Resumen de la materia
Lenguajes y Compiladores

Agustín Curto, agucurto95@gmail.com

2019

Índice

Nota: Este resumen se corresponde con la materia dictada en el año 2019. El autor no se responsabiliza de posibles cambios que pudiesen realizarse en los temas dictados en la misma, así como tampoco de errores involuntarios que pudiesen existir en dicho resumen.

Por favor, mejorá este documento en github
https://github.com/ResumenesFaMAF/resumenCompiladores

1. Semántica

Gramáticas

$$\begin{array}{rcl} \langle intexp \rangle & ::= & 0 \mid 1 \mid 2 \mid \ldots \\ & & \langle \mathsf{var} \rangle \\ & & - \langle intexp \rangle \\ & & \langle intexp \rangle + \langle intexp \rangle \\ & & \langle intexp \rangle * \langle intexp \rangle \\ & & \langle intexp \rangle - \langle intexp \rangle \\ & & \langle intexp \rangle \,/\, \langle intexp \rangle \\ & & \langle intexp \rangle \,\%\, \langle intexp \rangle \end{array}$$

$$\begin{array}{rcl} \langle assert \rangle & ::= & \mathbf{true} \mid \mathbf{false} \\ & & \langle intexp \rangle = \langle intexp \rangle \\ & & \langle intexp \rangle < \langle intexp \rangle \\ & & \langle intexp \rangle \leq \langle intexp \rangle \\ & & \langle intexp \rangle > \langle intexp \rangle \\ & & \langle intexp \rangle \geq \langle intexp \rangle \\ & & \neg \langle assert \rangle \\ & & \langle assert \rangle \vee \langle assert \rangle \\ & & \langle assert \rangle \wedge \langle assert \rangle \\ & & \exists \langle \mathsf{var} \rangle .\, \langle assert \rangle \\ & & \forall \langle \mathsf{var} \rangle .\, \langle assert \rangle \end{array}$$

Función Semántica

Es una función que a cada frase abstracta del lenguaje le asigna una denotación en un dominio determinado.

Todas las funciones son totales, es decir, que está definida para todas las expresiones.

Dirección por sintáxis

Un conjunto de ecuaciones es *dirigido por sintáxis* cuando se satisfacen las siguientes condiciones:

- hay una ecuación por cada producción de la gramática abstracta
- cada ecuación que expresa el significado de una frase compuesta, lo hace puramente en función de los significados de sus subfrases inmediatas

Composicionalidad

Una semántica se dice que es *composicional* cuando el significado de una frase no depende de ninguna propiedad de sus subfrases, salvo de sus significados.
Podemos reemplazar una subfrase e_0 de e por otra de igual significado que e_0, sin alterar el significado de e.

Dirección por sintáxis $\Rightarrow$ composicionalidad

Ligadura

- Ocurrencia ligadora: es la que se encuentra directamente después de un cuantificador.
- Alcance de una ocurrencia ligadora: en $Qv.p$, el predicado p es el alcance de la ocurrencia ligadora de v.
- Ocurrencia ligada: cualquier ocurrencia de v en el alcance de una ocurrencia ligadora de v.
- Ocurrencia libre: ocurrencia que no es ligadora ni ligada.
- Variable libre: variable que tiene ocurrencias libres.
- Expresión cerrada: sin variables libres.

Sustitución

$$
\begin{array}{rcl}
\Delta & = & \langle var \rangle \to \langle intexp \rangle \\
 & \in & \langle intexp \rangle \times \Delta \to \langle intexp \rangle \\
0/\delta & = & 0 \\
1/\delta & = & 1 \\
v/\delta & = & \sigma v \\
(-e)/\delta & = & -(e/\delta) \\
(e+f)/\delta & = & (e/\delta) + (f/\delta) \\
(Qv.b)/\delta & = & Qv_{new}.(b/[\delta|v : v_{new}]) \\
 & & \\
\text{donde } v_{new} & \notin & \bigcup\limits_{\omega \in FV(b-\{v\})} FV(\delta\omega)
\end{array}
$$

Propiedades

- **Teorema de Coincidencia:** expresa que el significado de una frase no puede depender de variables que no ocurran libres en la misma.

  Enunciado: Si dos estados σ, σ' coinciden en las variables libres de p, entonces da lo mismo evaluar p en σ o σ'.
  $$(\forall \omega \in FV(p).\sigma\omega = \sigma'\omega) \Rightarrow [\![p]\!]\sigma = [\![p]\!]\sigma'$$

- **Teorema de Renombre:** asegura que el significado no depende de las variables ligadas de una frase.

  Enunciado: Los nombres de las variables ligadas no tienen importancia.
  $$u \notin FV(q) - \{v\} \Rightarrow [\![\forall u.q/u \to v]\!] = [\![\forall v.q]\!]$$

- **Teorema de Sustitución:** Si aplico la sustitución δ a p y luego evaluo en el estado σ, puedo obtener el mismo resultado a partir de p sin sustituir si evaluo en un estado que hace el trabajo de δ y de σ (en las variables libres de p).
  $$(\forall \omega \in FV(p).[\![\delta\ \omega]\!]\sigma = \sigma'\omega) \Rightarrow [\![p/\delta]\!]\sigma = [\![p]\!]\sigma'$$

2. Recursión

- **Orden parcial:** Relación reflexiva, antisimétrica y transitiva.
- **Poset:** Par $(P, \leq)$, donde P es un conjunto y $\leq$ un orden parcial.
- **Orden Discreto:** $(X, =)$
- **Espacio ordenado de funciones:** $(Y,\ \leq_Y)$ poset $\Rightarrow (X \to Y,\ \leq)$ poset donde $f \leq g\ sii\ \forall x \in X.fx \leq gx$ para $f, g \in X \to Y$.
- **Lifting:** $(X,\ \leq_X)$ poset $\Rightarrow (X_\perp,\ \leq)$ poset donde $x \leq y\ sii\ x \leq_X y \vee x = \perp$

  Ejemplo: $\mathbb{Z}_\perp$

  $$
  \begin{array}{ccccccccc}
  \dots & -3 & -2 & -1 & 0 & 1 & 2 & 3 & \dots \\
   & & \dots & \backslash & | & / & \dots & & \\
   & & & & \perp & & & &
  \end{array}
  $$

- **Infinito:** $(X,\ \leq_X)$ poset $\Rightarrow (X^\infty,\ \leq)$ poset donde $x \leq y\ sii\ x \leq_X y \vee y = \infty$

Ejemplo: $\mathbb{N}^\infty$

$$\begin{array}{c}\infty\\ \vdots\\ 3\\ 2\\ 1\\ 0\end{array}$$

- **Supremo:** Sea $Q \subseteq P$ donde $(P,\ \leq)$ poset, el supremo se define:
  - $\forall q \in Q.q \leq sup(Q)$
  - $\forall p \in P.(\forall q \in Q.q \leq p) \Rightarrow sup(Q) \leq p$
- **Cadenas:** $p_0 \leq p_1 \leq p_2 \dots$
  - Interesantes: si $\{p_0, p_1, p_2, \dots\}$ es infinita.
  - No interesantes: si $\{p_0, p_1, p_2, \dots\}$ es finita o repite infinitamente un elemento.
- **Predominios:** es un poset donde todas las cadenas (interesantes) tienen supremo. (órdenes discretos y llanos)

  Si Y es predominio entonces $X \to Y$ también lo es.
- **Dominios:** es un predominio con elemento mínimo. (órdenes llanos)

  Si D es dominio entonces $X \to D$ también lo es.
- **Monotonía:** Sean $(P,\ \leq_P)$ poset y $(Q,\ \leq_Q)$ poset, y $f \in P \to Q$, f es *monótona* si:

$$x \leq_P y \Rightarrow fx \leq_Q fy \qquad \text{(preserva orden)}$$

- **Continuidad:** Sean P, Q con $\leq_P, \leq_Q$ y sup_P, sup_Q predominios y $f \in P \to Q$, se dice que f es *continua* si preserva supremos de cadenas, es decir, si $p_0 \leq_P p_1 \dots \leq_P p_n$ entonces el supremo $sup_Q(\{fp_i|i \in \mathbb{N}\})$ existe y $sup_Q(\{fp_i|i \in \mathbb{N}\}) = fsup_P(\{p_i|i \in \mathbb{N}\})$
- **Funciones Estrictas:** Sean D, D' dominios con $\perp, \perp'$ respectivamente. Se dice que la función $f \in D \to D'$ es *estricta* si f preserva el elemento mínimo, es decir, $f\perp = \perp'$.

