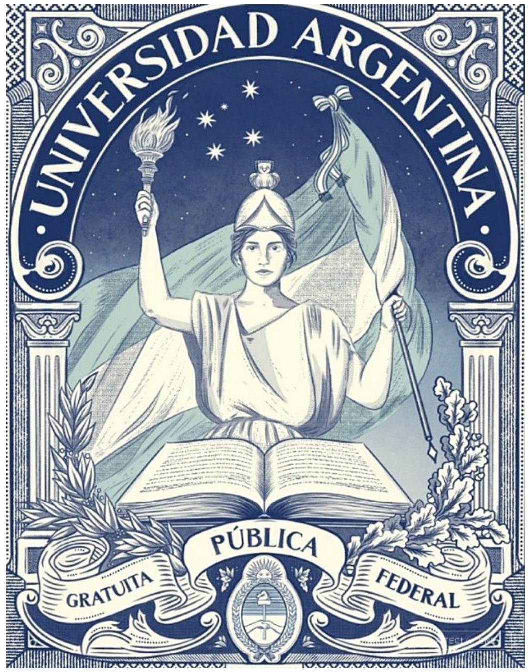
Modelos y simulación - Prácticos

FAMAF - UNC

Severino Di Giovanni



Contents

Figure 1: Severino Di Giovanni, el autor de este apunte. Un anarquista libertario, murió luchando por la libertad. Como él, otros miles han muerto para que nosotros gocemos de los derechos que tenemos. No te dejes engañar por los tristes pregoneros del egoísmo. Amá a tu prójimo y no olvides que si sus derechos se vulneran, los tuyos también. Ayudá a tu compañero de estudio, defendé tu universidad.

1 Teoría

1.1 Órdenes parciales y dominios

1.1.1 Órdenes discretos y llanos

Al orden parcial dado por la relación $a \preceq b \iff a = b$ lo llamamos el orden discreto. Ningún elemento es comparable con otro. Además, dados dos conjuntos X e Y, definimos el siguiente orden sobre $X \to Y$:

$$f \leq g \iff x \in \mathscr{D}_f \Rightarrow x \in \mathscr{D}_g \wedge f(x) \leq_Y f(y)$$

Es fácil demostrar que este orden es parcial.

Además, dado un conjunto X, definimos $X_\perp$ como la operación de *lifting* tal que

$$X_\perp = \left(X \cup \{\perp\}, \preceq \right)$$

con $\preceq$ idéntico al orden discreto, excepto que $\perp$ es menor (y por ende comparable) a todos los elementos de X. Es fácil probar que $X_\perp$ es un orden parcial. Al orden $\preceq$ dado por la operación de lifting se le llama el **orden llano**. Si X es un poset que posee un mínimo, usamos $\perp$ para denotar dicho mínimo.

Si X es un conjunto, definimos también

$$X^\infty = \left(X \cup \{\infty\}, \preceq \right)$$

donde $\preceq$ es el orden usual sobre X, excepto que ∞ es un máximo.

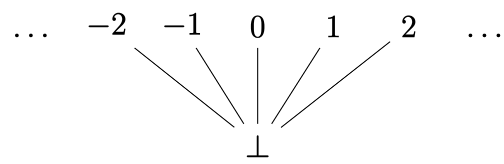


Figure 2: Diagrama de Hasse de $\mathbb{Z}_{\perp}$

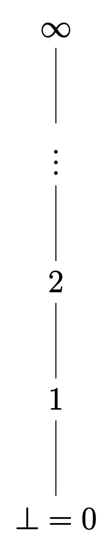


Figure 3: Diagrama de Hasse de $\mathbb{N}$

1.1.2 Cadenas

Una cadena C de un poset $\mathscr{P}$ es una secuencia infinita $\{p_i\}_{i\in\mathbb{N}}$ tal que $p_0 \leq p_1 \leq \ldots$, donde $\leq$ es el orden asociado a $\mathscr{P}$. Si el conjunto $\{p : p = p_i \text{ para algún } i \in \mathbb{N}\}$ es finito, decimos que la cadena es no-interesante. Si es infinito, decimos que la cadena es interesante. Notemos que el caso finito solo puede darse si, a partir de cierto $k \in \mathbb{N}$, $p_k = p_{k+1} = p_{k+2} = \ldots$. Es fácil notar que los órdenes discretos y llanos sólo tienen cadenas no-interesantes.

Si todas las cadenas de un orden parcial tienen supremo, decimos que dicho orden es un **predominio**. En general, si Y es un predominio, $X \to Y$ es un predominio.

> **Prueba.** Asuma que $(Y, \leq_Y)$ es predominio. Entonces, en particular, es un orden parcial. $\therefore$ Dado un conjunto X, tenemos el orden parcial asociado a $X \to Y$ dado por
>
> $$f \leq g \iff \forall x \in X : x \in \mathscr{D}_f \Rightarrow x \in \mathscr{D}_g \wedge f(x) \leq_Y g(x)$$
>
> Sea $f_1 \leq f_2 \leq \ldots$ una cadena arbitraria de $X \to Y$. Si dicha cadena es no-interesante, necesariamente tiene supremo, así que estudiemos el caso en que la cadena es interesante. Dado $x_0 \in \mathscr{D}_{f_1}$ arbitrario, la cadena $f_1 \leq f_2 \leq \ldots$ induce una cadena en Y dada por
>
> $$f_1(x_0) \leq f_2(x_0) \leq \ldots$$
>
> Por hipótesis, dicha cadena tiene un supremo $\bigsqcup_{i \in \mathbb{N}} f_i(x_0)$. Si definimos
>
> $$\mathscr{F}(x_0) = \bigsqcup_{i \in \mathbb{N}} f_i(x_0)$$
>
> tenemos que $\mathscr{F} \in (X \to Y)$, que $x_0 \in \mathscr{D}_{f_i} \Rightarrow x_0 \in \mathscr{D}_{\mathscr{F}}$, y que para todo i se cumple $f_i(x_0) \leq \mathscr{F}(x_0)$. $\therefore \mathscr{F}$ es cota superior de $f_1 \leq f_2 \leq \ldots$ Que es supremo se sigue fácilmente de que $\bigsqcup_{i \in \mathbb{N}} f_i(x_0)$ es supremo de $f_1(x_0) \leq f_2(x_0) \leq \ldots$

Un predominio que tiene un elemento mínimo se denomina **dominio**. Aquí otra vez se cumple que si D es dominio, $X \to D$ es dominio.

> **Prueba.** Asuma que D es dominio. Entonces es predominio y por lo tanto $X \to D$ es predominio. Sea $\psi \in X \to D$ definida como la función constante $\psi = \perp_D$. Entonces es claro que $\psi \leq f$ para toda $f \in X \to D$. Luego ψ es mínimo de $X \to D$.

1.2 Morfismos y funciones continuas

Sean X, Y posets. Si $f \in X \to Y$ preserva el orden parcial, se dice monótona. Si f preserva además el supremo de cadenas, se dice continua. Si además preserva el mínimo, se dice estricta.

Debería ser claro que si $x_1 \leq x_2 \leq \ldots$ es una cadena en X y $f \in X \to Y$ es monótona, entonces $f(x_1) \leq f(x_2) \leq \ldots$ es una cadena en Y. Cabe también destacar que si f es monótona, preserva el supremo de cadenas no-interesantes. Por lo tanto, podríamos definir la noción de continuidad como la preservación de supremos en cadenas interesantes.

> **Ejemplo.** Considere $f \in \mathbb{N}^\infty \to \{\top, \bot\}$. Asuma que $f(n) = \bot$ para todo $n \in \mathbb{N}$ y que $f(\infty) = \top$. Claramente, f es monótona, pero no preserva el supremo de $1 \leq 2 \leq 3 \ldots$ Aplicar f dicha cadena da $\bot \leq \bot \leq \ldots$ con supremo $\bot \neq f(\infty)$.

Aunque la monotonía no implica continuidad, una función monótona en $X \to Y$ nos da una cota superior para el supremo de las cadenas de Y.

Theorem 1 *Si P, Q son predominios y $f \in P \to Q$ es monótona,*

$$\bigsqcup_{i \in \mathbb{N}} f(p_i) \leq f\left(\bigsqcup_{i \in \mathbb{N}} p_i\right)$$

Prueba. Sea $f : P \mapsto Q$ monótona y $p_0 \leq_P p_1 \leq_P \ldots$ una cadena interesante de P.

Para todo j se cumple $p_j \leq \bigsqcup_{i \in \mathbb{N}} p_i$. Como f es monótona, $f(p_j) \leq f\left(\bigsqcup_{i \in \mathbb{N}} p_i\right)$.

$\therefore f\left(\bigsqcup_{i \in \mathbb{N}} p_i\right)$ es cota superior de $\{f(p_i)\}_{i \in \mathbb{N}}$.

Como Q es predominio, la cadena $\{f(p_i)\}_{i \in}$ tiene supremo. Como por def. dicho supremo es la menor cota superior,

$$\bigsqcup_{i \in \mathbb{N}} f(p_i) \leq f\left(\bigsqcup_{i \in \mathbb{N}} p_i\right) \qquad \blacksquare$$

Es fácil demostrar que si f es monótona entonces es continua. De este hecho y del teorema anterior se sigue que f es continua si y solo si

$$f\left(\bigsqcup_{i\in\mathbb{N}} p_i\right) = \bigsqcup_{i\in\mathbb{N}} f(p_i)$$

1.3 Extensión de funciones para lenguajes con fallas

Un estado abortivo, o estado con falla, es un par $\langle \textbf{abort}, \sigma\rangle$, donde **abort** es una nueva palabra del lenguaje y σ es un estado. Definimos

$$\hat{\Sigma} := \Sigma \cup \{\langle \textbf{abort}, \sigma \mid \sigma \in \Sigma\rangle\} = \Sigma \cup \{\textbf{abort}\} \times \Sigma$$

y

$$[\![\textbf{fail}]\!]\ \sigma := \langle \textbf{abort}, \sigma\rangle$$

Ahora damos la siguiente extensión. Si $f \in \Sigma \mapsto \widetilde{\Sigma_\perp}$, definimos $f_* : \widetilde{\Sigma_\perp} \mapsto \widetilde{\Sigma_\perp}$ como

$$f_*(\omega) = \begin{cases} \perp & \omega = \perp \\ f\ \omega & \omega \in \Sigma \\ \langle \textbf{abort}, \sigma\rangle & \omega = \langle \textbf{abort}, \sigma\rangle \end{cases}$$

Entonces se tiene

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

Análogamente,

$$
\begin{aligned}
[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!] &= \bigsqcup_{i \in \mathbb{N}} F^i \perp \\
\text{where } F\ f\ \sigma &= \begin{cases} f_*([\![c]\!]\ \sigma) & [\![b]\!] \\ \sigma & \text{otherwise} \end{cases}
\end{aligned}
$$

Si $f \in \Sigma \mapsto \Sigma$, definimos $f_\dagger \in \widetilde{\Sigma_\perp} \mapsto \widetilde{\Sigma_\perp}$ como:

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

Entonces

$$
\begin{aligned}
&[\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]\ \sigma \\
&\qquad = (\lambda \sigma' \in \Sigma.[\sigma' \mid v : \sigma\ v])_\dagger\ ([\![c]\!][\sigma \mid v : [\![e]\!]\ \sigma)
\end{aligned}
$$

1.4 Output

Una vez incorporado el output a nuestro lenguaje, hay tres maneras en que un programa se puede comportar:

- El programa produce una secuencia finita y luego se ejecuta indefinidamente sin generar más salida.
- El programa produce una secuencia finita y luego termina normalmente o de forma abortiva.
- El programa produce una secuencia infinita.

Definimos Ω como el dominio de salida y determinamos que Ω está ordenado como sigue:

$$\omega \sqsubseteq \psi \iff \omega \text{ es subsecuencia inicial de } \psi$$

Ahora consideremos una cadena $\{\omega_i\}_{i\in\mathbb{N}}$. Si $\{\omega_i\}_{i\in\mathbb{N}}$ no es interesante, su supremo será una secuencia finita de enteros o una secuencia finita con enteros y un último elemento que es un estado. Si $\{\omega_i\}_{i\in\mathbb{N}}$ es interesante, entonces cada ω_i es la "instantánea" de una computación infinita en los tiempos $i = 1, 2, \ldots$. Por ejemplo, en la computación de los dígitos de π, tendríamos:

$$\langle\rangle \sqsubseteq \langle 3\rangle\ \langle 3,1\rangle\ \langle 3,1,4\rangle \sqsubseteq \langle 3,1,4,1\rangle \sqsubseteq \langle 3,1,4,1,5\rangle \sqsubseteq \langle 3,1,4,1,5,9\rangle \sqsubseteq \ldots$$

Entonces el límite de la cadena es el elemento del dominio que describe la salida total de la computación. No es difícil notar que Ω es un dominio.

> **Prueba.** Los casos finitos son triviales. Sea $\{\omega_i\}_{i\in\mathbb{N}}$ una secuencia infinita con $\omega_i \in \Omega$.
>
> Dado que $\omega_i \sqsubseteq \omega_{i+1}$ y estas secuencias son distintas y finitas, ω_{i+1} tiene más elementos que ω_i. Además, si ω_i tiene un elemento j-ésimo, entonces ω_{i+1} también lo tiene, y estos coinciden. Por lo tanto, cualquier cota superior de la cadena es una secuencia infinita.

Además, si μ es una cota superior de la cadena, el i-ésimo elemento de μ debe ser idéntico al i-ésimo elemento de todas las secuencias de la cadena cuya longitud sea al menos i. En otras palabras, para todo $i \in \mathbb{N}$, el i-ésimo elemento de μ está determinado de manera única. $\therefore$ μ es único.

Dado que cualquier cota superior de la cadena está determinada de manera única, la cadena tiene una sola cota superior, y por lo tanto debe ser la menor cota superior.

$\therefore$ Ω es un predominio.

Es trivial observar que la secuencia vacía $\langle\rangle \in \Omega$, que es la salida de un programa que se ejecuta indefinidamente sin realizar ninguna operación de escritura, es el mínimo.

