_0, \dots, n_{k-1}, \sigma \rangle) = \langle n_0, \dots, n_{k-1} \rangle ++ f(\sigma)$$

$$f_* (\langle n_0, \dots, n_{k-1}, \langle \mathbf{abort}, \sigma \rangle \rangle) = \langle n_0, \dots, n_{k-1}, \langle \mathbf{abort}, \sigma \rangle \rangle$$

$$f_* (\langle n_0, \dots, n_{k-1}, \dots \rangle) = \langle n_0, \dots, n_{k-1}, \dots \rangle$$

## Como *map* para listas

$$(\_)_{\dagger}: (\Sigma \rightarrow \Sigma) \rightarrow (\Omega \rightarrow \Omega)$$

$$f_{\dagger} (\langle n_0, \dots, n_{k-1} \rangle) = \langle n_0, \dots, n_{k-1} \rangle$$

$$f_{\dagger} (\langle n_0, \dots, n_{k-1}, \sigma \rangle) = \langle n_0, \dots, n_{k-1}, f(\sigma) \rangle$$

$$f_{\dagger} (\langle n_0, \dots, n_{k-1}, \langle \mathbf{abort}, \sigma \rangle \rangle) = \langle n_0, \dots, n_{k-1}, \langle \mathbf{abort}, f(\sigma) \rangle \rangle$$

$$f_{\dagger} (\langle n_0, \dots, n_{k-1}, \dots \rangle) = \langle n_0, \dots, n_{k-1}, \dots \rangle$$

# Una presentación más estructural para $\Omega$

Consideremos las siguientes funciones:

$$\iota_{\perp} : \{\langle \rangle\} \rightarrow \Omega$$

$$\iota_{\perp}(\langle \rangle) = \langle \rangle$$

$$\iota_{abort} : \Sigma \rightarrow \Omega$$

$$\iota_{abort}(\sigma) = \langle \sigma \rangle$$

$$\iota_{term} : \Sigma \rightarrow \Omega$$

$$\iota_{term}(\sigma) = \langle \sigma \rangle$$

$$\iota_{out} : \mathbb{Z} \times \Omega \rightarrow \Omega$$

$$\iota_{out}(\langle n, \omega \rangle) = \langle n \rangle ++ \omega$$

## Propagación de errores

$$(\_)_*: (\Sigma \rightarrow \Omega) \rightarrow (\Omega \rightarrow \Omega)$$

$$f_* (\perp) = \perp$$

$$f_* (\iota_{term}(\sigma)) = f(\sigma)$$

$$f_* (\iota_{abort}(\sigma)) = \iota_{abort}(\sigma)$$

$$f_* (\iota_{out}(\langle n, \omega \rangle)) = \iota_{out}(\langle n, f_*(\omega) \rangle)$$

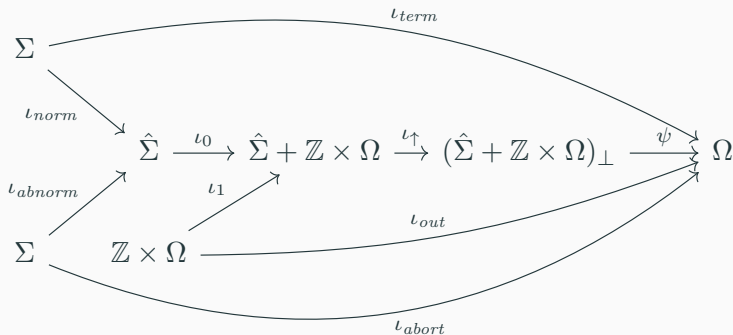
**Observación:** Es fundamental que  $f_*$  sea continua.

## Una nueva definición de Omega

$$\Omega \overset{\phi}{\underset{\psi}{\approx}} (\hat{\Sigma} + \mathbb{Z} \times \Omega)_{\perp}$$

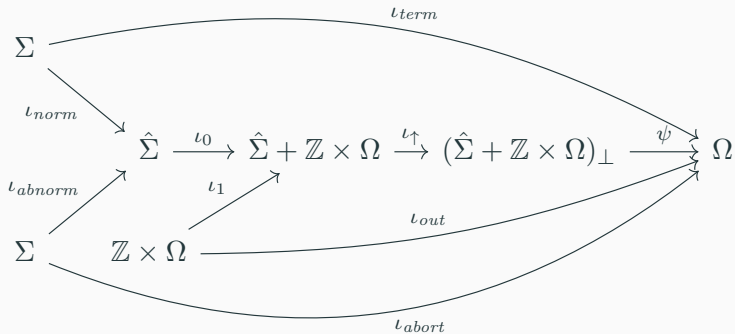
# Una nueva definición de Omega

$$\Omega \overset{\phi}{\underset{\psi}{\approx}} (\hat{\Sigma} + \mathbb{Z} \times \Omega)_{\perp}$$