<u>PROPIEDADES:</u>

- **Proposición 1:** Si f es monótona, f aplicada a los elementos de una cadena devuelve una cadena.
- **Proposición 2:** Si f es monótona, entonces f preserva el supremo de cadenas no interesantes.
- **Proposición 3:** Si la función $f \in P \to Q$ entre predominios es monótona entonces $sup_Q(\{fp_i|i \in \mathbb{N}\})$ existe y $sup_Q(\{fp_i|i \in \mathbb{N}\}) \leq_Q f\ sup_P(\{p_i|i \in \mathbb{N}\})$
- **Proposición 4:** Si f es continua, entonces f es monótona.

  La inversa, es decir, f monótona entonces f continua, solo vale para las cadenas no interesantes. Para las interesantes vale $sup_Q(\{fp_i|i \in \mathbb{N}\}) \leq_Q fsup_P(\{p_i|i \in \mathbb{N}\})$
- **Corolario:** Sean P, Q con $\leq_P, \leq_Q$ y sup_P, sup_Q predominios y $f \in P \to Q$ monótona, entonces f es continua sii si para toda cadena interesante $p_0 \leq_P p_1 \dots \leq_P p_n \leq_P \dots$, la desigualdad $f\ sup_P(\{p_i|i \in \mathbb{N}\}) \leq sup_Q(\{fp_i|i \in \mathbb{N}\})$ también vale.

TEOREMA DEL MENOR PUNTO FIJO

Teorema: Sea D un dominio, y $F \in D \to D$ continua, entonces $sup(F^i \bot)$ existe y es el menor punto fijo de F.
Prueba: Como $\bot$ es el elemento mínimo, $\bot \leq F\ \bot$. Como F es continua, F es monótona. Aplicando F a ambos lados obtenemos

$$F\ \bot \leq F\ (F\ \bot) = F^2\ \bot$$

Iterando esto obtenemos $\bot \leq F\ \bot \leq F^2\ \bot \leq F^3\ \bot \leq \ldots$, es decir que $\{F^i\ \bot | i \in \mathbb{N}\}$ es una cadena y por lo tanto el supremo $x = \sup(\{F^i\ \bot | i \in \mathbb{N}\})$ existe.
Veamos que es punto fijo de F, es decir, que $F\ x = x$:

$$\begin{array}{rcl} F\ x & = & F\ \sup(\{F^i\ \bot | i \in \mathbb{N}\}) \\ & = & \sup(\{F\ (F^i\ \bot) | i \in \mathbb{N}\}) \\ & = & \sup(\{F^{i+1}\ \bot | i \in \mathbb{N}\}) \\ & = & \sup(\{F^i\ \bot | i \in \mathbb{N}\}) \\ & = & x \end{array}$$

Veamos que es el menor de ellos. Sea y punto fijo de F, es decir $F\ y = y$. Veamos que $x \leq y$. Claramente $\bot \leq y$ por ser elemento mínimo. Como F es monótona, se obtiene $F\ \bot \leq F\ y = y$. Iterando, obtenemos $F^i\ \bot \leq y$ para todo i. Es decir, y es cota superior de la cadena $\{F^i\ \bot | i \in \mathbb{N}\}$. Como el supremo es la menor de esas cotas,

$$\begin{array}{rcl} x & = & \sup(\{F^i\ \bot | i \in \mathbb{N}\}) \\ & \leq & y \end{array}$$

3. Lenguaje Imperativo Simple

Gramática

$$\begin{array}{rcl} \langle comm \rangle & ::= & \textbf{skip} \\ & & \textbf{fail} \\ & & \langle \mathsf{var} \rangle := \langle intexp \rangle \\ & & \langle comm \rangle\ ;\ \langle comm \rangle \\ & & \textbf{if}\ \langle boolexp \rangle\ \textbf{then}\ \langle comm \rangle\ \textbf{else}\ \langle comm \rangle \\ & & \textbf{newvar}\ \langle \mathsf{var} \rangle\ \textbf{in}\ := \langle intexp \rangle\ \textbf{in}\ \langle comm \rangle \\ & & \textbf{while}\ \langle boolexp \rangle\ \textbf{do}\ \textbf{do}\ \langle comm \rangle \\ & & \textbf{catchin}\ \langle comm \rangle\ \textbf{with}\ \langle comm \rangle \\ & & !\,\langle intexp \rangle \\ & & ?\langle \mathsf{var} \rangle \end{array}$$

<u>Semántica Denotacional</u> (con fallas)

- $(*)$: Se tranfiere el control a f si NO HAY **abort**
- $(+)$: Se tranfiere el control a f SOLO en caso de **abort**
- $(\dagger)$: Se tranfiere el control a f SIEMPRE

$$
\begin{aligned}
[\![\mathbf{skip}]\!]\sigma &= \sigma \\
[\![\mathbf{fail}]\!]\sigma &= \langle \mathbf{abort}, \sigma \rangle \\
[\![v := e]\!]\sigma &= [\sigma | v : [\![e]\!]\sigma] \\
[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma &= \begin{cases} [\![c_0]\!]\sigma & \text{si } [\![b]\!]\sigma \\ [\![c_1]\!]\sigma & c.c \end{cases} \\
[\![c_0; c_1]\!]\sigma &= [\![c_1]\!]_*([\![c_0]\!]\sigma) \\
[\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]\sigma &= (\lambda\sigma' \in \Sigma.[\sigma'|v : \sigma v])_\dagger([\![c]\!][\sigma|v : [\![e]\!]\sigma]) \\
[\![\mathbf{catchin}\ c_0\ \mathbf{with}\ c_1]\!]\sigma &= [\![c_1]\!]_+([\![c_0]\!]\sigma) \\
[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma &= \bigsqcup_{i=0}^{\infty} F^i \bot \\
F\ \omega\ \sigma &= \begin{cases} \omega_*([\![c]\!]\sigma) & \text{si } [\![b]\!]\sigma \\ \sigma & \text{c.c} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\Sigma' &= \Sigma \cup \{\mathbf{abort}\} \times \Sigma \\
[\![\_]\!] &\in \langle comm \rangle \to \Sigma \to \Sigma'_\bot \\
f_*, f_+, f_\dagger &\in \Sigma'_\bot \to \Sigma'_\bot \\
f_* x &= \begin{cases} f\sigma & \text{si } x = \sigma \in \Sigma \\ x & \text{c.c} \end{cases} \\
f_+ x &= \begin{cases} f\sigma & \text{si } x = \langle \mathbf{abort}, \sigma \rangle \in \{\mathbf{abort}\} \times \Sigma \\ x & \text{c.c} \end{cases} \\
f_\dagger x &= \begin{cases} \langle \mathbf{abort}, f\sigma \rangle & x = \langle \mathbf{abort}, \sigma \rangle \\ fx & x \in \Sigma \\ \bot & x = \bot \end{cases}
\end{aligned}
$$

Variables libres y asignables

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

Teorema de Coincidencia (TC)

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

1. $\forall \omega \in FV(c).\ \sigma\omega = \sigma'\omega$ entonces
   - $[\![c]\!]\sigma = \bot = [\![c]\!]\sigma'$ o
   - $[\![c]\!]\sigma \neq \bot \neq [\![c]\!]\sigma'$ y $[\![c]\!]\sigma\omega = [\![c]\!]\sigma'\omega$
2. Si $[\![c]\!]\sigma \neq \bot$, entonces $\forall \omega \notin FA(c).\ [\![c]\!]\sigma\omega = \sigma\omega$

Teorema de Renombre (TR)

No importa el nombre de las variables utilizadas en las declaraciones de variables locales, es decir, las ligadas.