$\therefore$ Ω es un dominio.

1.5 Extensión de outputs

Sea $f : \Sigma \mapsto \Omega$ una función de estados en outputs. La extensión de f definida como $f_* : \Omega \mapsto \Omega$ se define como

$$f_*\ \omega = \begin{cases} \langle n_0, \ldots, n_{k-1} \rangle +\!\!+\ f\ \sigma & \omega = \langle n_0, \ldots, n_{k-1}, \sigma \rangle \\ \omega & c.c. \end{cases}$$

Es decir que f_* es inefectiva (es la función identidad) para todo $\omega \in \Omega$ que no tiene un estado final σ. Esto incluye los outputs infinitos, los outputs finitos sin estado final, y los outputs que terminan en $\langle \mathbf{abort}, \sigma \rangle$.

Esta función nos permite definir la concatenación de comandos:

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

Ejemplo. Si $[\![c_0]\!]\sigma$ termine en $\langle 1, 2, \sigma' \rangle$ y $[\![c_1]\!]\sigma'$ termina en $\langle 3, 4, \gamma \rangle$, $[\![c_0; c_1]\!]\sigma = \langle 1, 2, 3, 4, \gamma \rangle$.

También nos permite definir el **while** de manera que el estado final es inyectado a una secuencia, con $[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma = \bigsqcup_{i \in \mathbb{N}} F^i\ \bot$ y

$$F\ f\ \sigma = \begin{cases} \langle \sigma \rangle & \neg [\![b]\!]\sigma \\ f_*([\![c]\!]\sigma) & c.c. \end{cases}$$

Notemos que al usar f_*, el output de cada iteración del **while** será concatenado al output de la iteración anterior.

1.5.1 Transformación de estados finales

Si $f \in \Sigma \to \Sigma$, $f_\dagger : \Omega \to \Omega$ se define como

$$f_\dagger\ \omega = \begin{cases} \langle n_0, \ldots, n_{k-1}, \langle \mathbf{abort}, f\ \sigma \rangle\rangle & \omega = \langle n_0, \ldots, n_{k-1}, \langle \mathbf{abort}, \sigma \rangle\rangle \\ \langle n_0, \ldots, n_{k-1}, f\ \sigma \rangle & \omega = \langle n_0, \ldots, n_{k-1}, \sigma \rangle \\ \omega & c.c. \end{cases}$$

Es decir, $f_\dagger\ \omega$ aplica f al estado final de ω, incluso si dicho estado final es el estado en que se produjo una falla. Esto nos permite definir

$$[\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]\sigma = \mathscr{R}_\dagger\,([\![c]\!][\sigma \mid v : [\![e]\!]\sigma])$$

donde $\mathscr{R}$ es la restauración:

$$\mathscr{R} := \lambda\sigma' \in \Sigma.[\![\sigma' \mid v : \sigma\ v]\!]$$

1.6 Abstrayendo Ω

Ahora nos proponemos expresar la extensión f_* de f en función de cuatro inyeciones disjuntas. Dichas inyecciones son:

(1) $\iota_\perp \in \{\langle\rangle\} \to \Omega$ definida como $\iota_\perp() = \langle\rangle = \perp_\Omega$

(2) $\iota_{\text{term}} \in \Sigma \to \Omega$ definida como $\iota_{\text{term}}(\sigma) = \langle\sigma\rangle$

(3) $\iota_{\text{abort}} \in \Sigma \to \Omega$ definida como $\iota_{\text{abort}}(\sigma) = \langle\langle \mathbf{abort}, \sigma\rangle\rangle$

(4) $\iota_{\text{out}} \in \mathbb{Z} \times \Omega \to \Omega$ definida como $\iota_{\text{out}}(n, \omega) = \langle n \rangle +\!\!+\ \omega$

$$\begin{aligned} f_*\ \perp &= \perp \\ f_*(\iota_{\text{term}}\ \sigma) &= f\ \sigma \\ f_*(\iota_{\text{abort}}\sigma) &= \iota_{\text{abort}}\ \sigma \\ f_*(\iota_{\text{out}}(n, \omega)) &= \iota_{\text{out}}(n, f_*\omega) \end{aligned}$$

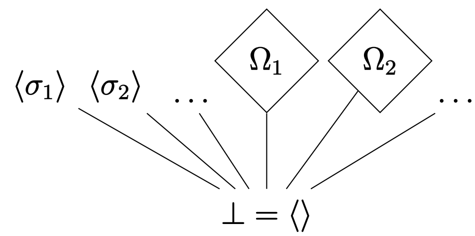
Un primer punto interesante es que ahora la definición de f_* para el caso de un output $\langle n_0, \ldots, n_{k-1}, \ldots \rangle$ es recursiva. También es importante notar que las funciones ι son inyectivas y tienen rangos disjuntos, y que cualquier $\omega \in \Omega$ finito puede formarse a través de sucesivas aplicaciones de estas funciones. Es decir, podemos pensar que estas funciones son constructores de una sintaxis abstracta cuyas frases son las secuencias finitas de Ω.

Análogamente, se puede definir

$$
\begin{aligned}
f_\dagger \, \langle \rangle &= \langle \rangle \\
f_\dagger \, \langle \sigma \rangle &= \langle f \ \sigma \rangle \\
f_\dagger \, \langle \langle \mathbf{abort}, \sigma \rangle \rangle &= \langle \langle \mathbf{abort}, f \ \sigma \rangle \rangle \\
f_\dagger \, (\langle n \rangle +\!\!+ \ \omega) &= \langle n \rangle +\!\!+ \ f_\dagger \ \omega
\end{aligned}
$$

1.6.1 Isomorfismos sobre Ω

El dominio Ω tiene el siguiente orden:



donde $\langle\sigma_i\rangle$ son secuencias con un único estado o un único par $\langle \textbf{abort}, \sigma\rangle$, y el diamante con Ω_k es el conjunto de secuencias que empiezan con el entero k.

Autosimilitud de Ω. Considere lo siguiente: para cada $\omega \in \Omega_k$, existe una única secuencia $\omega' \in \Omega$ tal que $\omega = k +\!\!+\, \omega'$. Es decir, existe una correspondencia uno a uno entre todos elementos de Ω_k y todos los elementos de Ω, y es fácil ver que dicha correspondencia es invertible. $\therefore$ Ω_k es isomórfico a Ω.

> ($\star$) Informalmente, podemos pensar que ciertos elementos de Ω se parecen a Ω, o que Ω tiene subconjuntos que difieren muy poco del mismo Ω.

Naturaleza recursiva de Ω. Considere $\mathbb{Z} \times \Omega$ con $\mathbb{Z}$ bajo el orden discreto. Claramente, para cada $k \in \mathbb{Z}$, existe (k, ω) para cada $\omega \in \Omega$. Es decir, se asocia una "copia" de Ω a cada entero. El orden punto a punto resulta

$$
\begin{tikzcd}
\boxed{\hat{\Sigma}} \arrow[dr, dash] & & \boxed{\mathbb{Z} \times \Omega} \arrow[dl, dash] \\
& \perp &
\end{tikzcd}
$$

$$\langle n, \omega \rangle \sqsubseteq \langle n', \omega' \rangle \iff n = n' \text{ y } \omega \sqsubseteq_{\Omega} \omega'$$

Es decir, cada "copia" tiene el mismo order que en Ω, y los miembros de copias diferentes son incomparables. Por lo tanto, cada $\langle n, \omega \rangle$ se corresponde con un único $\omega' \in \Omega_k$, y tenemos que

$$\bigcup_{k \in \mathbb{Z}} \Omega_k \simeq \mathbb{Z} \times \Omega$$

Por lo tanto, combinando que la unión de los Ω_k es isomórfica a $\mathbb{Z} \times \Omega$ con el orden de Ω dado en el primer diagrama, tenemos

En conclusión,

$$\Omega \simeq \left(\hat{\Sigma} + \mathbb{Z} \times \Omega \right)$$

Por lo tanto, existen funciones continuas φ, ψ

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

tales que $\varphi \circ \psi, \psi \circ \varphi$ son funciones identidad.

Esto revela la naturaleza recursiva de Ω porque nos dice que todo elemento de Ω es o bien una secuencia con un único estado (tal vez abortivo), o bien $\perp$, o bien (caso recursivo) un entero pareado con otro elemento de Ω.

En particular, parear cada elemento de $\hat{\Sigma}$ con 0 y cada elemento de $\mathbb{Z} \times \Omega$ con 1 no cambia ninguno de los resultados anteriores. Por lo tanto, podemos descomponer Ω del siguiente modo:

$$
\begin{tikzcd}
\Sigma \arrow[dr, "\iota_{\mathrm{norm}}"'] & & & & \\
\Sigma \arrow[r, "\iota_{\mathrm{abnorm}}"'] & \hat{\Sigma} \arrow[r, "\iota_0"] & \hat{\Sigma} + \mathbb{Z} \times \Omega \arrow[r, "\iota_\uparrow"] & (\hat{\Sigma} + \mathbb{Z} \times \Omega)_\perp \arrow[r, "\psi"] & \Omega \\
& & \mathbb{Z} \times \Omega \arrow[u, "\iota_1"'] & &
\end{tikzcd}
$$

Acá, $\iota_{\mathrm{norm}}, \iota_{\mathrm{abnorm}}$ inyectan estados en $\hat{\Sigma}$, mapeando $\sigma \mapsto \sigma$ y $\sigma \mapsto \langle \mathbf{abort}, \sigma \rangle$, respectivamente. Las funciones ι_0, ι_1 son la unión disjunta y $\iota_\uparrow$ es simplemente el lifting. Tenemos entonces:

$$
\begin{aligned}
\iota_{\mathrm{term}} &= \psi \circ l_\uparrow \circ \iota_0 \circ \iota_{\mathrm{norm}} \in \Sigma \to \Omega \\
\iota_{\mathrm{abort}} &= \psi \circ l_\uparrow \circ \iota_0 \circ \iota_{\mathrm{abnorm}} \in \Sigma \to \Omega \\
\iota_{\mathrm{out}} &= \psi \circ l_\uparrow \circ \iota_1 \in (\mathbb{Z} \times \Omega) \to \Omega
\end{aligned}
$$

Dar las inyecciones en términos de la relación entre estos conjuntos, y no en términos de la semántica de los mismos, nos permite librarnos del requisito de que el dominio Ω sea una secuencia de enteros y estados. Cualquier grupo de conjuntos donde las ι sean inyecciones con rangos disjuntos satisface las propiedades requeridas.

1.7 Input y nueva extensión de Ω

Si agregamos <comm> ::=?<var> al lenguaje, que hace que una variable tome un valor dado por input, debemos extender la semántica todavía más. Recordemos que, sin input, con $\Omega \simeq (\hat{\Sigma}+\mathbb{Z}\times\Omega)_\perp$, habían cuatro posibilidades para un programa $\mathscr{P}$:

- $\mathscr{P}$ no se detiene y $\omega = \perp$.
- $\mathscr{P}$ termina en σ, y $\omega = \iota_{\text{term}}\ \sigma$.
- $\mathscr{P}$ aborta en σ y $\omega = \iota_{\text{abort}}\ \sigma$.
- $\mathscr{P}$ escribe un $k \in \mathbb{Z}$ y luego se comporta según ω', es decir $\omega = \iota_{\text{out}}(k, \omega')$.

Para describir input, introducimos una posibilidad nueva, donde el input es representado por una función $g \in \mathbb{Z} \to \Omega$:

- $\mathscr{P}$ lee un entero k y su comportamiento es determinado por $g\ k$. En este caso decimos $\omega = \iota_{\text{in}}\ g$ con $g \in \mathbb{Z} \to \Omega$.

Habiendo añadido esta posibilidad, tenemos que tomar Ω como una solución de

$$\Omega \simeq \left(\hat{\Sigma} + (\mathbb{Z} \times \Omega) + (\mathbb{Z} \mapsto \Omega)\right)_\perp$$

Las ocurrencias de Ω que están en $\mathbb{Z} \to \Omega$ son llamadas *resumptions*, porque refieren comportamientos que se dan cuando el proceso se resume después de input o output. Se define

$$\iota_{\text{in}} = \psi \circ l_\uparrow \circ \iota_2 \in (\mathbb{Z} \to \Omega) \to \Omega$$

2 Semántica operacional

Completar.

3 Cálculo lambda

Surge con el trabajo de Church motivados por fundamentar la matemática. Como notación, se usa para generar una expresión que denota una función sin necesidad de nombrarla. Todas las funciones recursivas son definibles en el cálculo lambda. Su sintaxis abstracta es

$$\begin{array}{rl} <\text{expr}> ::= & | <\text{var}> \\ & | <\text{expr}><\text{expr}> \\ & | \lambda<\text{var}>.<\text{expr}> \end{array}$$

La aplicación asocia a izquierda. Por ejemplo,

$$\lambda x.(\lambda y.xyx)x = .(\lambda y.(xy)x)x$$

4 Práctico 1

Considere la gramática

<bin> ::= 0 | 1 | 0<bin> | 1<bin>

(a) Sea $[\![]\!]_s$: <bin> $\to \mathbb{N}$ definida como

$$[\![\alpha_0 \dots \alpha_{n-1}]\!]_s = \sum_{i=1}^{n} \alpha_{i-1} 2^{n-1}$$

¿Es dirigida por sintaxis? ¿Es composicional?

