1 Especificación Formal del Lenguaje

Sintaxis Concreta

⟨*expr*⟩ ::= ⟨*id*⟩
| ⟨*num*⟩
| ⟨*bool*⟩
| ⟨*string*⟩
| (⟨*op*⟩ ⟨*expr*⟩ {⟨*expr*⟩})
| (let ([⟨*id*⟩ ⟨*expr*⟩] {[⟨*id*⟩ ⟨*expr*⟩]} ⟨*expr*⟩)
| (let* ([⟨*id*⟩ ⟨*expr*⟩] {[⟨*id*⟩ ⟨*expr*⟩]} ⟨*expr*⟩)
| (if ⟨*expr*⟩ ⟨*expr*⟩ ⟨*expr*⟩)
| (cond ({[⟨*expr*⟩ ⟨*expr*⟩]}) (else ⟨*expr*⟩))
| (lambda (⟨*id*⟩ {⟨*id*⟩}) ⟨*expr*⟩)
| (⟨*expr*⟩ ⟨*expr*⟩ {⟨*expr*⟩})
| (letrec (⟨*id*⟩ ⟨*expr*⟩) ⟨*expr*⟩)
| (list {⟨*expr*⟩})

⟨*num*⟩ ::= ... 2.5 | -1 | 0 | 1 | 18.35 ...

⟨*bool*⟩ ::= #t | #f

⟨*string*⟩ ::= "a" | "b" | "Hello␣world!" | ...

⟨*op*⟩ ::= + | - | * | / | add1 | sub1 | sqrt | expt |
< | > | = | not | or | and |
head | tail | length | reverse | concat | map | filter |
sconcat | at | lstostr

⟨*id*⟩ ::= a | b | foo | ...

Sintaxis Abstracta Endulzada

$$Ops = \{\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{add1}, \texttt{sub1}, \texttt{sqrt}, \texttt{expt}, \texttt{<}, \texttt{>}, \texttt{=}, \texttt{not}, \texttt{or}, \texttt{and}, \texttt{head}, \texttt{tail}, \texttt{length}, \texttt{reverse}, \texttt{concat}, \texttt{map}, \texttt{filter} \texttt{sconcat}, \texttt{at}, \texttt{lstostr}\}$$

$$\texttt{Binding} \subseteq \texttt{String} \times \texttt{SASA}$$

$$\frac{i\texttt{:String}}{IdS(i)\texttt{:SASA}} \qquad \frac{b\texttt{:[Binding]} \quad c\texttt{:SASA}}{Let1(b,\, c)\texttt{:SASA}} \qquad \frac{i\texttt{:String} \quad v\texttt{:SASA} \quad c\texttt{:SASA}}{Letrec(i,\, v,\, c)\texttt{:SASA}}$$

$$\frac{n \in \mathbb{Z}}{NumS(n)\texttt{:SASA}} \qquad \frac{c\texttt{:SASA} \quad t\texttt{:SASA} \quad e\texttt{:SASA}}{If(c,\, t,\, e)\texttt{:SASA}}$$

$$\frac{b \in \mathbb{B}}{BooleanS(b)\texttt{:SASA}} \qquad \frac{cs\texttt{:[(SASA, SASA)]} \quad e\texttt{:SASA}}{Cond(cs,\, e)\texttt{:SASA}} \qquad \frac{l\texttt{:[SASA]}}{List(l)\texttt{:SASA}}$$

$$\frac{f \in Ops \quad args\texttt{:[SASA]}}{Op(f,\, args)\texttt{:SASA}} \qquad \frac{p\texttt{:[String]} \quad c\texttt{:SASA}}{Fun(p,\, c)\texttt{:SASA}}$$

$$\frac{b\texttt{:[Binding]} \quad c\texttt{:SASA}}{Let(b,\, c)\texttt{:SASA}} \qquad \frac{f\texttt{:SASA} \quad a\texttt{:[SASA]}}{App(f,\, a)\texttt{:SASA}} \qquad \frac{s\texttt{:String}}{StringS(s)\texttt{:SASA}}$$