$u \notin FV(c) - \{v\} \Rightarrow [\![\mathbf{newvar}\ u := e\ \mathbf{in}\ c/v \to u]\!] = [\![\mathbf{newvar}\ u := e\ \mathbf{in}\ c]\!]$

Teorema de Sustitución (TS)

Si δ es inyectiva sobre $FV(c)$ y $\forall \omega \in FV(c).\ \sigma(\delta\omega) = \sigma'\omega$ entonces:

- $[\![c/\delta]\!]\sigma = \bot = [\![c]\!]\sigma'$ o
- $[\![c/\delta]\!]\sigma \neq \bot \neq [\![c]\!]\sigma'$ y $[\![c/\delta]\!]\sigma(\delta\omega) = [\![c]\!]\sigma'\omega$

Lema de Sustitución (LS)

Sea $FV(c) \subseteq V \subseteq \langle var \rangle$ tal que δ es inyectiva sobre V y $\forall \omega \in V.\ \sigma(\delta\omega) = \sigma'\omega$ entonces:

- $[\![c/\delta]\!]\sigma = \bot = [\![c]\!]\sigma'$ o
- $[\![c/\delta]\!]\sigma \neq \bot \neq [\![c]\!]\sigma'$ y $[\![c/\delta]\!]\sigma(\delta\omega) = [\![c]\!]\sigma'\omega$

<u>Semántica Denotacional</u> (Completa)

$$
\begin{array}{rcl}
\Omega & \approx & (\Sigma' + \mathbb{Z} \times \Omega + \mathbb{Z} \to \Omega)_\perp \\
\iota_{term} & = & \psi \ . \ \iota_\perp \ . \ \iota_0 \ . \ \iota_{norm} \in \Sigma \to \Omega \\
\iota_{abort} & = & \psi \ . \ \iota_\perp \ . \ \iota_0 \ . \ \iota_{abnorm} \in \Sigma \to \Omega \\
\iota_{out} & = & \psi \ . \ \iota_\perp \ . \ \iota_1 \in \mathbb{Z} \times \Sigma \to \Omega \\
\iota_{in} & = & \psi \ . \ \iota_\perp \ . \ \iota_2 \in (\mathbb{Z} \to \Sigma) \to \Omega \\
\perp_\Omega & = & \psi(\perp) \in \Omega
\end{array}
$$

$$f_*, f_+, f_\dagger \in \Omega \to \Omega$$

$$
\begin{array}{rcl}
[\![\mathbf{skip}]\!]\sigma & = & \iota_{term}\sigma \\
[\![\mathbf{fail}]\!]\sigma & = & \iota_{abort}\sigma \\
[\![v := e]\!]\sigma & = & \iota_{term}[\sigma|v : [\![e]\!]\sigma] \\
[\![!e]\!]\sigma & = & \iota_{out}([\![e]\!]\sigma, \iota_{term}\sigma) \\
[\![?v]\!]\sigma & = & \iota_{in}(\lambda n \in \mathbb{Z}.\iota_{term}[\sigma|v : n]) \\
[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma & = & \begin{cases} [\![c_0]\!]\sigma & \text{si } [\![b]\!]\sigma \\ [\![c_1]\!]\sigma & c.c \end{cases} \\
[\![c_0; c_1]\!]\sigma & = & [\![c_1]\!]_*([\![c_0]\!]\sigma) \\
[\![\mathbf{catchin}\ c_0\ \mathbf{with}\ c_1]\!]\sigma & = & [\![c_1]\!]_+([\![c_0]\!]\sigma) \\
[\![\mathbf{newvar}\ v := e\ \mathbf{in}\ c]\!]\sigma & = & (\lambda\sigma' \in \Sigma.[\sigma'|v : \sigma v])_\dagger([\![c]\!][\sigma|v : [\![e]\!]\sigma]) \\
[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma & = & \bigsqcup_{i=0}^{\infty} F^i \perp \\
F\ \omega\ \sigma & = & \begin{cases} \omega_*([\![c]\!]\sigma) & \text{si } [\![b]\!]\sigma \\ \iota_{term}\sigma & \text{c.c} \end{cases}
\end{array}
$$

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

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

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

<u>Semántica Operacional</u>

$$\frac{}{\langle \mathbf{skip}, \sigma \rangle \to \sigma}$$

$$\frac{}{\langle v := e, \sigma \rangle \to [\sigma|v : [\![e]\!]\sigma]}$$

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

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

$$\frac{([\![e]\!]\sigma = V)}{\langle \mathbf{if}\ e\ \mathbf{then}\ c\ \mathbf{else}\ c', \sigma \rangle \to \langle c, \sigma \rangle}$$

$$\frac{([\![e]\!]\sigma = F)}{\langle \mathbf{if}\ e\ \mathbf{then}\ c\ \mathbf{else}\ c', \sigma \rangle \to \langle c', \sigma \rangle}$$

$$\frac{([\![e]\!]\sigma = F)}{\langle \mathbf{while}\ e\ \mathbf{do}\ c, \sigma \rangle \to \sigma}$$

$$\frac{([\![e]\!]\sigma = V)}{\langle \mathbf{while}\ e\ \mathbf{do}\ c, \sigma \rangle \to \langle c; \mathbf{while}\ e\ \mathbf{do}\ c, \sigma \rangle}$$

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

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

Definición: Por *ejecución* entendemos una secuencia $c_0 \to c_1 \to ...$ maximal, esto es, que no puede prolongarse más de lo que está. Dicha ejecución es infinita o termina en una configuración terminal σ. Si la ejecución es infinita, decimos que diverge y escribimos $c\uparrow$.

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

Con fallas

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

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

$$\frac{\langle c_0, \sigma\rangle \to \sigma'}{\langle \mathbf{catchin}\ c_0\ \mathbf{with}\ c_1, \sigma\rangle \to \sigma'} \qquad \frac{\langle c, \sigma\rangle \to \langle \mathbf{abort}, \sigma'\rangle}{\langle \mathbf{newvar}\ v := e\ \mathbf{in}\ c, \sigma\rangle \to \langle \mathbf{abort}, [\sigma' | v : \sigma v]\rangle}$$

$$\{\!\{c\}\!\}_\sigma = \begin{cases} \perp & \text{si } \langle c, \sigma\rangle \uparrow \ \ (\uparrow\text{: ejecución infinita}) \\ \sigma' & \text{si existe } \sigma' \text{ tal que } \langle c, \sigma\rangle \to^* \sigma' \\ \langle \mathbf{abort}, \sigma'\rangle & \text{si existe } \sigma' \text{ tal que } \langle c, \sigma\rangle \to^* \langle \mathbf{abort}, \sigma\rangle' \end{cases}$$

PROPIEDADES:

Lema 1:

1. Si $\langle c_0, \sigma\rangle \to^* \sigma'$ entonces $\langle c_0; c_1, \sigma\rangle \to^* \langle c_1, \sigma'\rangle$
2. Si $\langle c, [\sigma | v : [\![e]\!]\sigma]\rangle \to^* \sigma'$ entonces $\langle \mathbf{newvar}\ v := e\ \mathbf{in}\ c, \sigma\rangle \to^* [\sigma' \ | v : \sigma v]$
3. Si $\langle c, [\sigma | v : [\![e]\!]\sigma]\rangle \to^* \langle c', \sigma'\rangle$ entonces $\langle \mathbf{newvar}\ v := e\ \mathbf{in}\ c, \sigma\rangle \to^* \langle \mathbf{newvar}\ v := \sigma' v\ \mathbf{in}\ c', [\sigma' | \ v : \sigma v]\rangle$

Prueba:

1. Supongamos $G_0 = \langle c_0, \sigma\rangle \to^* \sigma'$, y que la ejecución $G_0 \to^* \sigma'$ tiene n pasos, es decir:

   $$G_0 \to G_1 \to ...G_n = \sigma'$$

   Probaremos por inducción en n

   Caso base: $n = 1$ Surge directamente de la regla: $\dfrac{\langle c_0, \sigma\rangle \to \sigma'}{\langle c_0; c_1, \sigma\rangle \to \langle c_1, \sigma'\rangle}$

   Caso inductivo: Tomamos como HI que a) vale para ejecuciones $\leq n$.