(b) Considere

$$[\![\alpha_0 \alpha_1 \dots . \alpha_{n-1}]\!]_i = \alpha_0 2^{n-1} + [\![\alpha_1 \dots \alpha_{n-1}]\!]_i$$

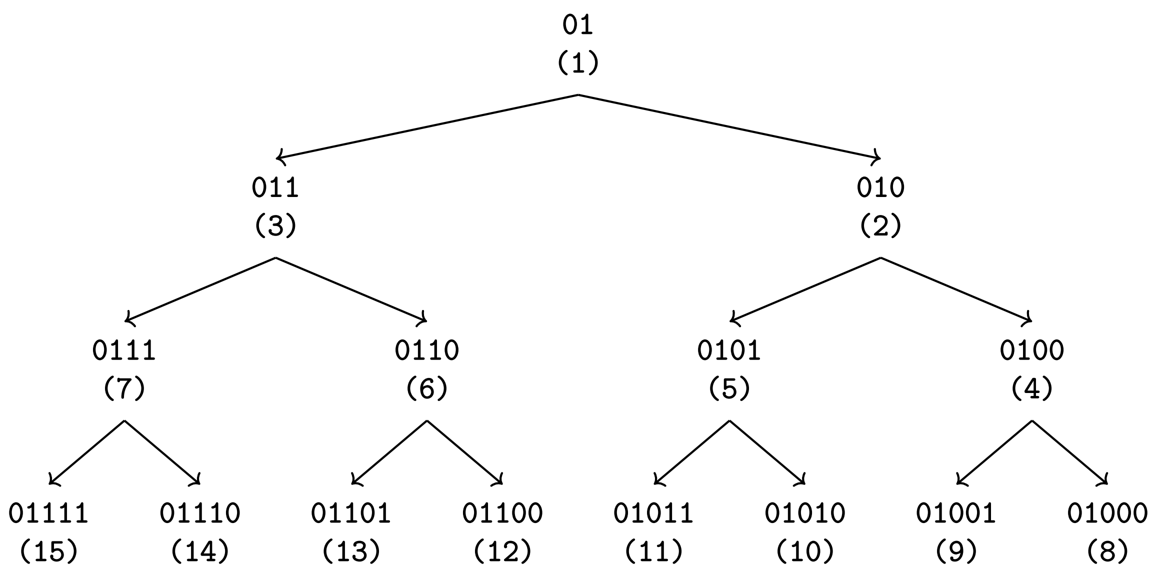
¿Es dirigida por sintaxis?

(c) ¿Puede dar una semántica mediante un conjunto de ecuaciones dirigido por sintaxis?

(a) Una semántica $\mathscr{F}$ sobre un lenguaje $\mathscr{L}$ es composicional si y solo si, para toda $\ell \in \mathscr{L}$, $\mathscr{F}(\ell)$ no depende de ninguna propiedad de ℓ excepto el valor de $\mathscr{F}$ en las sub-frases de ℓ. Debería ser claro que $[\![\alpha_0 \dots \alpha_{n-1}]\!]_s$ no depende en absoluto del significado de las sub-frases $\alpha_0, \dots, a_{n-1}$. Por lo tanto, no es composicional. Como la dirección por sintaxis garantiza composicionalidad, tampoco puede ser dirigido por sintaxis (esto debería además ser obvio, pues no hay una ecuación por cada regla de formación de la gramática).

(b) Dos argumentos distintos para establecer que no es dirigida por sintaxis. (1) Si lo fuera, sería composicional, pero el significado de una frase depende de propiedades externas a la semántica de sus subfrases. Por ejemplo, depende de la cantidad n de subfrases. (2) No hay una ecuación por cada regla de producción.

(c) Es inmediato hacer $[\![0]\!] = 0, [\![1]\!] = 1$. Estudiemos cómo se relaciona el valor de una palabra con el valor de su sub-frase inmediata. Imaginemos que empezamos con 01, que tiene el valor 1. Entonces podemos producir palabras de las siguientes manera:



Debería ser claro que, cada vez que tenemos una palabra binaria b cuya interpretación (informal) es el número k, la interpretación (informal) de $b0$ es $2k$ y la de $b1$ es $2k+1$. Así, por ejemplo, 0101 es interpetado como 5, 01010 como 10, y 01011 como 11. Por lo tanto, planteamos la siguiente semántica dirigida por sintaxis:

$$
\begin{aligned}
[\![0]\!] &= 0 \\
[\![1]\!] &= 1 \\
[\![b0]\!] &= 2 \cdot [\![b]\!] \\
[\![b0]\!] &= 2 \cdot [\![b]\!] + 1
\end{aligned}
$$

Debería ser claro que el significado de una palabra no depende de ninguna propiedad de sus sub-frases excepto la semántica de las mismas. Y existe una ecuación por cada regla de producción. Por lo tanto, la semántica dada es dirigida por sintaxis.

5 Práctico 3: Recursión, predominios y dominios, etc.

(1) Decidir si los siguientes órdenes parciales son predominios o dominios.

(a) <intexp> con el orden discreto

(b) <intexp> $\mapsto \mathbb{B}_\perp$

(c) $\mathbb{B}_\perp \mapsto$ <intexp>.

(a) En el orden discreto, ningún par de elementos es comparable y por lo tanto toda cadena es no interesante. $\therefore$ Toda cadena tiene un supremo. Pero $<$ intexp $>$ bajo dicho orden carece de mínimo. $\therefore$ Es predominio y no es dominio.

(b) $\mathbb{B}_\perp = \{0, 1, \perp\}$ es llano y por lo tanto es predominio, porque toda cadena es no interesante. Tiene mínimo $\perp$ y por ende estambién dominio.

(c) Puesto que <intexp> es predominio, $\mathbb{B}_\perp \mapsto$ <intexp> es predominio.

(4) Calcular el supremo de los siguientes conjuntos.

(a) $\mathscr{A} := \{n \in \mathbb{N} : n \text{ is even}\} \subseteq \mathbb{N}_\perp$

El conjunto ni siquiera tiene cota superior.

(b) $\mathscr{A} := \{n \in \mathbb{N} : n \text{ is even}\} \subseteq \mathbb{N}_\infty$

∞ es la única cota superior de $\mathscr{A}$. $\therefore \infty$ es supremo de $\mathscr{A}$.

(c) $\mathscr{A} := \{n \in \mathbb{N} : n \text{ is prime}\} \subseteq \mathbb{N}^\infty$

Mismo razonamiento que (b).

(d) $\mathscr{A} := \{V, F\} \subseteq \mathbb{B}_\perp$

El conjunto no tiene cota superior porque $\mathbb{B}_\perp$ es el orden llano.

(e) $\mathscr{F} := \{f_n : n \in \mathbb{N}\} \subseteq (\mathbb{N} \mapsto \mathbb{N}_\perp)$ where

$$f_n(x) = \begin{cases} 1 & x \mid n \\ \perp & \text{otherwise} \end{cases}$$

Para todo $k \in \mathbb{N}$ $f_k(n) \leq 1$. Por lo tanto la función que es constantemente 1, C_1, es cota superior de $\mathscr{F}$. Sea $g \in \mathbb{N} \mapsto \mathbb{N}_\perp$ otra cota superior de $\mathscr{F}$. Como 1 es el menor natural, $g \leq C_1 \iff g = \perp$. Pero esto contradiría que g es cota superior.

$\therefore C_1$ es la menor cota superior (el supremo).

$(f*)$ $\mathscr{F} = \{f_n : n \in \mathbb{N}\} \subseteq (\mathbb{N} \mapsto \mathbb{N}_\perp)$ where

$$f_n(x) = \begin{cases} x & |x - 10| < \ln{(n+1)} \\ \perp & \text{otherwise} \end{cases}$$

Dado $x_0 \in \mathbb{N}$, como $\ln(n+1) \to \infty$ cuando $n \to \infty$, siempre podremos encontrar un n_0 tal que

$$f_{n_0}(x_0) = x_0 \neq \perp$$

En otras palabras, para todo x_0, existe algún índice en que la función evaluada en x_0 no es $\perp$. Por ende, es razonable proponer

$$\bigsqcup_{n \in \mathbb{N}} f_n(x) = I_{\mathbb{N} \mapsto \mathbb{N}_\perp}$$

donde I_S es la función identidad del conjunto S.

Es fácil demostrar por casos que $f_i \leq I$. Tomemos $g \in \mathbb{N} \to \mathbb{N}_\perp$ una cota superior de $\mathscr{F}$ y probemos que $I_{\mathbb{N} \to \mathbb{N}_\perp} \leq g$.

Sea $x_0 \in \mathbb{N}$ fijo. Observemos que

$$|x_0 - 10| < \ln(n+1) \iff e^{|x_0 - 10|} < n$$

Tomemos $k_0 := e^{|x_0 - 10|}$ y veamos que

$$|x_0 - 10| < \ln\left(e^{|x_0 - 10|} + 1\right) \iff e^{|x_0 - 10|} < e^{|x_0 - 10|} + 1$$

Entonces, como $|x_0 - 10| < \ln(k_0 + 1) < \ln(\lceil k_0 \rceil + 1)$, y $\lceil k_0 \rceil \in \mathbb{N}$, tenemos garantizado que

$$f_{\lceil k_0 \rceil}(x_0) = x_0$$

Pero entonces, por ser g cota superior de la cadena,

$$f_{\lceil k_0 \rceil} \leq g(x_0) \leq I_{\mathbb{N} \mapsto \mathbb{N}_0}(x_0)$$

Pero entonces tenemos $x_0 \leq g(x_0) \leq x_0$.

$\therefore g = I_{\mathbb{N} \mapsto \mathbb{N}_0}$.

$\therefore \bigsqcup_{i \in \mathbb{N}} \mathscr{F} = I_{\mathbb{N} \mapsto \mathbb{N}_0}$.

(6) Caracterizar todas las funciones continuas en los siguientes conjuntos.

(a) $\mathbb{B}_\perp \mapsto \mathbb{B}_\perp$.

Toda función continua debe ser monótona, así que podemos empezar preguntando qué funciones son monótonas.

Proposición. Si $f(\perp) = \perp$, entonces f es monótona.

Demostración. Dados $a, b \in \mathbb{B}\perp$, $a \leq b$ si y solo si $a = \perp$. Por lo tanto, si $f(\perp) = \perp$, entonces $f(\perp) \leq b$ para todo $b \in \mathbb{B}\perp$. En particular, $f(\perp) \leq f(b)$ para todo $b \in \mathbb{B}_\perp$.

Proposición. Si $f(\perp) \neq \perp$, entonces f es monótona si y solo si f es constante.

Demostración. Supongamos que $f(\perp) \neq \perp$ y que f es monótona. Sea $b \in \{0,1\}$ fijo pero arbitrario. Dado que $\perp \leq b$, se requiere $f(\perp) \leq f(b) \Rightarrow f(\perp) = f(b)$. Ahora sea b^c el complemento de b, es decir, $b^c = 1$ si $b = 0$ y $b^c = 0$ si $b = 1$. El mismo razonamiento que dimos para b demuestra que se requiere $f(\perp) = f(b^c)$. $\therefore f(\perp) = f(b) = f(b^c)$.

Dado que $\{f : f(\perp) = \perp\} \cup \{f : f(\perp) \neq \perp\}$ es una partición de $\mathbb{B}\perp \mapsto \mathbb{B}\perp$, y $\{f : f(\perp) \neq \perp\}$ puede dividirse en funciones constantes y no constantes,

$$\begin{aligned}\mathbb{B} \to \mathbb{B}_\perp = &\{f : f(\perp) = \perp\} \\ &\cup \{C_k : k \neq \perp\} \\ &\cup \{f : f \text{ no constante}, f(\perp) \neq \perp\}\end{aligned}$$

y el conjunto de estos conjuntos es una partición del espacio de funciones que estudiamos. En particular, los dos primeros conjuntos son las funciones monótonas.

Preguntamos: ¿cuáles de estas son continuas? Pero ya hemos afirmado que, dado que $\mathbb{B}_\perp$ es finito, todas sus cadenas son poco interesantes. Y dado que las funciones monótonas preservan cadenas, toda función monótona es continua.

$\therefore$ Las funciones continuas de $\mathbb{B}\bot \mapsto \mathbb{B}\bot$ son todas las funciones que envían $\bot$ a $\bot$ y todas las funciones constantes.

Vayamos aún más lejos y contemos el número de funciones monótonas (continuas). Sabemos que $|A \to B| = |B|^{|A|}$, lo cual significa que $|\mathbb{B}\bot \mapsto \mathbb{B}\bot| = 3^3 = 27$.

Obviamente hay dos funciones en $C_k : k \neq \bot$. En $f : f(\bot) = \bot$ tenemos $3^2 = 9$ funciones. En resumen, hay $9{+}2 = 11$ funciones continuas y $27 - 11 = 16$ funciones no continuas.

(b) $\mathbb{N} \mapsto \mathbb{N}_\perp$

Los argumentos dados en el caso anterior todavía aplican.

Sea $f_0(\perp) := m_0 \neq \perp$. Probaremos que f_0 monotónica si y solo si f_0 constante.

Que constante $\Rightarrow$ monotónica es trivial, así que veamos el otro caso. Asuma que f_0 es monotónica y que existe un $k_0 \in \mathbb{N}$ tal que $f_0(k_0) \neq f_0(\perp)$. Como $\perp \leq k_0$ y f_0 monotónica, tenemos $f_0(\perp) \leq f(k_0)$. Si $f(k_0) = \perp$, entonces tenemos $m_0 \leq \perp$, lo cual es claramente absurdo porque $m_0 \neq \perp$. Si $f(k_0) := m_1 \neq \perp$, entonces tenemos $m_0 \leq m_1$ con ambos siendo números naturales. Pero esto es absurdo, porque en $\mathbb{N}_\perp$ ningún par de naturales es comparable. La contradicción viene de asumir que $f_0(k_0) \neq f_0(\perp)$. Luego $f_0(k) = f(\perp)$ para todo k, y f_0 es constante.

Ahora probaremos que si $f(\perp) = \perp$ entonces f es monotónica. Si $f(\perp) = \perp$, al tomar cualquier par a, b que satisfaga $a \leq b$, tenemos necesariamente $a = \perp$. Por lo tanto $f(a) \leq f(b)$ si y solo si $f(\perp) \leq f(b)$ si y solo si $\perp \leq f(b)$ lo cual es verdadero.

Por lo tanto, vale lo mismo que antes:

$$
\begin{aligned}
\mathbb{B}_\perp \to \mathbb{B}\perp = & \{f : f(\perp) = \perp\} \\
& \cup \{C_k : k \neq \perp\} \\
& \cup \{f : f \text{ not constant}, f(\perp) \neq f(\perp)\}
\end{aligned}
$$

y los primeros dos conjuntos son las funciones monótonas. Como no hay cadenas interesantes, éstas son a su vez las funciones continuas.