Sintaxis Abstracta Desendulzada

$$U = \{\texttt{add1}, \texttt{sub1}, \texttt{sqrt}, \texttt{not}, \texttt{head}, \texttt{lstostr}\}$$

$$B = \{\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{expt}, \texttt{<}, \texttt{>}, \texttt{=}, \texttt{or}, \texttt{and}, \texttt{map}, \texttt{filter}, \texttt{sconcat}, \texttt{at}\}$$

$$U_{ns} = \{\texttt{tail}, \texttt{length}, \texttt{reverse}\}$$

$$B_{ns} = \{\texttt{concat}\}$$

$$\frac{i\texttt{:String}}{Id(i)\texttt{:ASA}} \qquad \frac{f \in B \quad i\texttt{:ASA} \quad d\texttt{:ASA}}{Binop(f,\, i,\, d)\texttt{:ASA}} \qquad \frac{l\texttt{:[ASA]}}{List(l)\texttt{:ASA}}$$

$$\frac{n \in \mathbb{Z}}{Num(n)\texttt{:ASA}} \qquad \frac{c\texttt{:ASA} \quad t\texttt{:ASA} \quad e\texttt{:ASA}}{If(c,\, t,\, e)\texttt{:ASA}} \qquad \frac{s\texttt{:String}}{String(s)\texttt{:ASA}}$$

$$\frac{b \in \mathbb{B}}{Boolean(b)\texttt{:ASA}} \qquad \frac{p\texttt{:String} \quad c\texttt{:ASA}}{Fun(p,\, c)\texttt{:ASA}}$$

$$\frac{f \in U \quad arg\texttt{:ASA}}{Unop(f,\, arg)\texttt{:ASA}} \qquad \frac{f\texttt{:ASA} \quad a\texttt{:ASA}}{App(f,\, a)\texttt{:ASA}}$$

Valores finales

$$\frac{n \in \mathbb{Z}}{NumV(n)\texttt{:Value}} \qquad \frac{p\texttt{:String} \quad c\texttt{:Value} \quad \varepsilon\texttt{:Env}}{\langle p,\, c,\, \varepsilon\rangle\texttt{:Value}} \qquad \frac{l\texttt{:[Value]}}{ListV(l)\texttt{:Value}}$$

$$\frac{b \in \mathbb{B}}{BooleanV(b)\texttt{:Value}} \qquad \frac{e\texttt{:ASA} \quad \varepsilon\texttt{:Env}}{\langle e,\, \varepsilon\rangle\texttt{:Value}} \qquad \frac{s\texttt{:String}}{StringV(s)\texttt{:Value}}$$

Semántica Natural

Identificadores se buscan en el ambiente y se retorna error de variable libre si no son encontrados

$$\frac{}{Id(i), \varepsilon \Rightarrow \varepsilon(i)}$$

Numeros se reducen a si mismos

$$\frac{}{Num(n), \varepsilon \Rightarrow NumV(n)}$$

Booleanos se reducen a si mismos

$$\frac{}{Boolean(b), \varepsilon \Rightarrow BooleanV(b)}$$

Listas se reducen a si mismas

$$\frac{}{List(l), \varepsilon \Rightarrow ListV(l)}$$

Strings se reducen a si mismas

$$\frac{}{String(s), \varepsilon \Rightarrow StringV(s)}$$

Las operaciones unarias y binarias que requieren puntos estrictos (U, B) se interpretan mediante la operación correspondiente

$$\frac{elige(f) = g \qquad arg, \varepsilon \Rightarrow a \qquad strict(a) = v' \qquad g(v') = v''}{Unop(f,\ arg), \varepsilon \Rightarrow v''}$$

$$\frac{elige(f) = g \qquad i, \varepsilon \Rightarrow i' \quad d, \varepsilon \Rightarrow d' \qquad strict(i') = i'' \quad strict(d') = d'' \qquad g(i'', d'') = v}{Binop(f, i, d), \varepsilon \Rightarrow v}$$

Donde *elige* es una función que transforma la operación en sintaxis abstracta a la operación correspondiente en el lenguaje anfitrión

Las operaciones unarias y binarias que **no** requieren puntos estrictos (U_{ns}, B_{ns}) se interpretan mediante la operación correspondiente

$$\frac{elige(f) = g \qquad arg, \varepsilon \Rightarrow a \qquad g(a) = v''}{Unop_{ns}(f,\ arg), \varepsilon \Rightarrow v''}$$

$$\frac{elige(f) = g \qquad i, \varepsilon \Rightarrow i' \quad d, \varepsilon \Rightarrow d' \qquad g(i', d') = v}{Binop_{ns}(f, i, d), \varepsilon \Rightarrow v}$$