   Supongamos que tenemos una ejecución de n pasos, $G_0 \to G_1 \to ...G_n = \sigma'$.

   Sea $G_1 \to^* \sigma'$, de $n-1$ pasos, entonces por HI tenemos que $\langle c_0^1; c_1, \sigma^1\rangle \to^* \langle c_1, \sigma'\rangle$

   Luego, utilizando la segunda regla de ; obtenemos: $\dfrac{\langle c_0, \sigma\rangle \to \langle c_0^1, \sigma^1\rangle}{\langle c_0; c_1, \sigma\rangle \to \langle c_0^1; c_1, \sigma^1\rangle}$

   Finalmente, $\langle c_0; c_1, \sigma\rangle \to^* \langle c_1, \sigma'\rangle$

2.

3.

Lema 2:

1. $\langle c, \sigma \rangle \to \sigma' \Rightarrow [\![c]\!]\sigma = \sigma'$
2. $\langle c, \sigma \rangle \to \langle c', \sigma' \rangle \Rightarrow [\![c]\!]\sigma = [\![c']\!]\sigma'$

Prueba:

Lema 3:

$[\![c]\!]\sigma = \sigma' \Rightarrow \langle c, \sigma \rangle \to^* \sigma'$

Prueba:

Teorema:

Para todo comando c se tiene $\{\!\{c\}\!\} = [\![c]\!]$

Prueba:

Teoremas de Coincidencia, Renombre y Sustitución para la semántica operacional

Todos estos teoremas son válidos para la semántica operacional también. Se prueban sencillamente utilizando el teorema anterior, que dice que para todo comando c vale $\{\!\{c\}\!\} = [\![c]\!]$.

4. Cálculo Lambda

$$
\begin{array}{lcll}
\langle\mathsf{expr}\rangle & ::= & & \text{término lambda, o expresión} \\
& | & \langle\mathsf{var}\rangle & \text{variable} \\
& | & \langle\mathsf{expr}\rangle\langle\mathsf{expr}\rangle & \text{aplicación, el primero es el operador y el segundo el operando} \\
& | & \lambda\langle\mathsf{var}\rangle.\langle\mathsf{expr}\rangle & \text{abstracción o expresión lambda}
\end{array}
$$

$$
\begin{aligned}
FV(v) &= \{v\} \\
FV(e_0e_1) &= FV(e_0) \cup FV(e_1) \\
FV(\lambda v.e) &= FV(e) - \{v\}
\end{aligned}
$$

También se define sustitución:

$$
\begin{array}{rcl}
\Delta & = & \langle\mathsf{var}\rangle \to \langle\mathsf{expr}\rangle \\
\_/\_ & \in & \langle\mathsf{expr}\rangle \times \Delta \to \langle\mathsf{expr}\rangle \\
v/\delta & = & \delta v \\
(e_0e_1)/\delta & = & (e_0/\delta)(e_1/\delta) \\
(\lambda v.e)/\delta & = & \lambda v'.(e/[\delta|v : v']) \\
& & \text{donde } v' \notin \displaystyle\bigcup_{w\in FV(e)-\{v\}} FV(\delta w)
\end{array}
$$

Propiedad 1.

1. *si para todo* $w \in FV(e)$, $\delta w = \delta' w$ *entonces* $(e/\delta) = (e/\delta')$
2. *sea* i *la sustitución identidad, entonces* $e/i = e$.
3. $FV(e/\delta) = \bigcup_{w\in FV(e)} FV(\delta w)$

Definiciones:

- Expresión cerrada: sin variables libres
- Forma canónica: abstracciones $(\lambda x.e)$
- *Redex*: expresión de la forma $(\lambda v.e)e'$.
- Expresion lambda cerrada: $\lambda v.e$ tal que $FV(e) = \{v\}$
- Formal normal: expresión sin rédices. No necesariamente son abstracciones, por ejemplo, $(\lambda x.y)(\lambda z.z)$ β-contrae a y es es formal normal pero no canónica.

Renombre (α) La operación de cambiar una ocurrencia de la expresión lambda $\lambda v.e$ por $\lambda v'.(e/v \to v')$ donde $v' \notin FV(e) - \{v\}$ se llama renombre o cambio de variable ligada.

Contracción (β) Es la aplicación de una función $(\lambda v.e)$ a su argumento e'. Debe calcularse reemplazando las ocurrencias libres de v en e por e', es decir $(e/v \to e')$.

Teorema 1. *Church-Rosser. Si* $e \to^* e_0$ *y* $e \to^* e_1$, *entonces existe* e' *tal que* $e_0 \to^* e'$ *y* $e_1 \to^* e'$.

Corolario 1. *Salvo renombre, toda expresión tiene a lo sumo una forma normal.*

NO TODA EXPRESIÓN TIENE FORMA NORMAL (contraejemplos).

- Sea $\Delta = (\lambda x.xx)$, $\Delta\Delta$ no es forma normal
- Sea $\Delta' = (\lambda x.xxy)$, $\Delta'\Delta'$ no es forma normal

Propiedad 2. *Una aplicación cerrada no puede ser forma normal.*

Demostración. Sea e una aplicación cerrada, es decir $e = e_0 e_1 \ldots$. Si e_0 fuera una variable, e no sería cerrada, por ende e_0 es una abstracción. Por lo tanto e contiene el redex $e_0 e_1$ y por esto no es normal. $\square$

Corolario 2. *Si una expresión cerrada, es forma normal entonces es forma canónica.*

Al revés no vale, contrajemplo $\lambda x.(\lambda y.y)x$.

Diferencias entre $\rightarrow$ y $\Rightarrow$

1. solo se evaluan expresiones cerradas
2. es determinística
3. no busca formas normales sino formas canónicas

4.1. Evaluación

En este tipo de semántica operacional se describen la relación entre los términos y sus valores, que también son términos, formas canónicas. Llamamos $\Rightarrow$ a esta relación. Cuando decimos que $e \Rightarrow z$ se cumple, estamos diciendo que existe un árbol de derivación que prueba $e \Rightarrow z$.

Puede ocurrir que la evaluación eager no termine mientras que la normal si, por ejemplo: $(\lambda x.\lambda y.y)(\Delta\Delta) \Rightarrow_N \lambda y.y$, mientras que en eager diverge.