(c) $\mathbb{N}^\infty \mapsto \mathbb{N}_\perp$

Sea f continua en $\mathbb{N}^\infty \mapsto \mathbb{N}_\perp$.

Proposition. Si $f(\perp) = \perp$ entonces $f = C_\perp$, donde $C_k = \lambda n.k$ con dominio $\mathbb{N}^\infty$.

Proof. Como f es continua, $a \leq b$ implica $f(a) \leq f(b)$ para todo $a, b \in \mathbb{N}^\infty$. En particular, para todo $n \in \mathbb{N}^\infty$, $n \leq \infty$. Por lo tanto, $f(n) \leq \perp$.

$\therefore$ For all $n \in \mathbb{N}^\infty$, $f(n) = \perp$.

Proposition. Si $f(\perp) \neq \perp$, entonces $f = C_k$ para algún $k \in \mathbb{N}_\perp$.

Proof. Considere la siguiente cadena interesante

$$1 \leq 2 \leq \ldots$$

cuyo supremo es ∞. Como f es continua,

$$f(1) \leq f(2) \leq \ldots$$

es una cadena con supremo $f(\infty)$. Pero claramente $f(n_0), f(n_1)$ ocurren en la cadena. Si asumimos, sin pérdida de generalidad, que $f(n_0)$ aparece antes que $f(n_1)$, tenemos $f(n_0) \leq f(n_1)$. Pero $f(n_0), f(n_1) \in \mathbb{N}_\perp$ y por lo tanto o bien $f(n_0) = \perp$ o bien $f(n_0) = f(n_1)$. Si $f(n_0) = \perp$, como n_0 es un natural arbitrario, esto vale para todo $n \in \mathbb{N}$ y $f(n) = \perp$. Luego $f = C_\perp$. Si $f(n_0) = f(n_1) \neq \perp$, entonces $f = C_{f(n_0)}$.

$\therefore$ f es constante.

(d) $\mathbb{N}^\infty \mapsto \mathbb{N}^\infty$

Si f es continua, entonces necesariamente $f(1) \leq f(2) \leq \ldots$. Pero $f(k) \in \mathbb{N}^\infty$ para todo $k \in \mathbb{N}^\infty$. Por lo tanto se dan uno de dos casos.

Si no existe ningún natural n_0 tal que $f(n_0) = \infty$, entonces el hecho de que

$$f(1) \leq f(2) \leq \ldots$$

sea una cadena solo implica dos cosas: (a) que $f(\infty) = \infty$, (b) que $f(k)$ sea mayor a $f(k-1)$. Por lo tanto, f es definida por todas las funciones que son solución de la siguiente ecuación funcional:

$$F\ f\ n = \begin{cases} \infty & n = \infty \\ f\ (n-1) + k_n & n \neq \infty \end{cases}$$

Si existe un $n_0 \in \mathbb{N}$ tal que $f(n_0) = \infty$, entonces la cadena es de la forma

$$f(1) \leq f(2) \leq \ldots \leq f(n_0) \leq \ldots$$

Por lo tanto, se requiere que $f(n) = \infty$ para todo $n \geq n_0$ y todas las funciones continuas son solución de la ecuación

$$F\ f\ n = \begin{cases} \infty & n = \infty \vee n \geq n_0 \\ f\ (n-1) + k_n & c.c. \end{cases}$$

En síntesis, las funciones continuas son todas las funciones crecientes que mapean $\infty \mapsto \infty$.

(8) Caracterizar los puntos fijos y determinar si existe uno menor para:

(a) $f : \mathbb{N} \mapsto \mathbb{N}$ tal que $f(n) = n$.

Todo valor $n \in \mathbb{N}$ es un punto fijo porque f es identidad. Existe uno menor, naturalmente: el cero.

(b) $f : \mathbb{N}^\infty \mapsto \mathbb{N}^\infty$ tal que $f(n) = n + 1$.

$\infty + n$ no está definido para ningún natural n. Claramente ningún natural es punto fijo.

(c) g : <intexp> $\mapsto$ <intexp> defined as $g(e) = e$.

Esta es la identidad en <intexp> $\mapsto$ <intexp>, por lo cual todo valor es un punto fijo. Sin embargo, <intexp> no es un conjunto ordenado y por ende no tiene sentido hablar de un punto fijo mínimo.

(d) $f : \mathbb{N}^\infty \mapsto \mathbb{N}^\infty$ defined as

$$f(n) = \begin{cases} n + 1 & n < 8 \\ n & \text{otherwise} \end{cases}$$

Si $n \geq 9$ (excepto por ∞), entonces n es punto fijo. Si $n < 8$, no lo es.

(9) Determine si las siguientes funciones en $(\mathbb{N} \mapsto \mathbb{N}_\perp) \mapsto (\mathbb{N} \mapsto \mathbb{N}_\perp)$

son continuas y calcule la i-ésima aplicación de ellas sobre el argumento $\perp_{\mathbb{N}\mapsto\mathbb{N}_\perp}$ para $i = 0, 1, 2$.

(a) F definida como

$$F(f) = \begin{cases} f & \text{es total} \\ \perp_{\mathbb{N}\to\mathbb{N}_\perp} & \text{c. c.} \end{cases}$$

Solución. Sean $\varphi, \psi \in \mathbb{N} \mapsto \mathbb{N}_\perp$ tales que $\varphi \leq \psi$. Es fácil ver que si φ es total entonces ψ es total, de lo cual sale fácilmente por casos que $F(\varphi) \leq F(\psi)$.

Para probar que F no es continua, daremos una cadena interesante cuyo supremo no es preservado por F. Sea

$$\varphi_i(n) = \begin{cases} n & i \leq n \\ \perp & \text{c.c.} \end{cases}$$

y considere la cadena

$$\perp_{\mathbb{N}\to\mathbb{N}_\perp} < \varphi_1 \leq \varphi_2 \leq \varphi_3 \leq \dots$$

.

Proposición. Toda cota superior de $\{\varphi_i\}_{i\in\mathbb{N}}$ es una función total.

Prueba. Para todo $n \in \mathbb{N}$ puede darse un $i \in \mathbb{N}$ tal que $\varphi_i(n)$ está definido. Si g es cota superior, como $\varphi_i \leq g$, tenemos que si $\varphi_i(n)$ está definido también lo está $g(n)$. Es decir, para todo $n \in \mathbb{N}$, $g(n)$ está definido. $\therefore$ g es total.

Proposición. $F\left(\bigsqcup_{i \in \mathbb{N}} \varphi_i\right) = \bigsqcup_{i \in \mathbb{N}} \varphi_i \neq \bot_{\mathbb{N} \to \mathbb{N}_\bot}$.

Prueba. Como toda cota superior es total, en particular el supremo es total, de lo cual la primera identidad se sigue por def. de F. Que el supremo no es bottom se sigue de que bottom es menor estricto a cada φ_i.

Proposición. $\bigsqcup_{i \in \mathbb{N}} F\ \varphi_i = \bot_{\mathbb{N} \to \mathbb{N}_\bot}$.

Prueba. Como cada φ_i es no-total, $F(\varphi_i) = \bot_{\mathbb{N} \to \mathbb{N}_\bot}$. Por lo tanto, la cadena $F(\varphi_1), F(\varphi_2), \ldots$ es simplemente la cadena $\bot_{\mathbb{N} \mapsto \mathbb{N}_\bot} \leq \bot_{\mathbb{N} \mapsto \mathbb{N}_\bot} \leq \ldots$ que tiene supremo $\bot_{\mathbb{N} \mapsto \mathbb{N}_\bot}$.

$$\therefore\ F\left(\bigsqcup_{i \in \mathbb{N}} \varphi_i\right) \neq \bigsqcup_{i \in \mathbb{N}} F(\varphi_i)$$

(c) F definida como

$$F\left(f\left(n\right)\right)=\begin{cases}0 & n=0\\ f(n-2) & \text{c.c}\end{cases}$$

Solución. Es claro que toda f en el dominio de F debe estar definida *al menos* en todos los pares, pues $F\ f\ n$ se define en los valores $0, 2, 4, \ldots$. Más aún, es claro que la imagen de F es una única función: la constante 0 definida *únicamente* en todos los pares.

Sean $\varphi, \psi \in \mathscr{D}(F)$ tales que $\varphi \leq \psi$. Como φ, ψ están definidas en los pares, es claro que $F(\varphi) \leq F(\psi) \iff 0 \leq 0$.

Sea $\varphi_1 \leq \varphi_2 \leq \ldots$ una cadena interesante de funciones en el dominio de F. Es claro que $F(\varphi_i)$ es la constante cero definida en los pares, con lo cual F preserva el supremo y etc.

(10) Calcular la menor $f \in \mathbb{Z} \mapsto \mathbb{Z}_\perp$ que satisface

$$f(n) = \begin{cases} 1 & n = 0 \\ n \cdot f(n-1) & n \neq 0 \end{cases}$$

notando que n corre sobre todo $\mathbb{Z}$.

Solución. Sea $F \in (\mathbb{Z} \mapsto \mathbb{Z}_\perp) \mapsto (\mathbb{Z} \mapsto \mathbb{Z}_\perp)$ definida como

$$F(g) = n \mapsto \begin{cases} 1 & n = 0 \\ n \cdot g(n-1) & n \neq 0 \end{cases}$$

Considere la cadena

$$F^1(\perp_{(\mathbb{Z} \mapsto \mathbb{Z}_\perp)}), F^2\left(\perp_{(\mathbb{Z} \mapsto \mathbb{Z}_\perp)}\right), \ldots$$

algunos de cuyos valores son:

$$\begin{aligned} g_1 := F^1\left(\perp_{(\mathbb{Z} \mapsto \mathbb{Z}_\perp)}\right) &= n \mapsto \begin{cases} 1 & n = 0 \\ n \cdot \perp_{(\mathbb{Z} \mapsto \mathbb{Z}_\perp)}(n-1) & n \neq 0 \end{cases} \\ &= n \mapsto \begin{cases} 1 & n = 0 \\ n \cdot \perp_{\mathbb{Z}_\perp} & n \neq 0 \end{cases} \\ &= n \mapsto \begin{cases} 1 & n = 0 \\ \perp_{\mathbb{Z}_\perp} & n \neq 0 \end{cases} \end{aligned}$$

$$
g_2 := F^2\left(g_1\right) = n \mapsto \begin{cases} 1 & n = 0 \\ n \cdot g_1(n-1) & n \neq 0 \end{cases}
$$

$$
= n \mapsto \begin{cases} 1 & n = 0 \\ n \cdot 1 & n - 1 = 0 \\ n \cdot \perp_{\mathbb{Z}_\perp} & n - 1 \neq 0 \end{cases}
$$

$$
= n \mapsto \begin{cases} 1 & n = 0 \\ n & n = 1 \\ \perp_{\mathbb{Z}_\perp} & n > 1 \end{cases}
$$

$$
g_3 := F^2\left(g_1\right) = n \mapsto \begin{cases} 1 & n = 0 \\ n \cdot g_2(n-1) & n \neq 0 \end{cases}
$$

$$
= n \mapsto \begin{cases} 1 & n = 0 \\ n \cdot 1 & n - 1 = 0 \\ n \cdot (n-1) & n - 1 = 1 \\ \perp_{\mathbb{Z}_\perp} & n - 1 > 1 \end{cases}
$$

$$
= n \mapsto \begin{cases} 1 & n = 0 \\ n & n = 1 \\ n(n-1) & n = 2 \\ \perp_{\mathbb{Z}_\perp} & n > 2 \end{cases}
$$

Proponemos que la forma general de F^k es

$$
F^k((\mathbb{Z} \mapsto \mathbb{Z}_\perp)) = n \mapsto \begin{cases} 1 & n \leq 1 \\ n(n-1)\ldots 2 \cdot 1 & 2 \leq n \leq k \\ \perp_{\mathbb{Z}_\perp} & k < n \end{cases}
$$

Ya hemos dado caso base, así asumamos que la fórmula vale para un k arbitrario y veamos el caso $k+1$. Tenemos que

$$
\begin{aligned}
F^{k+1}\left((\mathbb{Z} \mapsto \mathbb{Z}_{\perp})\right) &= F\left(F^{k}((\mathbb{Z} \mapsto \mathbb{Z}_{\perp}))\right) \\
&= n \mapsto \begin{cases} 1 & n = 0 \\ n \cdot F^{k}((\mathbb{Z} \mapsto \mathbb{Z}_{\perp}))(n-1) & n \neq 0 \end{cases} \\
&= n \mapsto \begin{cases} 1 & n = 0 \\ n \cdot 1 & n-1 \leq 1 \\ n \cdot \Big((n-1)((n-1)-1) \ldots \cdot 2 \cdot 1 & 2 \leq n-1 \leq k \\ n \cdot \perp_{\mathbb{Z}_{\perp}} & k < n-1 \end{cases} \\
&= n \mapsto \begin{cases} 1 & n \leq 1 \\ 2 & n = 2 \\ n(n-1)(n-2) \ldots \cdot 2 \cdot 1 & 3 \leq n \leq k+1 \\ \perp_{\mathbb{Z}_{\perp}} & k+1 < n \end{cases}
\end{aligned}
$$

Los dos primeros casos se contienen, porque si $n = 2$ aplicando la tercer clausual resulta $2 \cdot 1 = 2$. Es decir, tenemos

$$
F^{k+1}((\mathbb{Z} \mapsto \mathbb{Z}_{\perp})) = n \mapsto \begin{cases} 1 & n \leq 1 \\ n(n-1)(n-2) \ldots \cdot 2 \cdot 1 & 2 \leq n \leq k+1 \\ \perp_{\mathbb{Z}_{\perp}} & k+1 < n \end{cases}
$$

que es lo que queríamos probar. Es conclusión,

$$
F^{k}((\mathbb{Z} \mapsto \mathbb{Z}_{\perp})) = n \mapsto \begin{cases} n! & 0 \leq n \leq k \\ \perp_{\mathbb{Z}_{\perp}} & k < n \end{cases}
$$

6 Práctico 4: Lenguaje imperativo simple

(**1**) Demostrar o refutar.