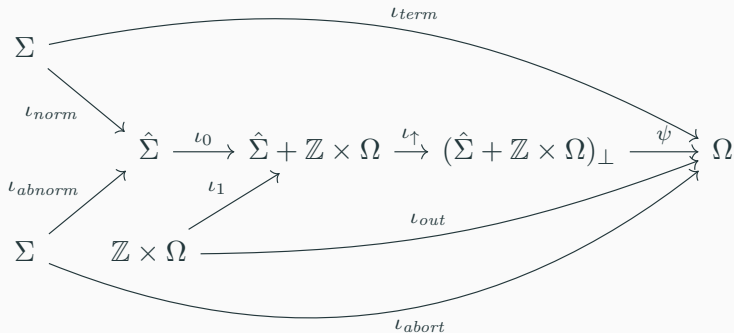


**Observación:** El orden de  $\Omega$  lo podemos pensar como el orden  $(\hat{\Sigma} + \mathbb{Z} \times \Omega)_{\perp}$ .

# Una nueva definición de Omega



# Una nueva definición de Omega



$$l_{term} = \psi \circ l_{\uparrow} \circ l_0 \circ l_{norm}$$

$$l_{abort} = \psi \circ l_{\uparrow} \circ l_0 \circ l_{abnorm}$$

$$l_{out} = \psi \circ l_{\uparrow} \circ l_1$$

**Observación:** Podemos extender  $\Omega$  con nuevas construcciones.



# Propagación de comportamientos y map

$$(\_)_*: (\Sigma \rightarrow \Omega) \rightarrow (\Omega \rightarrow \Omega)$$

$$f_*(\perp) = \perp$$

$$f_*(\iota_{term}(\sigma)) = f(\sigma)$$

$$f_*(\iota_{abort}(\sigma)) = \iota_{abort}(\sigma)$$

$$f_*(\iota_{out}(n, \omega)) = \iota_{out}(n, f_*(\omega))$$

$$(\_)_{\dagger}: (\Sigma \rightarrow \Sigma) \rightarrow (\Omega \rightarrow \Omega)$$

$$f_{\dagger}(\perp) = \perp$$

$$f_{\dagger}(\iota_{term}(\sigma)) = \iota_{term}(f(\sigma))$$

$$f_{\dagger}(\iota_{abort}(\sigma)) = \iota_{abort}(f(\sigma))$$

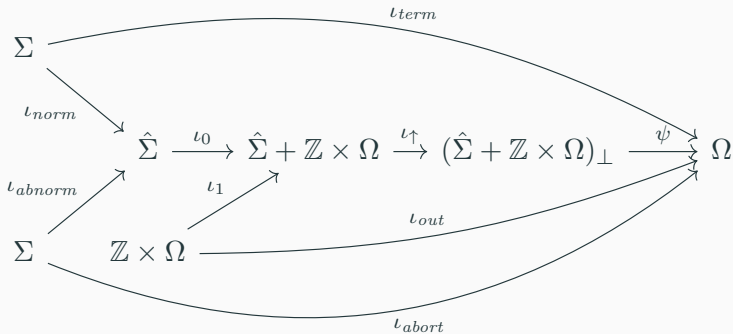
$$f_{\dagger}(\iota_{out}(\langle n, \omega \rangle)) = \iota_{out}(\langle n, f_{\dagger}(\omega) \rangle)$$

## Una nueva definición de Omega

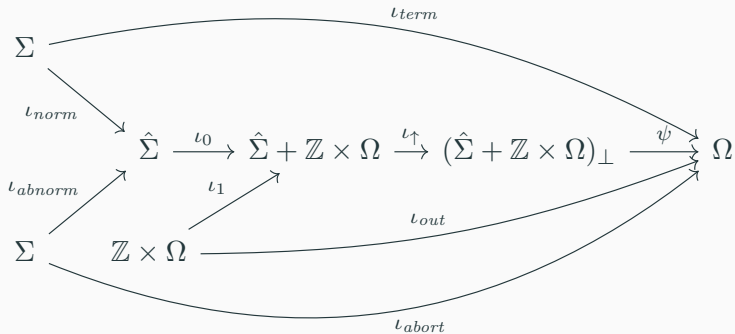
$$\Omega \overset{\phi}{\underset{\psi}{\approx}} (\hat{\Sigma} + \mathbb{Z} \times \Omega)_{\perp}$$