Regla η Un η-redex es una expresión de la forma $\lambda v.e\, v$ donde $v \notin FV(e)$. Gracias a la regla β, uno obtiene que $(\lambda v.e\, v)e'$ contrae a $e\, e'$ para toda expresión e'. Si uno asume que toda expresión lambda denota una función, $\lambda v.e\, v$ y e parecen comportarse extensionalmente igual: cuando se las aplica a e', ambas dan ee'. Esto motiva la regla η:

$$\frac{}{(\lambda v.ev) \rightarrow e} \text{ si } v \notin FV(e)$$

4.2. Semántica denotacional

Sea $D_\infty \cong [D_\infty \rightarrow D_\infty]$, donde $D_\infty \rightarrow D_\infty$ se refiere al espacio de funciones continuas donde todas tiene punto fijo, y sean:

$$\begin{aligned} \phi &\in D_\infty \rightarrow [D_\infty \rightarrow D_\infty] \\ \psi &\in [D_\infty \rightarrow D_\infty] \rightarrow D_\infty \end{aligned}$$

los isomorfismos tales que

$$\phi.\psi = id$$
$$\psi.\phi = id$$

$$\begin{aligned} \text{Ambiente: } Env &= \langle \mathsf{var} \rangle \rightarrow D_\infty \\ \eta &\in Env \\ [\![\_]\!] &\in \langle \mathsf{expr} \rangle \rightarrow Env \rightarrow D_\infty \end{aligned}$$

$$\begin{aligned} [\![v]\!]\eta &= \eta\, v \\ [\![e_0\, e_1]\!]\eta &= \phi([\![e_0]\!]\eta)([\![e_1]\!]\eta) \\ [\![\lambda x.e]\!]\eta &= \psi(\lambda d \in D_\infty.[\![e]\!][\eta|v : d]) \end{aligned}$$

Teorema 2. *Coincidencia Si $\eta\, w = \eta'\, w$ para todo $w \in FV(e)$, entonces $[\![e]\!]\eta = [\![e]\!]\eta'$.*

Teorema 3. *Sustitución Si* $[\![\delta\, w]\!]\eta = \eta' w$ *para todo* $w \in FV(e)$*, entonces* $[\![e/\delta]\!]\eta = [\![e]\!]\eta'$.

Teorema 4. *Sustitución Finita* $[\![e/v_1 \to e_1, \ldots, v_n \to e_n]\!]\eta = [\![e]\!][\eta|v_1 : [\![e_1]\!]\eta| \ldots |v_n : [\![e_n]\!]\eta]$.

Teorema 5. *Renombre Si* $v' \notin FV(e) - \{v\}$*, entonces* $[\![\lambda v'.(e/v \to v')]\!] = [\![\lambda v.e]\!]$.

Propiedad 3. *(correctitud de la regla* β*):* $[\![(\lambda v.e)\, e']\!] = [\![e/v \to e']\!]$.

Propiedad 4. *(correctitud de la regla* η*): Si* $v \notin FV(e)$*, entonces* $[\![\lambda v.ev]\!] = [\![e]\!]$.

Corolario 3. *Si* $e \to^* e'$*, entonces* $[\![e]\!] = [\![e']\!]$

Asumiremos que $[\![\Delta\Delta]\!]\eta = \perp$.

4.2.1. Normal

$$
\begin{array}{rcl}
D & = & V_{\perp}, \text{ donde } V \cong [D \to D] \\
\\
\phi & \in & V \to [D \to D] \\
\psi & \in & [D \to D] \to V \\
\\
\lambda d & \in & D.\perp \text{ bottom de } D \to D \\
\psi(\lambda d & \in & D.\perp) \text{ bottom de V pero no de D} \\
\\
\phi_{\perp\!\!\!\perp} & \in & D \to [D \to D] \\
\iota_{\perp}.\psi & \in & [D \to D] \to D \\
\\
Env & = & \langle \mathsf{var} \rangle \to D \\
[\![\_]\!] & \in & \langle \mathsf{expr} \rangle \to Env \to D \\
\\
[\![v]\!]\eta & = & \eta v \\
[\![e_0\, e_1]\!]\eta & = & \phi_{\perp\!\!\!\perp}([\![e_0]\!]\eta)([\![e_1]\!]\eta) \\
[\![\lambda x.e]\!]\eta & = & (\iota_{\perp}.\psi)(\lambda d \in D.[\![e]\!][\eta|v : d])
\end{array}
$$

Los teoremas antes vistos junto con la regla β siguen valiendo pero no la regla η, pues $[\![\lambda y.\Delta\Delta y]\!] = \psi(\lambda d \in D.\perp)$ mientras que $[\![\Delta\Delta]\!] = \perp$.

4.2.2. Eager

$$
\begin{aligned}
D &= V_\perp,\ \text{donde } V \cong [V \to D] \\
\\
\phi &\in V \to [V \to D] \\
\psi &\in [V \to D] \to V \\
\\
\phi_{\perp\!\!\perp} &\in D \to [V \to D] \\
\iota_\perp.\psi &\in [V \to D] \to D \\
\\
Env &= \langle \mathsf{var} \rangle \to V \\
[\![\_]\!] &\in \langle \mathsf{expr} \rangle \to Env \to D \\
\\
[\![v]\!]\eta &= \iota_\perp(\eta v) \\
[\![e_0\, e_1]\!]\eta &= (\phi_{\perp\!\!\perp}([\![e_0]\!]\eta))_{\perp\!\!\perp}\,([\![e_1]\!]\eta) \\
[\![\lambda x.e]\!]\eta &= (\iota_\perp.\psi)(\lambda z \in V.[\![e]\!][\eta|x:z])
\end{aligned}
$$

Los teoremas antes vistos (salvo sustitución) siguen valiendo pero no las reglas β, ya que $[\![(\lambda y\lambda x.x)(\Delta\Delta)]\!]$ diverge en eager y no da $[\![\lambda x.x]\!]$ como esperaría la regla y η, ya que $[\![\lambda y.\Delta\Delta y]\!] = \psi(\lambda d \in D.\perp)$ mientras que $[\![\Delta\Delta]\!] = \perp$.

5. Lenguajes Aplicativos

5.1. Gramáticas

$\langle \mathsf{expr} \rangle$	$::=$		término lambda, o expresión
	$\mid$	$\langle \mathsf{var} \rangle$	variable
	$\mid$	$\langle \mathsf{expr} \rangle \langle \mathsf{expr} \rangle$	aplicación, el primero es el operador y
	$\mid$	$\lambda \langle \mathsf{var} \rangle . \langle \mathsf{expr} \rangle$	abstracción o expresión lambda
	$\mid$	$\langle natconst \rangle \mid \langle boolconst \rangle$	
	$\mid$	$-\langle \mathsf{expr} \rangle \mid \langle \mathsf{expr} \rangle + \langle \mathsf{expr} \rangle \mid \langle \mathsf{expr} \rangle * \langle \mathsf{expr} \rangle \mid ...$	operadores aritméticos
	$\mid$	$\langle \mathsf{expr} \rangle \geq \langle \mathsf{expr} \rangle \mid \langle \mathsf{expr} \rangle \leq \langle \mathsf{expr} \rangle \mid \langle \mathsf{expr} \rangle < \langle \mathsf{expr} \rangle \mid ...$	operadores relacionales
	$\mid$	$\langle \mathsf{expr} \rangle \wedge \langle \mathsf{expr} \rangle \mid \langle \mathsf{expr} \rangle \vee \langle \mathsf{expr} \rangle \mid \neg \langle \mathsf{expr} \rangle$	operadores lógicos
	$\mid$	**if** $\langle \mathsf{expr} \rangle$ **then** $\langle \mathsf{expr} \rangle$ **else** $\langle \mathsf{expr} \rangle$	
	$\mid$	$\langle \langle \mathsf{expr} \rangle, ... \langle \mathsf{expr} \rangle \rangle$	
	$\mid$	$\langle \mathsf{expr} \rangle . \langle natconst \rangle$	
	$\mid$	**letrec** $\langle \mathsf{var} \rangle \equiv \lambda \langle \mathsf{var} \rangle . \langle \mathsf{expr} \rangle$ **in** $\langle \mathsf{expr} \rangle$	(Eval. Eager)
	$\mid$	**rec** $\langle \mathsf{expr} \rangle$	(Eval. Normal)
	$\mid$	**error** $\mid$ **typeerror**	
$\langle natconst \rangle$	$::=$	$0 \mid 1 \mid 2 \mid ....$	
$\langle boolconst \rangle$	$::=$	**true** $\mid$ **false**	