Donde *elige* es una función que transforma la operación en sintaxis abstracta a la operación correspondiente en el lenguaje anfitrión

Condicional `if`

$$\frac{c, \varepsilon \Rightarrow c' \qquad strict(c') = Boolean(True) \qquad t, \varepsilon \Rightarrow t'}{If(c, t, e), \varepsilon \Rightarrow t'}$$

$$\frac{c, \varepsilon \Rightarrow c' \qquad strict(c') = Boolean(False) \qquad e, \varepsilon \Rightarrow e'}{If(c, t, e), \varepsilon \Rightarrow e'}$$

Expresiones `lambda`

$$\frac{}{Fun(p, c), \varepsilon \Rightarrow \langle p, c, \varepsilon \rangle}$$

Aplicaciones de función

$$\frac{f, \varepsilon \Rightarrow f' \qquad strict(f') = \langle p, c, \varepsilon' \rangle \qquad c, \varepsilon'[p \leftarrow \langle a, \varepsilon \rangle] \Rightarrow c_v}{App(f, a), \varepsilon \Rightarrow c_v}$$

Notemos que el `letrec` se convertirá en una aplicación de función con el combinador Y

```
(letrec (ft (lambda (n) (if (= n 0) 1 (* n (ft (- n 1)))))) (ft 5)) ⇒
((lambda (ft) (ft 5)) (Y (lambda (ft) (lambda (n) (if (= n 0) 1 (* n (ft (- n 1))))))))
```

Lo cuál es igual a

```
((Y (lambda (ft) (lambda (n) (if (= n 0) 1 (* n (ft (- n 1))))))) 5)
```

$ft = \lambda ft.\lambda n.$ if $n = 0$ then 1 else $n \cdot ft(n-1)$

$(Y\ ft)\ 5$

2 Algoritmo de Inferencia de Tipos

El **Algoritmo de Inferencia de Tipos** es un proceso que permite deducir el tipo de las expresiones en un programa sin necesidad de anotaciones explícitas. Uno de los algoritmos más utilizados es el **Algoritmo W**, asociado al sistema de tipos Hindley-Milner [**?**].

2.1 Descripción del Algoritmo W

El Algoritmo W funciona recorriendo el árbol sintáctico abstracto del programa y generando un conjunto de ecuaciones de tipos (también conocidas como restricciones de tipos). Luego, resuelve estas ecuaciones mediante unificación, encontrando el tipo más general que satisface todas las restricciones [**?**].

2.2 Restricciones

Identificadores
Igualar todas las apariciones
$[\![x_1]\!] = [\![x_2]\!] = \cdots = [\![x_n]\!]$
Que en realidad genera
$[\![x_1]\!] = [\![x_2]\!]$
...
$[\![x_1]\!] = [\![x_n]\!]$

Números
$[\![n]\!] =$ `number`

Booleanos
$[\![b]\!] =$ `boolean`

Strings
$[\![s]\!] =$ `string`

Listas
$[\![l]\!] =$ `list`

Operaciones aritméticas
$op = \{$`add1`, `sub1`, `sqrt`, `+`, `-`, `*`, `/`, `expt`$\}$
$[\![$ `(`$op\ n_1\ \ldots\ n_i$`)` $]\!] =$ `number`
$[\![n_1]\!] =$ `number`
...
$[\![n_i]\!] =$ `number`

Comparaciones
$op = \{$`<`, `>`, `=`$\}$
$[\![$ `(`$op\ n_1\ \ldots\ n_i$`)` $]\!] =$ `boolean`
$[\![n_1]\!] =$ `number`
...
$[\![n_i]\!] =$ `number`

Operaciones booleanas
$op = \{$`not`, `or`, `and`$\}$
$[\![$ `(`$op\ b_1\ \ldots\ b_n$`)` $]\!] =$ `boolean`
$[\![b_1]\!] =$ `boolean`
...
$[\![b_n]\!] =$ `boolean`

If
$[\![$ `(if` $c\ t\ e$`)` $]\!] = [\![t]\!]$
$[\![$ `(if` $c\ t\ e$`)` $]\!] = [\![e]\!]$
$[\![c]\!] =$ `boolean`
$[\![t]\!] = [\![e]\!]$