(c) (**if** b **then** c_0 **else** c_1); $c_2 \equiv$ **if** b **then** $c_0; c_2$ **else** $c_1; c_2$

(d) $c2$; (**if** b **then** $c0$ **else** $c1$) $\equiv$ **if** b **then** $c2; c0$ **else** $c_2; c_1$

(c) Sea $p =$ **if** b **then** c_0 **else** c_1 y

$$f = [\![\textbf{if } b \textbf{ then } c_0; c_2 \textbf{ else } c_1; c_2]\!] = \sigma \mapsto \begin{cases} [\![c_0; c_2]\!]\sigma & [\![b]\!]\sigma \\ [\![c_1; c_2]\!]\sigma & c.c. \end{cases}$$

Deseamos probar que $[\![p; c_2]\!] = f$. Por def.

$$\begin{aligned} [\![p; c_2]\!]\sigma &= [\![c_2]\!] \left([\![p]\!]\sigma\right) \\ &= \begin{cases} [\![c_2]\!] \left([\![c_0]\!]\sigma\right) & [\![b]\!]\ \sigma \\ [\![c_2]\!] \left([\![c_1]\!]\sigma\right) & c.c. \end{cases} \\ &= \begin{cases} [\![c_2; c_0]\!]\sigma & [\![b]\!]\sigma \\ [\![c_2; c_1]\!]\sigma & c.c. \end{cases} \end{aligned}$$

$\therefore [\![p; c_2]\!] = f$.

$$
\begin{aligned}
(d)\ [\![c_2; \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!] &= [\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]\,([\![c_2]\!]\sigma) \\
&= \begin{cases} [\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]\,([\![c_2]\!]\sigma) & [\![c_2]\!]\sigma \neq \perp \\ \perp & c.c. \end{cases} \\
&= \begin{cases} [\![c_0]\!]\,([\![c_2]\!]\sigma) & [\![c_2]\!]\sigma \neq \perp \wedge [\![b]\!]\sigma \\ [\![c_1]\!]\,([\![c_2]\!]\sigma) & [\![c_2]\!]\sigma \neq \perp \wedge \neg[\![b]\!]\sigma \\ \perp & [\![c_2]\!]\sigma = \perp \end{cases} \\
&= \begin{cases} [\![c_2; c_0]\!]\sigma & [\![c_2]\!]\sigma \neq \perp \wedge [\![b]\!]\sigma \\ [\![c_2; c_1]\!]\sigma & [\![c_2]\!]\sigma \neq \perp \wedge \neg[\![b]\!]\sigma \\ \perp & [\![c_2]\!]\sigma = \perp \end{cases} \\
&= \begin{cases} [\![c_2; c_0]\!]\sigma & [\![b]\!]\sigma \\ [\![c_2; c_1]\!]\sigma & \neg[\![b]\!]\sigma \end{cases} \\
&= [\![\textbf{if } b \textbf{ then } c_2; c_0 \textbf{ else } c_2; c_1]\!]\sigma
\end{aligned}
$$

(5) (a) Dar la semántica de **while** $x < 2$ **do if** $x < 0$ **then** $x := 0$ **else** $x := x + 1$.

Razonamiento previo. Si $\sigma\ x < 2$, el **while** incrementa x hasta alcanzar el valor 2, por lo que su semántica converge a:

$$\sigma \mapsto \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & \sigma\ x < 2 \end{cases}$$

En el peor caso ($\sigma\ x < 0$), el bucle realiza a lo sumo 3 iteraciones: una para corregir $x < 0$, y dos más para alcanzar 2.

De esto se sigue que: (1) el bucle siempre termina en a lo sumo 3 pasos, y (2) sólo $F^1\ \perp$ a $F^4\ \perp$ aportan información; luego, la cadena se vuelve no interesante.

Por simplicidad, hagamos $p :=$ **if** $x < 0$ **then** $x := 0$ **else** $x := x + 1$ y observemos que

$$[\![p]\!]\sigma = \begin{cases} [\sigma \mid x : 0] & \sigma\ x < 0 \\ [\sigma \mid x : \sigma\ x + 1] & \sigma\ x \geq 0 \end{cases} \tag{1}$$

Definamos $F : (\Sigma \mapsto \Sigma_\perp) \mapsto (\Sigma \mapsto \Sigma_\perp)$ como

$$F\ f\ \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ f\ [\![p]\!]\ \sigma & \sigma\ x < 2 \end{cases}$$

Aplicando (1), obtenemos

$$F\ f\ \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ f\left([\sigma \mid x : \sigma\ x + 1]\right) & \sigma\ x \in \{0, 1\} \\ f\left([\sigma \mid x : 0]\right) & \sigma\ x < 0 \end{cases}$$

Es trivial observar que

$$F \perp \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ \perp & \sigma\ x < 2 \end{cases}$$

Ahora bien,

$$F^2 \perp \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ (F \perp)\,([\sigma \mid x : \sigma\ x + 1]) & \sigma\ x \in \{0, 1\} \\ (F \perp)\,([\sigma \mid x : 0]) & \sigma\ x < 0 \end{cases}$$

En el caso $\sigma\ x \in \{0, 1\}$, tenemos

$$(F \perp)\,([\sigma \mid x : \sigma\ x + 1]) = \begin{cases} F([\sigma \mid x : 2]) & \sigma\ x = 1 \\ F([\sigma \mid x : 1]) & \sigma\ x = 0 \end{cases} = \begin{cases} [\sigma \mid x : 2] & \sigma\ x = 1 \\ \perp & \sigma\ x = 0 \end{cases}$$

En el caso $\sigma\ x < 0$, claramente $F([\sigma \mid x < 0]) = \perp$. Con lo cual

$$F^2 \perp \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & \sigma\ x = 1 \\ \perp & \sigma\ x < 1 \end{cases}$$

De manera análoga se demuestra que

$$F^3 \perp \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & \sigma\ x \in \{0, 1\} \\ \perp & \sigma\ x < 1 \end{cases}$$

Entonces

$$F^4 \perp \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ (F^3 \perp)\,([\sigma \mid x : \sigma\ x + 1]) & \sigma\ x \in \{0, 1\} \\ (F^3 \perp)\,([\sigma \mid x : 0]) & \sigma\ x < 0 \end{cases}$$

$$= \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & \sigma\ x < 2 \end{cases}$$

Es obvio entonces que a partir de $k \geq 4$, $F^{k+1}\perp = F^{k}\perp$, con lo cual $F^1 \perp, F_2 \perp, \ldots$ es una cadena no interesante con supremo $F^4 \perp$.

$$\therefore \bigsqcup_{i \in \mathbb{N}} F^i \perp = \lambda\sigma. \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & \sigma\ x < 2 \end{cases} = [\![\textbf{if } \sigma\ x \geq 2 \textbf{ then skip else } \sigma\ x := 2]\!]$$

(5) (b) Dar la semántica de

$$\textbf{while } x < 2 \textbf{ do if } y = 0 \textbf{ then } x := x + 1 \textbf{ else skip}$$

Debería ser claro que si $y \neq 0$ el ciclo no termina, pues se ejecuta **skip** indefinidamente.

Sea p el comando **if** ejecutado dentro del **while**. Si definimos $F : (\Sigma \mapsto \Sigma_\perp) \mapsto (\Sigma \mapsto \Sigma_\perp)$ como

$$F\ f\ \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ f\,(\ [\![p]\!]\sigma) & \sigma\ x < 2 \end{cases}$$

entonces, desarrollando la semántica de p, tenemos

$$F\ f\ \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ f\,([\sigma \mid x : \sigma\ x + 1]) & \sigma\ x < 2 \wedge \sigma\ y = 0 \\ f\sigma & \sigma\ x < 2 \wedge \sigma\ y \neq 0 \end{cases}$$

Ahora daremos el menor punto fijo de F, que será la semántica del comando. Claramente,

$$F\ \perp\ \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ \perp & \text{c. c.} \end{cases}$$

Continuando,

$$F^2 \perp \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ (F \perp)\left([\sigma \mid x : \sigma\ x + 1]\right) & \sigma\ x < 2 \wedge \sigma\ y = 0 \\ (F \perp)\sigma & \sigma\ x < 2 \wedge \sigma\ y \neq 0 \end{cases}$$

$$= \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : \sigma\ x + 1] & \sigma\ x = 1 \wedge y = 0 \\ \perp & \sigma\ x < 1 \wedge y = 0 \\ \perp & \sigma\ x < 2 \wedge \sigma\ y \neq 0 \end{cases}$$

$$= \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & \sigma\ x = 1 \wedge y = 0 \\ \perp & \sigma\ x < 1 \wedge y = 0 \\ \perp & \sigma\ x < 2 \wedge \sigma\ y \neq 0 \end{cases}$$

Solo para ser explícitos, veamos que

$$F^3 \perp \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ (F \perp)^2\left([\sigma \mid x : \sigma\ x + 1]\right) & \sigma\ x < 2 \wedge \sigma\ y = 0 \\ (F \perp)^2\sigma & \sigma\ x < 2 \wedge \sigma\ y \neq 0 \end{cases}$$

$$= \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & \sigma\ x \in \{0, 1\} \wedge y = 0 \\ \perp & \sigma\ x < 0 \wedge y = 0 \\ \perp & \sigma\ x < 2 \wedge \sigma\ y \neq 0 \end{cases}$$

Planteamos como hipótesis inductiva que

$$F^k \perp \sigma = \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & 3 - k \leq \sigma\ x \leq 1 \wedge y = 0 \\ \perp & c.c. \end{cases}$$

Entonces

$$
\begin{aligned}
F^{k+1} \perp \sigma &= \begin{cases} \sigma & \sigma\ x \geq 2 \\ (F \perp)^k ([\sigma \mid x : \sigma\ x + 1]) & \sigma\ x < 2 \wedge \sigma\ y = 0 \\ (F \perp)^k \sigma & \sigma\ x < 2 \wedge \sigma\ y \neq 0 \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : \sigma\ x + 1] & \sigma\ x < 2 \wedge \sigma\ x + 1 \geq 2 \wedge y = 0 \\ [\sigma \mid x : 2] & \sigma\ x < 2 \wedge 3 - k \leq \sigma\ x + 1 \leq 1 \wedge y = 0 \\ \perp & c.c. \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & \sigma\ x < 2 \wedge \sigma\ x \geq 1 \wedge y = 0 \\ [\sigma \mid x : 2] & \sigma\ x < 2 \wedge 3 - (k+1) \leq \sigma\ x \leq 0 \wedge y = 0 \\ \perp & c.c. \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & \sigma\ x = 1 \wedge y = 0 \\ [\sigma \mid x : 2] & 3 - (k+1) \leq \sigma\ x \leq 0 \wedge y = 0 \\ \perp & c.c. \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & 3 - (k+1) \leq \sigma\ x \leq 1 \wedge y = 0 \\ \perp & c.c. \end{cases}
\end{aligned}
$$

quod erat demonstrandum. Se sigue entonces que

$$
\bigsqcup_{i \in \mathbb{N}} F^i \perp = \lambda\sigma. \begin{cases} \sigma & \sigma\ x \geq 2 \\ [\sigma \mid x : 2] & \sigma\ x \leq 1 \wedge y = 0 \\ \perp & \sigma\ y \neq 0 \end{cases}
$$

Aclaración. Para no escribir tanto, agrupamos $\perp$ en un solo caso durante el desarrollo de $F^1 \perp, F^2 \perp$, etc. Pero debería ser claro que en uno de los casos damos $\perp$ porque la cantidad de iteraciones es limitada, mientras que en otro caso damos $\perp$ porque $\sigma\ y \neq 0$. En el primer caso, a medida que se aumentan las iteraciones, se añade más y más información y, en el límite, la indefinición desaparece. En el segundo caso, la indefinición no desaparece: siempre que $\sigma\ y \neq 0$, se da $\perp$.

(6) Asuma que $[\![\textbf{while } b \textbf{ do } c]\!]\sigma \neq \bot$. Demuestre (a) que existe $n \geq 0$ tal que $F^n \perp \sigma \neq \bot$. Demuestre (b) que si $\sigma' = [\![\textbf{while } b \textbf{ do } c]\!]\sigma$, entonces $\neg[\![b]\!]\sigma'$.

(a) Sabemos que

$$[\![\textbf{while } b \textbf{ do } c]\!] = \bigsqcup_{i \in \mathbb{N}} F^i \perp$$

para

$$F\ f\ \sigma = \begin{cases} \sigma & \neg[\![b]\!]\sigma \\ f[\![c]\!]\sigma & c.c. \end{cases}$$

Asuma que no existe $n \geq 0$ tal que $F^n \perp \sigma \neq \bot$. Se sigue que la cadena $\{F^i \perp\}_{i \in \mathbb{N}}$ es simplemente la cadena $\bot \sqsubseteq \bot \sqsubseteq \bot \sqsubseteq \ldots$. El supremo de esta cadena es $\bot$. Por lo tanto,

$$[\![\textbf{while } b \textbf{ do } c]\!] = \bigsqcup_{i \in \mathbb{N}} F^i \perp = \bot$$

lo cual contradice la hipótesis. La contradicción viene de asumir que no existe $n \geq 0$ tal que $F^n \perp \sigma \neq \bot$.