Formas canónicas

$$
\begin{aligned}
\langle cnf \rangle &::= \langle intcnf \rangle \mid \langle boolcnf \rangle \mid \langle funcnf \rangle \mid \langle tuplecnf \rangle \\
\langle intcnf \rangle &::= ... \mid -2 \mid -1 \mid 0 \mid 1 \mid 2 \mid ... \\
\langle boolcnf \rangle &::= \langle boolconst \rangle \\
\langle funcnf \rangle &::= \lambda \langle \mathsf{var} \rangle . \langle \mathsf{expr} \rangle \\
\langle tuplecnf \rangle &::= \langle \langle cnf \rangle, ..., \langle cnf \rangle \rangle
\end{aligned}
$$

5.2. Evaluación

5.2.1. Compartido entre Normal y Eager

Para todas las formas canónicas vale: $\overline{z \Rightarrow z}$

- **Aritméticos y relacionales:** $\oplus \in \{+,-,*,=,\neq,\leq,\geq,...\}$, $\oslash \in \{/, mod\}$
  - $\dfrac{e \;\Rightarrow\; \lfloor i \rfloor}{-e \;\Rightarrow\; \lfloor -i \rfloor}$
  - $\dfrac{e \;\Rightarrow\; \lfloor i \rfloor \qquad e' \;\Rightarrow\; \lfloor i' \rfloor}{e \oplus e' \;\Rightarrow\; \lfloor i \oplus i' \rfloor}$
  - $\dfrac{e \;\Rightarrow\; \lfloor i \rfloor \qquad e' \;\Rightarrow\; \lfloor i' \rfloor}{e \oslash e' \;\Rightarrow\; \lfloor i \oslash i' \rfloor}$

    $i' \neq 0$
- **Booleanos:**
  - $\dfrac{e \;\Rightarrow\; \lfloor b \rfloor}{\neg e \;\Rightarrow\; \lfloor \neg b \rfloor}$
  - $\dfrac{e \;\Rightarrow\; \textbf{true} \qquad e_0 \;\Rightarrow\; z}{\textbf{if } e \textbf{ then } e_0 \textbf{ else } e_1 \;\Rightarrow\; z}$
  - $\dfrac{e \;\Rightarrow\; \textbf{false} \qquad e_1 \;\Rightarrow\; z}{\textbf{if } e \textbf{ then } e_0 \textbf{ else } e_1 \;\Rightarrow\; z}$
- **Definiciones locales y patrones: let** $p_1 \equiv e_1, \ldots, p_n \equiv e_n$ **in** $e = (\lambda p_1 \ldots \lambda p_n . e) e_1 \ldots e_n$

$$\frac{e' \;\Rightarrow\; z' \qquad e/v \to z' \;\Rightarrow\; z}{\textbf{let } v \equiv e' \textbf{ in } e \;\Rightarrow\; z}$$

5.2.2. Normal

- **Funciones:**

$$\frac{e \Rightarrow \lambda v.e'' \qquad (e''/v \to e') \Rightarrow z}{ee' \Rightarrow z} \text{ (aplicación)}$$

- **Booleanos:**
  - $e \wedge e' =$ **if** e **then** e' **else** $false$
  - $e \vee e' =$ **if** e **then** $true$ **else** e'
  - $e \Rightarrow e' =$ **if** e **then** e' **else** $true$
- **Tuplas:**

$$\frac{}{\langle e_1, ..., e_n \rangle \Rightarrow \langle e_1, ..., e_n \rangle} \qquad \frac{e \Rightarrow \langle e_1, ..., e_n \rangle \qquad e_k \Rightarrow z}{e.\lfloor k \rfloor \Rightarrow z} \quad \text{para } k \leq n$$

- **Recursión:**

$$\frac{e \; (\textbf{rec } e) \Rightarrow z}{\textbf{rec } e \Rightarrow z}$$

5.2.3. Eager

- **Funciones:**

$$\frac{e \Rightarrow \lambda v.e'' \qquad e' \Rightarrow z' \qquad (e''/v \to z') \Rightarrow z}{ee' \Rightarrow z} \text{ (aplicación)}$$

- **Booleanos:** $\oslash \in \{\wedge, \vee, \Rightarrow, ...\}$

$$\frac{e \;\Rightarrow\; \lfloor b \rfloor \qquad e' \;\Rightarrow\; \lfloor b' \rfloor}{e \oslash e' \;\Rightarrow\; \lfloor b \oslash b' \rfloor}$$

- **Tuplas:**

$$\frac{e_1 \Rightarrow z_1 \quad ... \quad e_n \Rightarrow z_n}{\langle e_1, ..., e_n\rangle \;\Rightarrow\; \langle z_1, ..., z_n\rangle} \qquad \frac{e \Rightarrow \langle z_1, ..., z_n\rangle}{e.\lfloor k\rfloor \Rightarrow\; z_k} \qquad \text{para } k \leq n$$

- **Recursión:**

$$\frac{e/(f \mapsto \lambda v.\mathbf{letrec}\ f \equiv \lambda v.e' \ \mathbf{in}\ e' \ ) \Rightarrow z}{\mathbf{letrec}\ f \equiv \lambda v.e' \ \mathbf{in}\ e \;\Rightarrow\; z} \quad f \neq v$$

5.3. Semántica denotacional

5.3.1. Compartido entre Normal y Eager

V: es el conjunto de valores, de la misma forma que a las formas canónicas las llamamos también valores en su momento.
D: es el conjunto de resultados, que incluyen valores y otras cosas, en este caso, sólo se agrega bottom.

$$\begin{array}{rcl} D & = & (V + \{\mathbf{error}, \mathbf{typeerror}\})_\perp \\ \iota_{norm} & = & \iota_\perp.\iota_0 \in V \to D \\ err & = & \iota_\perp.(\iota_1\ error) \in D \\ tyerr & = & \iota_\perp.(\iota_1\ typeerror) \in D \end{array} \qquad \begin{array}{rcl} V & \simeq & V_{int} + V_{bool} + V_{fun} + V_{tuple} \\ \phi & \in & V \to\ V_{int} + V_{bool} + V_{fun} + V_{tuple} \\ \psi & \in & V_{int} + V_{bool} + V_{fun} + V_{tuple} \to V \end{array}$$

Donde: $V_{int} = \mathbb{Z}, V_{bool} = \mathbb{B}$ y V_{fun}, V_{tuple} se definen para cada semántica en particular.

Si $f \in V \to D$ entonces $f_* \in D \to D$ se define: Si $\ell \in \{int, bool, fun, tuple\}, f \in V_\ell \to D$ entonces f_ℓ se define:

$$\begin{array}{rcl} f_*(\iota_{norm}\ z) & = & f\ z \\ f_*(err) & = & err \\ f_*(tyerr) & = & tyerr \\ f_*(\perp) & = & \perp \end{array} \qquad \begin{array}{rcl} f_\ell(\iota_\ell\ z) & = & f\ z \\ f_\ell(\iota_{\ell'}\ z) & = & tyerr,\ \text{si } \ell \neq \ell' \end{array}$$

$$\begin{array}{rcl} \iota_{int} & = & \psi.\iota_0 \in V_{int} \to V \\ \iota_{bool} & = & \psi.\iota_1 \in V_{bool} \to V \\ \iota_{fun} & = & \psi.\iota_2 \in V_{fun} \to V \\ \iota_{tuple} & = & \psi.\iota_3 \in V_{tuple} \to V \end{array}$$