Cond
$[\![\texttt{(cond ([}c_1\ t_1\texttt{] ... [}c_n\ t_n\texttt{]) (else } e\texttt{))}]\!] = [\![t_1]\!]$
...
$[\![\texttt{(cond ([}c_1\ t_1\texttt{] ... [}c_n\ t_n\texttt{]) (else } e\texttt{))}]\!] = [\![t_n]\!]$
$[\![\texttt{(cond ([}c_1\ t_1\texttt{] ... [}c_n\ t_n\texttt{]) (else } e\texttt{))}]\!] = [\![e]\!]$
$[\![c_1]\!] = \texttt{boolean}$
...
$[\![c_n]\!] = \texttt{boolean}$
$[\![t_1]\!] = [\![t_2]\!]$
...
$[\![t_1]\!] = [\![t_n]\!]$
$[\![t_1]\!] = [\![e]\!]$

Let
$[\![\texttt{(let ([}i_1\ v_1\texttt{] ... [}i_n\ v_n\texttt{]) } c\texttt{)}]\!] = [\![c]\!]$
$[\![i_1]\!] = [\![v_1]\!]$
...
$[\![i_n]\!] = [\![v_n]\!]$

Let*
$[\![\texttt{(let* ([}i_1\ v_1\texttt{] ... [}i_n\ v_n\texttt{]) } c\texttt{)}]\!] = [\![c]\!]$
$[\![i_1]\!] = [\![v_1]\!]$
...
$[\![i_n]\!] = [\![v_n]\!]$

Funciones
$[\![\texttt{(lambda (}p_1\ \ldots\ p_n\texttt{) } b\texttt{)}]\!] = [\![p_1]\!] \to \cdots \to [\![p_n]\!] \to [\![b]\!]$

Aplicaciones de función $\texttt{(}f\ \ a_1\ \ldots\ a_n\texttt{)}$
$[\![f]\!] = [\![a_1]\!] \to \cdots \to [\![a_n]\!] \to [\![\texttt{(}f\ \ a_1\ \ldots\ a_n\texttt{)}]\!]$

Concatenación de Strings
$[\![\texttt{(sconcat } s_1\ \ldots\ s_n\texttt{)}]\!] = \texttt{string}$
$[\![s_1]\!] = \texttt{string}$
...
$[\![s_n]\!] = \texttt{string}$

At (Obtener cadena en un índice)
$[\![\texttt{(at } n\ s\texttt{)}]\!] = \texttt{string}$
$[\![n]\!] = \texttt{number}$
$[\![s]\!] = \texttt{string}$

Head
$[\![\texttt{(head } l\texttt{)}]\!] = T_{uuid}$
$[\![l]\!] = \texttt{list}$

Tail
$[\![\texttt{(tail } l\texttt{)}]\!] = \texttt{list}$
$[\![l]\!] = \texttt{list}$

Length
$[\![\texttt{(length } l\texttt{)}]\!] = \texttt{number}$
$[\![l]\!] = \texttt{list}$

Reverse
$[\![\texttt{(reverse } l\texttt{)}]\!] = \texttt{list}$
$[\![l]\!] = \texttt{list}$

Concatenación
$[\![\texttt{(concat } l_1\ l_2\texttt{)}]\!] = \texttt{list}$
$[\![l_1]\!] = \texttt{list}$
$[\![l_2]\!] = \texttt{list}$

Lista a String
$[\![\texttt{(lstostr } l\texttt{)}]\!] = \texttt{string}$
$[\![l]\!] = \texttt{list}$

Filter
$[\![\texttt{(filter } f\ l\texttt{)}]\!] = \texttt{list}$
$[\![f = \texttt{(lambda (}p\texttt{) } b\texttt{)}]\!] = [\![p]\!] \rightarrow [\![b]\!]$
$[\![b]\!] = \texttt{boolean}$
$[\![l]\!] = \texttt{list}$

Map
$[\![\texttt{(map } f\ l\texttt{)}]\!] = \texttt{list}$
$[\![f = \texttt{(lambda (}p\texttt{) } b\texttt{)}]\!] = [\![p]\!] \rightarrow [\![b]\!]$
$[\![l]\!] = \texttt{list}$