$\therefore$ Existe $n \geq 0$ tal que $F^n \perp \sigma \neq \bot$. $\blacksquare$

(b) Dado que la semántica de **while** b **do** c es un punto fijo de F, si usamos $\varphi := [\![\textbf{while } b \textbf{ do } c]\!]$, entonces

$$\varphi\ \sigma = F\ \varphi\ \sigma$$

Si σ es tal que $\neg[\![b]\!]\sigma$, entonces se sigue inmediatamente de la definición de F que en el estado $\varphi\sigma$ no se cumple b. Veamos el caso en que se cumple $[\![b]\!]\sigma$. Por la definición de F,

$$\varphi\ \sigma = \varphi\,(\varphi \ldots (\varphi[\![c]\!]\sigma)) = \varphi^k[\![c]\!]\sigma \tag{2}$$

donde la hipótesis de que el ciclo nunca es $\perp$ nos permite garantizar que existe tal $k \in \mathbb{N}$. Ahora bien, por la definición de F, k es definido estrictamente por el hecho de que

$$\neg[\![b]\!](\varphi^k[\![c]\!]\sigma)$$

Por la ecuación (2), resulta entonces

$$\neg[\![b]\!]\,(\varphi\sigma) \quad \blacksquare$$

(7) Demostrar o refutar:

(a) **while false do** $c \equiv$ **skip**

(b) **while** b **do** $c \equiv$ **while** b **do** $(c; c)$

(c) (**while** b **do** c); **if** b **then** c_0 **else** $c_1 \equiv$ (**while** b **do** c); c_1

(a) Es trivial.

(b) Falso. Basta dar un contraejemplo. Sea σ un estado con $\sigma\ x = 0$ y considere

$$w_1 := \textbf{while } x \leq 0 \textbf{ do } x := x{+}1, \qquad w_2 := \textbf{while } x \leq 0 \textbf{ do } (x := x{+}1); (x := x{+}1)$$

Claramente, $[\![w_1]\!]\sigma\ x = 1$ y $[\![w_2]\!]\sigma\ x = 2$. Sin embargo, dados comandos c_1, c_2,

$$c_1 \equiv c_2 \iff \forall \sigma \in \Sigma : [\![c_1]\!]\sigma = [\![c_2]\!]\sigma$$

$\therefore\ w_1 \not\equiv w_2$.

(c) Es verdadero. En el ejercicio anterior, demostramos que si un ciclo termina, entonces la guarda no puede cumplirse en el estado resultante del ciclo. Es decir que si

$$[\![\textbf{while } b \textbf{ do } c]\!]\sigma = \sigma' \neq \perp$$

entonces $[\![b]\!]\sigma' \equiv$ **False**. Por lo tanto, asumiendo que $[\![$**while** b **do** $c]\!]\sigma$ termina y no es $\perp$,

$$
\begin{aligned}
&[\![(\textbf{while } b \textbf{ do } c); \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]\sigma \\
&= [\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!] \, ([\![\textbf{while } b \textbf{ do } c]\!]\sigma) \\
&= [\![c_1]\!] \, ([\![\textbf{while } b \textbf{ do } c]\!]\sigma) \\
&= [\![(\textbf{while } b \textbf{ do } c); c_1]\!]\sigma \qquad \blacksquare
\end{aligned}
$$

Ahora bien, si el ciclo no termina (es decir, si devuelve $\perp$), es trivial demostrar que la equivalencia también se cumple.

(8) Considerar las siguientes definiciones como syntactic sugar del comando

$$\textbf{for } v := e_0 \textbf{ to } e_1 \textbf{ do } c$$

(a) $v := e_0; \textbf{while } v \leq e_1 \textbf{ do } c; v := v + 1$

(b) $\textbf{newvar } v := e_0 \textbf{ in while } v \leq e_1 \textbf{ do } c; v := v + 1$

(c) $\textbf{newvar } w := e_1 \textbf{ in newvar } v := e_0 \textbf{ in while } v \leq w \textbf{ do } c; v := v + 1$

¿Es alguna satisfactoria? Justificar.

Recordemos que, al menos de acuerdo con Reynolds,

$$\begin{aligned} &\textbf{for } v := e_0 \textbf{ to } e_1 \textbf{ do } c \\ := &\textbf{ newvar } w := e_1 \textbf{ in newvar } v := e_0 \textbf{ in while } v \leq w \textbf{ do } (c; v := v + 1) \end{aligned}$$

que es la expresión (c). Para no ser tramposos, igual justificaremos por qué dicha definición es satisfactoria, llegado el momento.

(a) La definición es satisfactoria en el sentido de que, si se la llama en un estado σ, ejecutará el comando c en los sucesivos estados

$$\begin{array}{l} [\sigma \mid v : [\![e_0]\!]\sigma] \\ [\sigma \mid v : [\![e_0]\!]\sigma + 1] \\ \vdots \\ [\sigma \mid v : [\![e_1]\!]\sigma] \end{array}$$

Sin embargo, debemos notar que no se restaura el valor de v, i.e. v no es local al ciclo.

(b) Esta definición es funcional y restaura el valor de v. Sin embargo, es ineficiente, porque en cada llamada del while debe volver a computarse el valor de e_1 bajo el estado dado. Es concebible que e_1 sea una expresión compleja, e.g. una productoria de $k > 10.000$ variables, o cualquier locura que se nos ocurra. Por lo tanto, lo ideal sería computar la cota superior e_1 una sola vez y alocar dicho valor en otra variable.

(c) La definición (c) resuelve el problema de la (b), porque aloca en la variable local w el valor de la cota superior, que por lo tanto se computa una única vez. Una vez dicho valor es asignado a w, procede igual que en la def. (b): asigna a una variable local v el valor de e_0 e itera adecuadamente.

(9) Enunciar el teorema de coincidencia y demostra el caso **while**.

Teorema de coincidencia.

(a) Sean σ, σ' estados tales que $\sigma\ w = \sigma'\ w$ para toda $w \in FV(c)$. Entonces o bien $[\![c]\!]\sigma = [\![c]\!]\sigma' = \perp$ o bien $[\![c]\!]\sigma\ w = [\![c]\!]\sigma'\ w$ para toda $w \in FV(c)$.

(b) Si $[\![c]\!]\sigma \neq \perp$, entonces $[\![c]\!]\sigma\ w = \sigma\ w$ para toda $w \notin FA(c)$.

(a) Sean σ_1, σ_2 tales que $\sigma_1\ w = \sigma_2\ w$ para todo $w \in FV(c)$, donde

$$c := \textbf{while}\ b\ \textbf{do}\ d$$

Por def. de FV, tenemos que $\sigma_1\ w = \sigma_2\ w$ para toda $w \in FV(b) \cup FV(d)$. Asumamos como hipótesis inductiva que el teorema vale para b y d, y definamos

$$\begin{aligned} \gamma_1 &:= [\![d]\!]\sigma_1, & \gamma_{i+1} &:= [\![d]\!]\gamma_i \\ \beta_1 &:= [\![d]\!]\sigma_2, & \beta_{i+1} &:= [\![d]\!]\beta_i \end{aligned}$$

Es decir, $\{\gamma_i\}$ y $\{\beta_i\}$ son los estados correspondientes a las sucesivas iteraciones del **while**. Observemos que por HI resulta que $\gamma_1 = [\![d]\!]\sigma_1, \beta_1 = [\![d]\!]\sigma_2$ coinciden en las variables libres de d. Es fácil ver por inducción que entonces γ_i, β_i coinciden en las variables libres de d para toda i.

Hagamos una subdemostración de $(\star)$ $[\![b]\!]\gamma_i = [\![b]\!]\beta_i$.

> $(\star)$ Una ejecución de d sólo afecta la semántica de b a través de modificaciones de las variables en $FV(b) \cap FV(d) \subseteq FV(d)$. Pues

$\gamma_k\ w = \beta_k\ w$ para toda $w \in FV(d)$, esto vale en particular para toda $w \in FV(d) \cap FV(b)$.

$\therefore$ Si $w \in FV(b) \cap FV(d)$, entonces $\gamma_k\ w = \beta_k\ w$.

Si $w \in FV(b) - FV(d)$, entonces ninguna ejecución de d afecta el valor de w.

$\therefore$ Si $w \in FV(b) - FV(d)$, entonces $\gamma_k\ w = \sigma_1\ w, \beta_k\ w = \sigma_2\ w$, y por hipótesis $\sigma_1\ w = \sigma_2\ w$.

$\therefore \quad \forall w \in FV(b), k \in \mathbb{N} : \gamma_k\ w = \beta_k\ w$.

Como la semántica de b depende únicamente de el valor de sus variables libres, se sigue que $[\![b]\!]\gamma_k = [\![b]\!]\beta_k$ para todo $k \in \mathbb{N}$.

Asuma que $[\![c]\!]\sigma_1 = \perp$. Entonces, para toda i se cumple que $[\![b]\!]\gamma_i \equiv$ **True** (de otro modo el **while** terminaría). Por $(\star)$ se sigue que $[\![b]\!]\beta_i \equiv$ **True**. Como esto vale para toda i, las sucesivas iteraciones de $\{\beta_i\}$ nunca hacen la guarda falsa. Por lo tanto, el **while** nunca termina partiendo desde σ_2. $\therefore [\![c]\!]\sigma_2 = \perp$

Asuma que $[\![c]\!]\sigma_1 \neq \perp$. Un razonamiento idéntico al anterior nos da que $[\![c]\!]\sigma_2 \neq \perp$, y no sólo eso sino que se da la misma cantidad k de iteraciones en ambos casos. Es decir que los estados finales de ambos casos son γ_k, β_k, respectivamente. Ya observamos antes que γ_k, β_k coinciden en las variables libres de d y de b, lo cual concluye la prueba.

Demostración alternativa. Sean σ_1, σ_2 definidos como antes y valga la misma hipótesis inductiva. Vamos por casos.

(Caso $[\![\textbf{while}\ b\ \textbf{do}\ c]\!]\sigma_1 \neq \perp$). Sea $\pi := [\![\textbf{while}\ b\ \textbf{do}\ c]\!]$. Por el ejercicio **(6)**, sabemos que

- Existe $k \geq 0$ tal que $F^k \perp \sigma_1 \neq \perp$,
- $\neg[\![b]\!](\pi\ \sigma_1)$

Sabemos que $\pi = \bigsqcup_{i \in \mathbb{N}} F^i \perp \neq \perp$. Sea

$$k_0 := \min_k \left\{ F^k \perp : F^k \perp \sigma_1 \neq \perp \right\}$$

Entonces $\pi\ \sigma_1 = [\![c]\!]^{k_0 - 1}\ \sigma_1$. Probemos que $\pi\ \sigma_1\ w = \pi\ \sigma_2\ w$ para toda $w \in FV(b) \cup FV(c)$.

(10) Usando el Teorema de coincidencia para comandos, probar que para todo par de comandos c_0, c_1, si

$$FV(c_0) \cap FA(c_1) = FV(c_1) \cap FA(c_0) = \emptyset$$

entonces $[\![c_0; c_1]\!] = [\![c_1; c_0]\!]$

Veamos el caso $[\![c_0; c_1]\!] \neq \perp$, pues el caso en que el comando da $\perp$ es t rivial.

Asuma que $FV(c_0) \cap FA(c_1) = FV(c_1) \cap FA(c_0) = \emptyset$. Es decir, a ninguna variable libre de c_0 se le asigna un valor en c_1, y a ninguna variable libre de c_1 se le asigna un valor en c_0. Entonces, por inciso (b) del teorema de coincidencia,

$$\forall w \in FV(c_1) : [\![c_0]\!]\ \sigma\ w = \sigma\ w$$

Luego, por inciso (a) del teorema de coincidencia,

$$\forall w \in FV(c_1) : [\![c_1]\!]\ ([\![c_0]\!]\ \sigma)\ \ w = [\![c_1]\!]\ \sigma\ w$$

$\therefore\ \ \forall w \in FV(c_1) : [\![c_0; c_1]\!]\ \sigma\ w = [\![c_1]\!] \sigma\ w.$

De acuerdo con el mismo razonamiento, aplicando inciso (b) y luego inciso (a) del teorema de coincidencia pero ahora para el caso $w \in FV(c_0)$, obtenemos:

$\therefore\ \forall w \in FV(c_0) : [\![c_1; c_0]\!]\ \sigma\ w = [\![c_0]\!]\sigma\ w.$

Ahora consideremos $w \notin FV(c_1)$. Es claro entonces que $w \notin FA(c_1)$ y por lo tanto $[\![c_1]\!]\gamma\ w$ para todo γ. Por lo tanto,

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

$\therefore\ \forall w \notin FV(c_1) : [\![c_0; c_1]\!]\sigma = [\![c_0]\!]\sigma\ w.$

De acuerdo con el mismo razonamiento,

$\therefore\ \forall w \notin FV(c_0) : [\![c_1; c_0]\!]\sigma = [\![c_1]\!]\sigma\ w.$

Reunamos entonces todo lo que hemos concluido:

$$\begin{array}{l}
\forall w \in FV(c_1) : [\![c_0; c_1]\!]\ \sigma\ w = [\![c_1]\!]\sigma\ w. \\
\forall w \notin FV(c_1) : [\![c_0; c_1]\!]\sigma = [\![c_0]\!]\sigma\ w. \\
\forall w \in FV(c_0) : [\![c_1; c_0]\!]\ \sigma\ w = [\![c_0]\!]\sigma\ w. \\
\forall w \notin FV(c_0) : [\![c_1; c_0]\!]\sigma = [\![c_1]\!]\sigma\ w.
\end{array}$$

Sea $w_0 \in FV(c_1) \cup FV(c_0)$. De las proposiciones arriba se sigue

$$[\![c_0; c_1]\!]\ \sigma\ w_0 = \begin{cases} [\![c_0]\!]\sigma\ w_0 & w_0 \notin FV(c_1) \\ [\![c_1]\!]\sigma\ w_0 & w_0 \in FV(c_1) \end{cases}$$

$$[\![c_1; c_0]\!]\ \sigma\ w_0 = \begin{cases} [\![c_1]\!]\sigma\ w_0 & w_0 \notin FV(c_0) \\ [\![c_0]\!]\sigma\ w_0 & w_0 \in FV(c_0) \end{cases}$$

Pero como para $w_0 \in FV(c_1) \cup FV(c_0)$ tenemos que $w_0 \notin FV(c_1) \iff w_0 \in FV(c_0)$, y lo inverso también, entonces la segunda ecuación es:

$$[\![c_1; c_0]\!]\ \sigma\ w_0 = \begin{cases} [\![c_1]\!]\sigma\ w_0 & w_0 \in FV(c_1) \\ [\![c_0]\!]\sigma\ w_0 & w_0 \notin FV(c_1) \end{cases} = [\![c_0; c_1]\!]\sigma\ w_0$$

Como esto vale para toda variable $w_0 \in FV(c_1) \cup FV(c_0)$ y para todo σ,

$$[\![c_0; c_1]\!] = [\![c_1; c_0]\!]$$

(12) Considere

$$f_i\ \sigma = \begin{cases} \sigma & \sigma\ x \leq \sigma\ y \\ \bot & c.c. \end{cases}$$

Decida si existe un programa $\mathscr{P}$ tal que $\mathscr{P} = \bigsqcup_{i\in\mathbb{N}} f_i$.