$$\begin{array}{rcl} [\![0]\!]\eta & = & \iota_{norm}(\iota_{int} 0) \\ [\![\mathbf{true}]\!]\eta & = & \iota_{norm}(\iota_{bool} T) \\ \\ [\![-e]\!]\eta & = & (\lambda i \in V_{int}.\iota_{norm}(\iota_{int} - i))_{int*}([\![e]\!]\eta) \\ [\![\neg e]\!]\eta & = & (\lambda b \in V_{bool}.\iota_{norm}(\iota_{bool} \neg b))_{bool*}([\![e]\!]\eta) \\ \\ [\![e_0 + e_1]\!]\eta & = & (\lambda i \in V_{int}.\ (\lambda j \in V_{int}.\iota_{norm}(\iota_{int} i + j))_{int*}([\![e_1]\!]\eta))_{int*}([\![e_0]\!]\eta) \\ [\![e_0/e_1']\!]\eta & = & (\lambda i \in V_{int}.\ (\lambda j \in V_{int}. \left\{ \begin{array}{ll} err & j = 0 \\ \iota_{norm}(\iota_{int} i/j) & c.c \end{array} \right)_{int*} ([\![e_1]\!]\eta))_{int*}([\![e_0]\!]\eta) \\ [\![\mathbf{if}\ e\ \mathbf{then}\ e_0\ \mathbf{else}\ e_1]\!]\eta & = & (\lambda b \in V_{bool}. \left\{ \begin{array}{ll} [\![e_0]\!] & sib \\ [\![e_1]\!] & c.c \end{array} \right)_{bool*} ([\![e]\!]\eta) \end{array}$$

$$[\![\mathbf{let}\ v \equiv e' \ \mathbf{in}\ e]\!]\eta \ = \ (\lambda z \in V.[\![e]\!][\eta|v:z])_*([\![e']\!]\eta)$$

$$\begin{aligned}[\![error]\!]\eta &= err\\ [\![typeerror]\!]\eta &= tyerr\end{aligned}$$

5.3.2. Normal

$V_{fun} = D \to D \qquad V_{tuple} = D^*$

$$\begin{aligned}Env &= \langle\mathsf{var}\rangle \to D\\ [\![\_]\!] &\in \langle\mathsf{expr}\rangle \to Env \to D\end{aligned}$$

$$\begin{aligned}[\![v]\!]\eta &= \eta v\\ [\![e_0\, e_1]\!]\eta &= (\lambda f \in V_{fun}.f([\![e_1]\!]\eta))_{fun*}([\![e_0]\!]\eta)\\ [\![\lambda x.e]\!]\eta &= \iota_{norm}(\iota_{fun}(\lambda d \in D.[\![e]\!][\eta|x:d]))\end{aligned}$$

$$\begin{aligned}[\![\langle e_1,...,e_n\rangle]\!]\eta &= \iota_{norm}(\iota_{tuple}\,\langle[\![e_1]\!]\eta,...,[\![e_n]\!]\eta\rangle)\\ [\![e.[k]]\!]\eta &= \left(\lambda t \in V_{tuple}.\ \begin{cases} t.k & si\ k \leq \#t\\ tyerr & c.c\end{cases}\right)_{tuple*}([\![e]\!]\eta)\\ [\![\mathbf{rec}\ e]\!]\eta &= (\lambda f \in V_{fun}.\ Y f)_{fun*}([\![e]\!]\eta)\\ \text{donde}\ & \quad Y f = \sqcup_{i\geq 0} f^i \bot\end{aligned}$$

5.3.3. Eager

$V_{fun} = V \to D \qquad V_{tuple} = V^*$

$$\begin{aligned}Env &= \langle\mathsf{var}\rangle \to V\\ [\![\_]\!] &\in \langle\mathsf{expr}\rangle \to Env \to D\end{aligned}$$

$$\begin{aligned}[\![v]\!]\eta &= \iota_{norm}(\eta v)\\ [\![e_0\, e_1]\!]\eta &= (\lambda f \in V_{fun}.f_*([\![e_1]\!]\eta))_{fun*}([\![e_0]\!]\eta)\\ [\![\lambda x.e]\!]\eta &= \iota_{norm}(\iota_{fun}(\lambda z \in V.[\![e]\!][\eta|x:z]))\end{aligned}$$

$$\begin{aligned}[\![\langle e_1,...,e_n\rangle]\!]\eta &= (\lambda z_1 \in V....(\lambda z_n \in V.\iota_{norm}(\iota_{tuple}\,\langle z_1,...,z_n\rangle))_*([\![e_n]\!]\eta)...)_*([\![e_1]\!]\eta)\\ [\![e.[k]]\!]\eta &= \left(\lambda t \in V_{tuple}.\ \begin{cases} \iota_{norm} t.k & si\ k \leq \#t\\ tyerr & c.c\end{cases}\right)_{tuple*}([\![e]\!]\eta)\end{aligned}$$

$$\begin{aligned}[\![\mathbf{letrec}\ v \equiv \lambda u.e\ \mathbf{in}\ e'\ ]\!]\eta &= [\![e']\!][\eta|v:\iota_{fun} Y_{V_{fun}} F]\\ \text{donde}\ & \quad F\ f\ z = [\![e]\!][\eta|v:\iota_{fun} f|u:z]\\ & \quad Y_{V_{fun}} F = \sqcup_{i\geq 0} F^i \bot\end{aligned}$$

6. El lenguaje Iswin

Componente imperativa a un lenguaje aplicativo eager.

$\langle\mathsf{expr}\rangle$ ::=		
|	**ref** $\langle\mathsf{expr}\rangle$	genera una nueva referencia con el valor de e
|	**val** $\langle\mathsf{expr}\rangle$	devuelve el valor alojado en la referencia
|	$\langle\mathsf{expr}\rangle := \langle\mathsf{expr}\rangle$	modifica el estado, devolviendo el valor de la nueva expresión
|	$\langle\mathsf{expr}\rangle =_{ref} \langle\mathsf{expr}\rangle$	comprueba si las referencias son las mismas (comparación de punteros)

Definiciones:

- Rf: conjunto infinito de referencias
- $\Sigma = \bigcup_{F \subset_{fin} V_{ref}} F \rightarrow V$
- $new(\sigma) = new(dom(\sigma))$
- $new(\sigma) \in Rf$, devuelve una referencia nueva tal que $new(\sigma) \notin dom(\sigma)$
- V $= V_{int} + V_{bool} + V_{fun} + V_{tuple} + V_{ref}$
- D $= (\Sigma \times V + \{error, typeerror\})_{\perp}$
- $\iota_{norm} \langle \sigma, z \rangle \in D$, $\sigma \in \Sigma, z \in V$

$$\begin{array}{lcllcll}
V_{int} & = & \mathbf{Z} & \iota_{int} & \in & V_{int} & \rightarrow V \\
V_{bool} & = & \mathbf{B} & \iota_{bool} & \in & V_{bool} & \rightarrow V \\
V_{fun} & = & \Sigma \times V \rightarrow D & \iota_{fun} & \in & V_{fun} & \rightarrow V \\
V_{tuple} & = & V^{*} & \iota_{tuple} & \in & V_{tuple} & \rightarrow V \\
V_{ref} & = & Rf & \iota_{ref} & \in & V_{ref} & \rightarrow V
\end{array}$$

- Si $f \in \Sigma \times V \rightarrow D$, entonces $f_{*} \in \Sigma \times D \rightarrow D$ se define:

$$\begin{array}{llcl}
f_{*} & \iota_{norm} \langle \sigma, z \rangle & = & f \langle \sigma, z \rangle \\
f_{*} & err & = & err \\
f_{*} & tyerr & = & tyerr \\
f_{*} & \perp & = & \perp
\end{array}$$

- Si $f \in \Sigma \times V_{int} \rightarrow D$, entonces $f_{int} \in \Sigma \times V \rightarrow D$ se define:

$$\begin{array}{llcll}
f_{int} & \langle \sigma, \iota_{int} k \rangle & = & f \langle \sigma, k \rangle & \\
f_{int} & \langle \sigma, \iota_{\theta} z \rangle & = & tyerr & (\theta \neq int)
\end{array}$$