# Una nueva definición de Omega

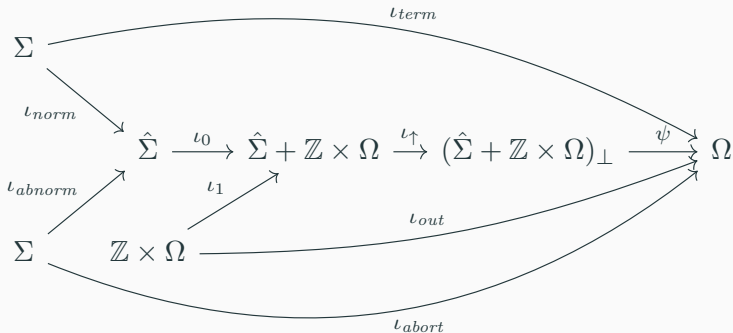
$$\Omega \overset{\phi}{\underset{\psi}{\approx}} (\hat{\Sigma} + \mathbb{Z} \times \Omega)_{\perp}$$



# Una nueva definición de Omega



# Una nueva definición de Omega



$$l_{term} = \psi \circ l_{\uparrow} \circ l_0 \circ l_{norm}$$

$$l_{abort} = \psi \circ l_{\uparrow} \circ l_0 \circ l_{abnorm}$$

$$l_{out} = \psi \circ l_{\uparrow} \circ l_1$$

**Observación:** Podemos extender  $\Omega$  con nuevas construcciones.

## Comandos

$\langle comm \rangle ::=$  **skip**  
|  $\langle var \rangle := \langle intexp \rangle$   
|  $\langle comm \rangle ; \langle comm \rangle$   
| **if**  $\langle boolexp \rangle$  **then**  $\langle comm \rangle$  **else**  $\langle comm \rangle$   
| **newvar**  $\langle var \rangle := \langle intexp \rangle$  **in**  $\langle comm \rangle$   
| **while**  $\langle boolexp \rangle$  **do**  $\langle comm \rangle$   
| **fail**  
| **catchin**  $\langle comm \rangle$  **with**  $\langle comm \rangle$   
| **!**  $\langle intexp \rangle$   
| **?**  $\langle var \rangle$

## Un dominio con comportamientos para Input

$$\Omega \overset{\phi}{\underset{\psi}{\approx}} (\hat{\Sigma} + \mathbb{Z} \times \Omega + \mathbb{Z} \rightarrow \Omega)_{\perp}$$

# Un dominio con comportamientos para Input

$$\Omega \overset{\phi}{\underset{\psi}{\approx}} (\hat{\Sigma} + \mathbb{Z} \times \Omega + \mathbb{Z} \rightarrow \Omega)_{\perp}$$

$$\iota_{term} = \psi \circ \iota_{\uparrow} \circ \iota_0 \circ \iota_{norm}$$

$$\iota_{abort} = \psi \circ \iota_{\uparrow} \circ \iota_0 \circ \iota_{abnorm}$$

$$\iota_{out} = \psi \circ \iota_{\uparrow} \circ \iota_1$$

$$\iota_{in} = \psi \circ \iota_{\uparrow} \circ \iota_2 \quad : (\mathbb{Z} \rightarrow \Omega) \rightarrow \Omega$$

**Observación:** Podemos extender  $\Omega$  con nuevas construcciones.



$$(\_)_*: (\Sigma \rightarrow \Omega) \rightarrow (\Omega \rightarrow \Omega)$$

$$f_* (\perp) = \perp$$

$$f_* (\iota_{term}(\sigma)) = f(\sigma)$$

$$f_* (\iota_{abort}(\sigma)) = \iota_{abort}(\sigma)$$

$$f_* (\iota_{out}(\langle n, \omega \rangle)) = \iota_{out}(\langle n, f_*(\omega) \rangle)$$

$$f_* (\iota_{in}(g)) = \iota_{in}(\lambda n \in \mathbb{Z} . f_*(g(n)))$$

## Propagación de “restauración” (map)

$$(\_)_{\dagger}: (\Sigma \rightarrow \Sigma) \rightarrow (\Omega \rightarrow \Omega)$$

$$f_{\dagger}(\perp) = \perp$$

$$f_{\dagger}(\iota_{term}(\sigma)) = \iota_{term}(f(\sigma))$$

$$f_{\dagger}(\iota_{abort}(\sigma)) = \iota_{abort}(f(\sigma))$$

$$f_{\dagger}(\iota_{out}(n, \omega)) = \iota_{out}(n, f_{\dagger}(\omega))$$

$$f_{\dagger}(\iota_{in}(g)) = \iota_{in}(\lambda n \in \mathbb{Z}. f_{\dagger}(g(n)))$$