Para todo $k \in \mathbb{N}$, tenemos $f_k = f_{k+1}$ y por lo tanto la cadena $f_1, f_2, \ldots$ es no interesante.

$$\therefore \bigsqcup_{i\in\mathbb{N}} f_i = \lambda\sigma. \begin{cases} \sigma & \sigma\ x \leq \sigma\ y \\ \bot & c.c. \end{cases}$$

Existen infinitos programas cuya semántica equivale a la función dada arriba, e.g.

if $x \leq y$ **then skip else while true do skip**

7 Práctico 5: Fallas

Dado $\mathscr{P}$ definido como

$$\begin{array}{l}\textbf{newvar } x := y + x \textbf{ in} \\ \quad \textbf{while } x > 0 \textbf{ do if } x > 0 \textbf{ then skip else fail}\end{array}$$

Caracterizar (sin necesariamente calcular) los estados $\sigma \in \Sigma$ en que $\mathscr{P}\sigma \equiv$ **skip**.

En el contexto del **while**, dado un estado inicial σ, la variable x toma el valor $\sigma\ x + \sigma\ y$. Si $\sigma\ x + \sigma\ y > 0$ se ejecuta el **while**, y se ejecuta indefinidamente **skip**. Si $\sigma\ x + \sigma\ y \leq 0$ el programa falla.

$$\therefore \{\sigma \in \Sigma : \mathscr{P}\ \sigma \equiv \textbf{skip}\} = \emptyset$$

Demostrar o refutar las siguientes equivalencias formalmente.

(a) $c;$ **while true do skip** $\equiv$ **while true do skip**

(b) $c;$ **fail** $\equiv$ **fail**

(c) **newvar** $v := e$ **in** $v := v + 1;$ **fail** $\equiv$ **newvar** $w := e$ **in** $w := w + 1;$ **fail**

(d) **while** b **do fail** $\equiv$ **if** b **then fail else skip**

(e) $x := 0;$ **catch** $x := 1$ **in while** $x < 1$ **do fail** $\equiv$ $x := 0;$ **while** $x < 1$ **do catch** $x := 1$ **in fail**

(a) Es fácil probar que $[\![\textbf{while true do skip}]\!] = \perp$ (creo que incluso es ejercicio de un práctico anterior). De esto se sigue fácilmente el resultado.

(b) Esta equivalencia es falsa porque, al llamar **fail** después de c, puede suceder que el programa falle dentro de c, o sea indefinido, sin alcanzar el **fail** final. Más formalmente,

$$
\begin{aligned}
[\![c; \textbf{fail}]\!]\sigma &= [\![\textbf{fail}]\!]_* ([\![c]\!]\sigma) \\
&= \begin{cases} \langle \textbf{fail}, [\![c]\!]\sigma \rangle & [\![c]\!]\sigma \in \Sigma \\ \langle \textbf{fail}, \sigma' \rangle & [\![c]\!]\sigma = \langle \textbf{fail}, \sigma' \rangle \\ \perp & [\![c]\!]\sigma = \perp \end{cases}
\end{aligned}
$$

Mientras que

$$[\![\textbf{fail}]\!]\sigma = \langle \textbf{fail}, \sigma \rangle$$

(c) Sean

$$\mathscr{P} := \textbf{newvar } v := e \textbf{ in } v := v + 1$$
$$\mathbb{Q} := \textbf{newvar } w := e \textbf{ in } w := w + 1$$

Debería ser claro que la semántica de $\mathscr{P}$ y $\mathbb{Q}$ son la misma. Para demostrarlo rigurosamente, basta observar que

$$\mathbb{Q} = \textbf{newvar } w := e \textbf{ in } (v := v + 1/v \to w)$$

donde $w \notin FV(\mathscr{P}) - \{v\}$. Luego, por teorema de renombre,

$$[\![\mathscr{P}]\!] = [\![\mathbb{Q}]\!]$$

$\therefore [\![\mathscr{P}; \textbf{fail}]\!] = [\![\textbf{fail}]\!]_* ([\![\mathscr{P}]\!]\sigma) = [\![\textbf{fail}]\!]_* ([\![\mathbb{Q}]\!]\sigma) = [\![\mathbb{Q}; \textbf{fail}]\!]$.

Es decir, la equivalencia es verdadera.

(d) Sabemos que la semántica de **while** b **do fail** satisface la ecuación

$$\begin{aligned} F\ f\ \sigma &= \begin{cases} \sigma & \neg[\![b]\!]\sigma \\ f_* ([\![\textbf{fail}]\!]\sigma) & c.c. \end{cases} \\ &= \begin{cases} \sigma & \neg[\![b]\!]\sigma \\ f_* (\langle \textbf{abort}, \sigma \rangle) & c.c. \end{cases} \end{aligned}$$

y que es dada por $\bigsqcup_{i \in \mathbb{N}} F^i \perp$. Claramente,

$$F \perp = \lambda\sigma. \begin{cases} \sigma & \neg[\![b]\!]\sigma \\ \perp & c.c. \end{cases}$$

$$
\begin{aligned}
F^2 \perp \sigma &= \begin{cases} \sigma & \neg[\![b]\!]\sigma \\ (F \perp)_* (\langle \mathbf{abort}, \sigma \rangle) & c.c. \end{cases} \\
&= \begin{cases} \sigma & \neg[\![b]\!]\sigma \\ \langle \mathbf{abort}, \sigma \rangle & c.c. \end{cases}
\end{aligned}
$$

donde $(F \perp)_* (\langle \mathbf{abort}, \sigma \rangle) = \langle \mathbf{abort}, \sigma \rangle$ por la definición de la extensión f_* para una función f. Debería ser claro entonces que $F^1 \perp = F^2 \perp = \ldots$, es decir que la cadena $\{F^i \perp\}$ es no interesante con supremo

$$
\bigsqcup_{i \in \mathbb{N}} F^i \perp = \lambda\sigma. \begin{cases} \sigma & \neg[\![b]\!]\sigma \\ \langle \mathbf{abort}, \sigma \rangle & c.c. \end{cases}
$$

Por def. esta es la semántica de $[\![\mathbf{if}\ b\ \mathbf{then\ fail\ else\ skip}]\!]$. Por lo tanto, la equivalencia es verdadera.

(e) Sean

$$
\begin{aligned}
\mathscr{P} &:= \mathbf{catch}\ x := 1\ \mathbf{in\ while}\ x < 1\ \mathbf{do\ fail} \\
\mathbb{Q} &:= \mathbf{while}\ x < 1\ \mathbf{do\ catch}\ x := 1\ \mathbf{in\ fail}
\end{aligned}
$$

Deseamos estudiar si

$$
x := 0; \mathscr{P} \equiv x := 0; \mathbb{Q}
$$

Intuitivamente, la equivalencia debería ser cierta, porque ambos programas, partiendo de un estado σ, terminan en un estado $[\sigma \mid x : 1]$. Veámoslo formalmente.

$$
\begin{aligned}
[\![x := 0; \mathscr{P}]\!]\sigma &= [\![\mathscr{P}]\!][\sigma \mid x : 0] \\
&= [\![x := 1]\!]_+ ([\![\textbf{while } x < 1 \textbf{ do fail}]\!][\sigma \mid x : 0])
\end{aligned}
$$

Ahora bien, en (d) ya vimos que

$$
[\![\textbf{while } b \textbf{ do fail}]\!] = \lambda\sigma. \begin{cases} \sigma & \neg[\![b]\!]\sigma \\ \langle \textbf{abort}, \sigma \rangle & c.c. \end{cases}
$$

Se sigue entonces que

$$
\begin{aligned}
[\![x := 1]\!]_+ ([\![\textbf{while } x < 1 \textbf{ do fail}]\!][\sigma \mid x : 0]) &= [\![x := 1]\!]_+ \langle \textbf{abort}, \sigma \rangle \\
&= [\![x := 1]\!]\sigma \\
&= [\sigma \mid x : 1]
\end{aligned}
$$

Por otra parte,

$$
[\![x := 0; \mathbb{Q}]\!]\sigma = [\![\textbf{while } x < 1 \textbf{ do catch } x := 1 \textbf{ in fail}]\!][\sigma \mid x : 0]
$$

Veamos que

$$
\begin{aligned}
F\ f\ \sigma &= \begin{cases} \sigma & \sigma\ x \geq 1 \\ f([\![\textbf{catch}\ x := 1\ \textbf{in}\ \textbf{fail}]\!]\sigma) & c.c. \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 1 \\ f([\![x := 1]\!]_+\,([\![\textbf{fail}]\!]\sigma)) & c.c. \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 1 \\ f([\![x := 1]\!]_+\,\langle \textbf{abort}, \sigma\rangle) & c.c. \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 1 \\ f\,([\![x := 1]\!]\sigma) & c.c. \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 1 \\ f[\sigma \mid x : 1] & c.c. \end{cases}
\end{aligned}
$$

Por ende

$$
F\ \perp\ \sigma = \begin{cases} \sigma & \sigma\ x \geq 1 \\ \perp & c.c. \end{cases}
$$

$$
\begin{aligned}
F^2 \perp \sigma &= \begin{cases} \sigma & \sigma\ x \geq 1 \\ F \perp [\sigma \mid x : 1] \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 1 \\ [\sigma \mid x : 1] & \sigma\ x < 1 \wedge [\sigma \mid x : 1]\ x \leq 1 \\ \perp & \sigma\ x < 1 \wedge [\sigma \mid x : 1]\ x > 1 \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 1 \\ [\sigma \mid x : 1] & \sigma\ x < 1 \wedge 1 \leq 1 \\ \perp & \sigma\ x < 1 \wedge 1 > 1 \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 1 \\ [\sigma \mid x : 1] & \sigma\ x < 1 \wedge \mathbf{True} \\ \perp & \mathbf{False} \end{cases} \\
&= \begin{cases} \sigma & \sigma\ x \geq 1 \\ [\sigma \mid x : 1] & \sigma\ x < 1 \end{cases}
\end{aligned}
$$

Es fácil notar que la cadena $\{F^i \perp\}_{i\in\mathbb{N}}$ será no interesante con supremo

$$
[\![\mathbf{while}\ x < 1\ \mathbf{do}\ \mathbf{catch}\ x := 1\ \mathbf{in}\ \mathbf{fail}]\!] = \bigsqcup_{i\in\mathbb{N}} F^i \perp = \lambda\sigma. \begin{cases} \sigma & \sigma\ x \geq 1 \\ [\sigma \mid x : 1] & \sigma\ x < 1 \end{cases}
$$

Por lo tanto,

$$
\begin{aligned}
[\![x := 0; \mathbb{Q}]\!]\sigma &= [\![\mathbf{while}\ x < 1\ \mathbf{do}\ \mathbf{catch}\ x := 1\ \mathbf{in}\ \mathbf{fail}]\!][\sigma \mid x : 0] \\
&= \left(\lambda\sigma. \begin{cases} \sigma & \sigma\ x \geq 1 \\ [\sigma \mid x : 1] & \sigma\ x < 1 \end{cases}\right) [\sigma \mid x : 0] \\
&= [\sigma \mid x : 1]
\end{aligned}
$$

que es lo que esperábamos.

$$\therefore \; [\![x := 0; \mathscr{P}]\!] = [\![x := 0; \mathbb{Q}]\!]$$

8 Práctico 6

8.1 Problemas

(3) Demostrar o refutar:

(a) $?x; ?y \equiv ?y; ?x$

(b) $?x; z := x \equiv ?z$

(c) **newvar** $x := e$ **in** $(?x; z := x) \equiv ?z$

(a) Recordemos que

$$[\![?v]\!]\sigma = \iota_{\text{in}}\,(\lambda k \in \mathbb{Z}\;.\;\iota_{\text{term}}[\sigma \mid v : k]) \tag{3}$$

También recordemos que

$$\iota_{\text{in}} = \psi \circ \iota_{\uparrow} \circ \iota_2 \in (\mathbb{Z} \to \Omega) \to \Omega$$

Finalmente, recordemos que

$$f_*(\iota_{\text{in}}\ g) = \iota_{\text{in}}\,(\lambda k \in \mathbb{Z}\;.\;f_*\ g\ k) \tag{4}$$

Entonces

$$
\begin{aligned}
[\![?x;?y]\!]\sigma &= [\![?y]\!]_* \left([\![?x]\!]\sigma\right) \\
&= [\![?y]\!]_* \left(\iota_{\text{in}} \left(\lambda k \in \mathbb{Z}\ .\ \iota_{\text{term}}[\sigma \mid x : k]\right)\right) && \{\text{Por } (3)\} \\
&= \iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}\ .\ [\![?y]\!]_* \left(\iota_{\text{term}}[\sigma \mid x : k]\right)\right) && \{\text{Por } (4)\} \\
&= \iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}\ .\ [\![?y]\!]_* \left(\langle[\sigma \mid x : k]\rangle\right)\right) && \{\iota_{\text{term}}\gamma = \langle\gamma\rangle\} \\
&= \iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}. \langle[\![?y]\!][\sigma \mid x : k]\rangle\right) && \{\text{Def. de } f_*\} \\
&= \iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}\ .\ \langle\iota_{\text{in}} \left(\lambda n \in \mathbb{Z}\ .\ [[\sigma \mid x : k] \mid y : n]\right)\rangle\right) \\
&= \iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}\ .\ \langle\iota_{\text{in}} \left(\lambda\ n \in \mathbb{Z}\ .\ [\sigma \mid x : k, y : n]\right)\rangle\right)
\end{aligned}
$$

Ahora bien, el mismo razonamiento nos da

$$
\begin{aligned}
[\![?y;?x]\!]\sigma &= [\![?x]\!]_* \left([\![?y]\!]\sigma\right) \\
&= [\![?x]\!]_* \left(\iota_{\text{in}} \left(\lambda k \in \mathbb{Z}\ .\ \iota_{\text{term}}[\sigma \mid y : k]\right)\right) && \{\text{Por } (3)\} \\
&= \iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}\ .\ [\![?x]\!]_* \left(\iota_{\text{term}}[\sigma \mid y : k]\right)\right) && \{\text{Por } (4)\} \\
&= \iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}\ .\ [\![?x]\!]_* \left(\langle[\sigma \mid y : k]\rangle\right)\right) && \{\iota_{\text{term}}\gamma = \langle\gamma\rangle\} \\
&= \iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}. \langle[\![?x]\!][\sigma \mid y : k]\rangle\right) && \{\text{Def. de } f_*\} \\
&= \iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}\ .\ \langle\iota_{\text{in}} \left(\lambda n \in \mathbb{Z}\ .\ [[\sigma \mid y : k] \mid x : n]\right)\rangle\right) \\
&= \iota_{in} \left(\lambda\ k \in \mathbb{Z}\ .\ \langle\iota_{\text{in}} \left(\lambda\ n \in \mathbb{Z}\ .\ [\sigma \mid y : k, x : n]\right)\rangle\right)
\end{aligned}
$$

Claramente, ambas funciones son iguales.