6.1. Semántica denotacional

$$\begin{array}{rcl}
[\![\mathbf{val}\ e]\!]\eta\sigma & = & (\lambda \langle \sigma', r \rangle .\ \in \Sigma \times V_{ref}. \left\{ \begin{array}{ll} \iota_{norm} \langle \sigma', \sigma' r \rangle & r \in dom(\sigma') \\ err & c.c \end{array} \right)_{ref*} ([\![e]\!]\eta\sigma) \\
[\![\mathbf{ref}\ e]\!]\eta\sigma & = & (\lambda \langle \sigma', r \rangle \in \Sigma \times V.\ \iota_{norm} \langle [\sigma' | r : z], \iota_{ref} r \rangle)_{*}([\![e]\!]\eta\sigma) \text{ donde } r = new(\sigma') \\
[\![e := e']\!]\eta\sigma & = & (\lambda \langle \sigma', r \rangle \in \Sigma \times V_{ref}.\ (\lambda \langle \sigma'', z \rangle \in \Sigma \times V.\ \iota_{norm} \langle [\sigma'' | r : z], z \rangle)_{*}([\![e']\!]\eta\sigma'))_{ref*}([\![e]\!]\eta\sigma) \\
[\![e =_{ref} e']\!]\eta\sigma & = & (\lambda \langle \sigma', r \rangle \in \Sigma \times V_{ref}.\ (\lambda \langle \sigma'', z \rangle \in \Sigma \times V_{ref}.\ \iota_{norm} \langle \sigma'', \iota_{bool} r = r' \rangle)_{ref*}([\![e']\!]\eta\sigma'))_{ref*}([\![e]\!]\eta\sigma)
\end{array}$$

$$\begin{array}{rcl}
[\![0]\!]\eta\sigma & = & \iota_{norm} \langle \sigma, \iota_{int} 0 \rangle \\
[\![\mathbf{true}]\!]\eta\sigma & = & \iota_{norm} \langle \sigma, \iota_{bool} T \rangle
\end{array}$$

$$\begin{array}{rcl}
[\![-e]\!]\eta\sigma & = & (\lambda \langle \sigma', i \rangle .\ \iota_{norm} \langle \sigma', \iota_{int} - i \rangle)_{int*}([\![e]\!]\eta\sigma) \\
[\![\neg e]\!]\eta\sigma & = & (\lambda \langle \sigma', b \rangle .\ \iota_{norm} \langle \sigma', \iota_{bool} \neg b \rangle)_{bool*}([\![e]\!]\eta\sigma) \\
[\![e + e']\!]\eta\sigma & = & (\lambda \langle \sigma', i \rangle \in \Sigma \times V_{int}.(\lambda \langle \sigma'', j \rangle \in \Sigma \times V_{int}.\ \iota_{norm} \langle \sigma'', \iota_{int} i + j \rangle)_{int*}([\![e']\!]\eta\sigma'))_{int*}([\![e]\!]\eta\sigma)
\end{array}$$

$$\begin{array}{rcl}
[\![v]\!]\eta\sigma & = & \iota_{norm}(\sigma, \eta v) \\
[\![ee']\!]\eta\sigma & = & (\lambda \langle \sigma', f \rangle \in \Sigma \times V_{fun}.\ f_{*}([\![e']\!]\eta\sigma'))_{fun*}([\![e]\!]\eta\sigma) \\
[\![\lambda v.\ e']\!]\eta\sigma & = & \iota_{norm} \langle \sigma, \iota_{fun}(\lambda \langle \sigma', z \rangle \in \Sigma \times V.\ [\![e]\!][\eta | v : z]\sigma') \rangle \\
[\![\mathbf{letrec}\ w & = & \lambda v.\ e\ \mathbf{in}\ e']\!]\eta\sigma \ = \ [\![e']\!][\eta | w : \iota_{fun} f]\sigma \\
\text{donde} & & f = \mathbf{Y}_{V_{fun}} F \\
& & F\ f\ \langle \sigma', z \rangle = [\![e]\!][\eta | w : \iota_{fun} f | v : z]\sigma'
\end{array}$$

6.2. Semántica operacional

$$\frac{\sigma, e \Rightarrow z, \sigma'}{\sigma, \mathbf{ref}\ e \Rightarrow r, [\sigma'|r : z]} \qquad r = new(\sigma')$$

$$\frac{\sigma, e \Rightarrow \lambda v.\ e_0, \sigma' \quad \sigma', e' \Rightarrow z', \sigma'' \quad \sigma'', (e_0/v \to z') \Rightarrow z, \sigma'''}{ee', \sigma \Rightarrow z, \sigma'''}$$

$$\frac{\sigma, e \Rightarrow r, \sigma'}{\sigma, \mathbf{val}\ e \Rightarrow \sigma' r, \sigma'} \qquad r \in dom(\sigma')$$

$$\frac{\sigma, e \Rightarrow r, \sigma' \quad \sigma', e' \Rightarrow z, \sigma''}{\sigma, e := e' \Rightarrow z, [\sigma''|r : z]}$$

$$\frac{\sigma, e \Rightarrow r, \sigma' \quad \sigma', e' \Rightarrow r', \sigma''}{\sigma, e =_{ref} e' \Rightarrow \lfloor r = r' \rfloor, \sigma''}$$

6.3. Algunas propiedades del fragmento imperativo

- $\mathbf{skip} =_{def} \langle\rangle$
- $e; e' \ =_{def}\ \mathbf{let}\ v = e\ \mathbf{in}\ e' \qquad (v \notin FV\ e')$

$$\frac{\sigma, e \Rightarrow z, \sigma' \quad \sigma', e' \Rightarrow z', \sigma''}{\sigma, e; e' \Rightarrow z', \sigma''}$$

$$[\![e; e']\!]\eta\sigma = (\lambda \langle \sigma', z \rangle\ .\ [\![e']\!]\eta\sigma')_*([\![e]\!]\eta\sigma)$$

- $\mathbf{newvar}\ v\ \mathbf{in}\ = e\ \mathbf{in}\ e' \ =_{def}\ \mathbf{let}\ v = \mathbf{ref}\ e\ \mathbf{in}\ e'$

$$\frac{\sigma, e \Rightarrow z, \sigma' \quad [\sigma'|r : z], (e'/v \mapsto r) \Rightarrow z', \sigma''}{\sigma, \mathbf{newvar}\ v\ \mathbf{in}\ := e\ \mathbf{in}\ e' \Rightarrow z', \sigma''} \quad (r = new(dom\ \sigma'))$$

$$[\![\mathbf{newvar}\ v\ \mathbf{in}\ = e\ \mathbf{in}\ e']\!]\eta\sigma = (\lambda \langle \sigma', z \rangle\ .[\![e']\!][\eta|v : \iota_{ref} r][\sigma'|r : z])_*([\![e]\!]\eta\sigma) \quad \text{donde } r = new(dom\ \sigma')$$

- $\mathbf{while}\ e\ \mathbf{do}\ \ \mathbf{do}\ e' \ =_{def}\ \mathbf{letrec}\ w = \lambda v.\ \mathbf{if}\ e\ \mathbf{then}\ e'; w\ v\ \mathbf{else\ skip\ in}\ w\ \langle\rangle$
  donde w y v no deben ocurrir en e ni e'

$$\frac{\sigma, e \Rightarrow \mathbf{false}, \sigma'}{\mathbf{while}\ e\ \mathbf{do}\ \ \mathbf{do}\ e', s \Rightarrow \langle\rangle, \sigma'} \qquad \frac{\sigma, e \Rightarrow \mathbf{true}, \sigma' \quad \sigma, e'; \mathbf{while}\ e\ \mathbf{do}\ e' \Rightarrow z', \sigma''}{\sigma, \mathbf{while}\ e\ \mathbf{do}\ e' \Rightarrow z', \sigma''}$$