(b) Ahora abreviaremos un poco el desarrollo y notaremos simplemente que

$$
\begin{aligned}
[\![?x; z := x]\!] &= [\![z := x]\!]_* \left(\iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}. \langle[\sigma \mid x : k]\rangle\right)\right) \\
&= \iota_{\text{in}} \left(\lambda k \in \mathbb{Z}\ .\ \langle[\![z := x]\!][\sigma \mid x : k]\rangle\right) \\
&= \iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}\ .\ \langle[\sigma \mid x : k, z : k]\rangle\right) \\
&\neq \iota_{\text{in}} \left(\lambda\ k \in \mathbb{Z}\ .\ \langle[\sigma \mid z : k]\rangle\right) \\
&= [\![?z]\!]\sigma
\end{aligned}
$$

Por lo tanto, la equivalencia es falsa.

(c) Veamos que

$$\begin{aligned}
[\![\textbf{newvar } x := e \textbf{ in } (?x; z := x)]\!]\sigma &= \mathscr{R}_{\dagger}\left([\![?x; z := x]\!][\sigma \mid x : [\![e]\!]\sigma]\right)\\
&= \mathscr{R}_{\dagger}\left(\iota_{\text{in}}\left(\lambda\ k \in \mathbb{Z}\ .\ \langle \sigma \mid x : k, z : k\rangle\right)\right) \quad \{\text{Por inciso } (b)\}\\
&= \iota_{\text{in}}\left(\lambda\ k \in \mathbb{Z}\ .\ \mathscr{R}\left\langle \sigma \mid x : k, z : k\right\rangle\right)\\
&= \iota_{\text{in}}\left(\lambda\ k \in \mathbb{Z}.\left\langle \sigma \mid z : k\right\rangle\right)\\
&= [\![?z]\!]\sigma
\end{aligned}$$

donde

$$\mathscr{R} := \lambda\sigma' \in \Sigma\ .\ ([\sigma' \mid x : \sigma\ x])$$

es la función de restauración respecto al σ pasado como argumento. Tenemos entonces que la equivalencia es verdadera.

(4) Sea $\mathscr{P}$ un programa que no incluye fallas, outputs, ni inputs. Asuma que $\{x, y\} \cap FV(c) = \emptyset$. Determine la validez de

$$?x; \mathscr{P}; !x \equiv ?y; \mathscr{P}; !y$$

Dado que $\mathscr{P}$ no incluye fallas, outputs, ni inputs, el teorema coincidencia para comandos aplica para $\mathscr{P}$. Por lo tanto,

$$[\![\mathscr{P}]\!]\ \sigma\ x = \sigma\ x, \qquad [\![\mathscr{P}]\!]\sigma\ y = \sigma\ y$$

para todo $\sigma \in \Sigma$. En particular,

$$\begin{aligned}
[\![?x; \mathscr{P}]\!]\sigma &= [\![\mathscr{P}]\!]_* \left([\![?x]\!]\sigma\right) \\
&= [\![\mathscr{P}]\!]_* \Big(\iota_{\text{in}} \left(\lambda\ k\ .\ \iota_{\text{term}}[\sigma \mid x : k]\right)\Big) \\
&= \iota_{\text{in}} \Big(\lambda k\ .\ [\![\mathscr{P}]\!]_* \left\langle [\sigma \mid x : k] \right\rangle\Big) \\
&= \iota_{\text{in}} \Big(\lambda k\ .\ \Big\langle [\![\mathscr{P}]\!][\sigma \mid x : k] \Big\rangle\Big)
\end{aligned}$$

Por lo tanto,

$$\begin{aligned}
[\![?x; \mathscr{P}; !x]\!]\sigma &= [\![!x]\!]_* \Big(\iota_{\text{in}} \Big(\lambda\ k \Big\langle [\![\mathscr{P}]\!][\sigma \mid x : k] \Big\rangle\Big)\Big) \\
&= \iota_{\text{in}} \Big(\lambda k\ .\ \Big\langle [\![x!]\!] \left([\![\mathscr{P}]\!][\sigma \mid x : k]\right) \Big\rangle\Big) \\
&= \iota_{\text{in}} \Big(\lambda\ k \left\langle [\![\mathscr{P}]\!][\sigma \mid x : k]\ x, [\![\mathscr{P}]\!][\sigma \mid x : k] \right\rangle\Big) \\
&= \iota_{\text{in}} \left(\lambda\ k.\ \left\langle [\sigma \mid x : k]\ x, [\![\mathscr{P}]\!][\sigma \mid x : k] \right\rangle\right) \\
&= \iota_{\text{in}} \left(\lambda\ k.\ \left\langle k, [\![\mathscr{P}]\!][\sigma \mid x : k] \right\rangle\right)
\end{aligned}$$

El mismo desarrollo nos hace ver que

$$[\![?y; \mathscr{P}; !y]\!]\sigma = \iota_{\text{in}}\,(\lambda\ k\ .\ \langle k, [\![\mathscr{P}]\!][\sigma \mid y : k]\rangle)$$

Entonces es claro que la equivalencia es falsa. El primer programa transforma x en algún k vía input y escribe k, el segundo transforma y en algún k y escribe k. Aunque ambos escriben el k dado via input, cambian variables distintas.

(5) Considere:

$$\begin{aligned}
\omega_1 &:= \iota_{\text{in}}\left(\lambda n.\iota_{\text{out}}(n, \perp)\right) \\
\omega_2 &:= \iota_{\text{out}}(0, \perp) \\
\omega_3 &:= \iota_{\text{in}}(\lambda n.\perp) \\
\omega_4 &:= \iota_{\text{in}}\ f \text{ con } f\ n = \begin{cases} \perp & n < 0 \\ \iota_{\text{out}}(n, \perp) & \text{c.c.} \end{cases}
\end{aligned}$$

Describa mediante un diagrama de Hasse las relaciones de orden que se establecen entre los elementos antedichos de Ω.

Estudiemos uno por uno los elementos dados de Ω. La estrategia será mirarlos como elementos de $\left(\hat{\Sigma} + \mathbb{Z} \times \Omega + \mathbb{Z} \to \Omega\right)_\perp$, no como elementos de Ω. Para ello recordemos que el isomorfismo entre ambos dominios es dado por

$$\Omega \underset{\psi}{\overset{\phi}{\rightleftarrows}} \left(\hat{\Sigma} + \mathbb{Z} \times \Omega + \mathbb{Z} \to \Omega\right)_\perp$$

Lo primero que debemos observar es que ω_2 será incomparable con los demás elementos, porque $\iota_{\text{out}}, \iota_{\text{in}}$ tienen rangos disjuntos. Más específicamente, $\phi(\omega_2) \in \mathbb{Z} \times \Omega$, mientras que $\phi(\omega_i) \in \mathbb{Z} \to \Omega$ para $i = 1, 3, 4$.

Ahora bien, veamos que

$$\begin{aligned}
\phi(\omega_1) &= \lambda\ n.\iota_{\text{out}}(n, \perp) = \lambda n.\ \langle n \rangle \\
\phi(\omega_3) &= \lambda n.\perp = \perp_{\mathbb{Z} \to \Omega} \\
\phi(\omega_4) &= \lambda n. \begin{cases} \perp & n < 0 \\ \iota_{\text{out}}(n, \perp) & c.c. \end{cases}
\end{aligned}$$

Claramente, $\phi(\omega_3)$ es el mínimo entre estos tres elementos (es el bottom del espacio de funciones $\mathbb{Z} \to \Omega$). Por otro lado, para todo $n \in \mathbb{Z}$ se cumple que $\phi(\omega_4)(n) \sqsubseteq_\Omega \phi(\omega_1)(n)$, porque si $n < 0$ tenemos $\perp \sqsubseteq_\Omega \langle n \rangle$ y si $n \geq 0$ tenemos $\langle n \rangle \sqsubseteq_\Omega \langle n \rangle$.

Los isomorfismos preservan el orden y $\psi(\phi(\omega_i)) = \omega_i$. Por lo tanto, vistos como elementos de Ω, se sigue de lo anterior que ω_3 es el mínimo de $\{\omega_1, \omega_3, \omega_4\}$, y que $\omega_4 \sqsubseteq_\Omega \omega_1$. Por lo tanto, el diagrama de Hasse del predominio

$$(\{\omega_1, \omega_2, \omega_3, \omega_4\}, \sqsubseteq_\Omega)$$

es

$$\begin{tikzcd} \omega_1 \arrow[d, dash] \\ \omega_4 \arrow[d, dash] \\ \omega_3 = \perp_{\mathbb{Z} \to \Omega} & \omega_2 \end{tikzcd}$$

Figure 4: Diagrama de Hasse del ejercicio **(5)**

(6) Dé un programa cuya semántica sea el supremo de la cadena

$$w_0 = \bot, \qquad w_{i+1} = \iota_{\text{in}}\Big(\lambda n. \iota_{\text{out}}\,(n, w_i)\,\Big)$$

Veamos que

$$\begin{aligned}
w_0\ n\ \omega &= \bot \\
w_1\ n\ \omega &= \iota_{\text{in}}\,(\lambda n. \iota_{\text{out}}(n, \bot)) = \iota_{\text{in}}(\lambda n.\, \langle n \rangle) = \iota_{\text{in}} \mathscr{O} \\
w_2 &= \iota_{\text{in}}\,(\lambda n. (\langle n \rangle +\!\!+\ \omega_1)) \\
&= \iota_{\text{in}}\Big(\lambda n. \big(\,\langle n \rangle +\!\!+\ \iota_{\text{in}}(\lambda m.\, \langle m \rangle)\big)\Big)
\end{aligned}$$

Detengámonos acá y pensemos lo que tenemos. w_0 no hace nada. w_1 escribe un n dado via input y se cuelga. w_2, por otro lado, concatena un n dado via input a la secuencia $\langle m \rangle$, donde m es dado via input, tras lo cual se cuelga. Evidentemente, cada elemento de la cadena se corresponde semánticamente con la escritura de un nuevo entero dado via input. Por lo tanto, el supremo de la cadena se corresponde con la semántica de

$$c := \textbf{while true do } ?x; !x$$

(7) Considere los programas de la forma **while true do** $(?x; c)$. La cadena $F^i \perp \sigma$ de la semántica de dicho **while**, ¿será siempre interesante en Ω? Justifique.

Veamos que

$$\begin{aligned}
[\![?x; c]\!]\sigma &= [\![c]\!]_*\Big([\![?x]\!]\sigma\Big) \\
&= [\![c]\!]_*\Big(\iota_{\text{in}}\,(\lambda\ k\ .\ [\sigma \mid x : k])\Big) \\
&= \iota_{\text{in}}\,\Big(\lambda\ k.\ \Big([\![c]\!]_*[\sigma \mid x : k]\Big)\Big) \\
&= \iota_{\text{in}}\,\Big(\lambda\ k.\ \Big([\![c]\!][\sigma \mid x : k]\Big)\Big)
\end{aligned}$$

Por lo tanto, la F en cuestión es

$$\begin{aligned}
F\ f\ \sigma &= f_*\,\Big(\iota_{\text{in}}\Big(\lambda\ k\ .\ ([\![c]\!][\sigma \mid x : k])\Big)\Big) \\
&= \iota_{\text{in}}\Big(\lambda\ k\ .\ f_*\,([\![c]\!][\sigma \mid x : k])\Big)
\end{aligned}$$

Claramente, $F \perp \sigma = \iota_{\text{in}}(\lambda\ k.\perp_\Omega)$ y se corresponde con pedir un input seguido de la no terminación.

Ahora bien,

$$\begin{aligned}
F^2 \perp \sigma &= \iota_{\text{in}}\,(\lambda\ k\ .(F\ \perp)_*([\![c]\!][\sigma \mid x : k])) \\
&= \iota_{\text{in}}\,(\lambda k.\iota_{\text{in}}(\lambda\ k.\perp_\Omega))
\end{aligned}$$

se corresponde con pedir un input, pedir otro input, y no terminar. En particular,

$$\lambda k.\perp_\Omega = \perp_{\mathbb{Z}\to\Omega} \sqsubseteq_{\mathbb{Z}\to\Omega} \lambda\ k.(\iota_{\text{in}}\perp_{\mathbb{Z}\to\Omega})$$

$$[\![?x]\!] = \iota_{\text{in}}\Big(\lambda k \in \mathbb{Z}.[\sigma \mid x : k]\Big)$$