1. Funciones de haskell

```haskell
foldr, foldl :: Foldable t => (a -> b -> b) -> b -> t a -> b
foldr1, foldl1 :: Foldable t => (a -> a -> a) -> t a -> a

map :: (a -> b) -> [a] -> [b]
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]

(++) :: [a] -> [a] -> [a]
(!!) :: [a] -> Int -> a
head, last :: [a] -> a
init, tail :: [a] -> [a]
length :: Foldable t => t a -> Int

reverse :: [a] -> [a]
concat :: Foldable t => t [a] -> [a]
union :: Eq a => [a] -> [a] -> [a]

all, any :: Foldable t => (a -> Bool) -> t a -> Bool

null :: Foldable t => t a -> Bool -- Es vacio
elem :: (Eq a, Foldable t) => a -> t a -> Bool


nub :: Eq a => [a] -> [a] -- Elimina duplicados
sort :: Ord a => [a] -> [a] -- Ordena la lista

concatMap :: Foldable t => (a -> [b]) -> t a -> [b]
find :: (a -> Bool) -> [a] -> Maybe a
filter :: (a -> Bool) -> [a] -> [a]
iterate :: (a -> a) -> a -> [a]
span :: (a -> Bool) -> [a] -> ([a], [a])
replicate :: Int -> a -> [a]
take, drop :: Int -> [a] -> [a]
takeWhile, dropWhile :: (a -> Bool) -> [a] -> [a]

and, or :: Foldable t => t Bool -> Bool
maximum, minimum :: (Ord a, Foldable t) => t a -> a
sum :: (Num a, Foldable t) => t a -> a

max, min :: Ord a => a -> a -> a
rem :: Integral a => a -> a -> a
ord :: Char -> Int
chr :: Int -> Char

fromJust :: Maybe a -> a
isNothing :: Maybe a -> Bool
lookup :: Eq a => a -> [(a, b)] -> Maybe b
maybe :: b -> (a -> b) -> Maybe a -> b
```

2. Esquemas de recursión

```haskell
map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (x:xs) = (f x) : (map f xs)

filter :: (a -> Bool) -> [a] -> [a]
filter _ [] = []
filter p (x:xs) | (p x)     = x : (filter p xs)
                | otherwise = filter p xs

foldr :: (a -> b -> b) -> b -> [a] -> b
foldr _ z [] = z
foldr f z (x:xs) = f x (foldr f z xs)


foldl :: (b -> a -> b) -> b -> [a] -> b
foldl f z [] = z
foldl f z (x : xs) = foldl f (f z x) xs

recr :: b -> (a -> [a] -> b -> b) -> [a] -> b
recr z _ []= z
recr z f (x:xs) = f x xs (recr z f xs)

type DivideConquer a b  = (a -> Bool) -> (a -> b) -> (a -> [a]) ->
                          ([b] -> b) -> a -> b

divideConquerListas :: DivideConquer [a] b
-- Esto significa que DivideConquerLista es de tipo
-- ([a] -> Bool) -> ([a] -> b) -> ([a] -> [[a]]) -> ([b] -> b)
-- -> [a] -> b

divideConquerListas esTrivial resolver repartir combinar l =
        if (esTrivial l) then resolver l
        else combinar (map dc (repartir l))
where dc = divideConquerListas esTrivial resolver repartir combinar
```

3. Cálculo lambda λ^{bn}

Función Free Variables

$$\begin{aligned}
FV(x) &\stackrel{def}{=} x \\
FV(true) = FV(false) &\stackrel{def}{=} \varnothing \\
FV(if\ M\ then\ P\ else\ Q) &\stackrel{def}{=} FV(M) \cup FV(P) \cup FV(Q) \\
FV(M\ N) &\stackrel{def}{=} FV(M) \cup FV(N) \\
FV(\lambda x : \sigma.M) &\stackrel{def}{=} FV(M) \backslash \{x\}
\end{aligned}$$

Sustitución

$$\begin{aligned}
x\{x \leftarrow N\} &\stackrel{def}{=} N \\
a\{x \leftarrow N\} &\stackrel{def}{=} a \text{ si } a \in \{true, false\} \cup \mathcal{X} \backslash \{x\} \\
(if\ M\ then\ P\ else\ Q)\{x \leftarrow N\} &\stackrel{def}{=} if\ M\{x \leftarrow N\}\ then\ P\{x \leftarrow N\}\ else\ Q\{x \leftarrow N\} \\
(M_1\ M_2)\{x \leftarrow N\} &\stackrel{def}{=} M_1\{x \leftarrow N\}\ M_2\{x \leftarrow N\} \\
(\lambda y : \sigma.M)\{x \leftarrow N\} &\stackrel{def}{=} \lambda y : \sigma.M\{x \leftarrow N\}\ x \neq y,\ y \notin\ FV(N)
\end{aligned}$$

3.0.1. Tipos

$\sigma, \tau \ ::= \ Bool \mid Nat \mid \sigma \to \tau$

Expresiones

$$\begin{aligned}
M, P, Q \ ::= &\ true \mid false \mid if\ M\ then\ P\ else\ Q \\
&\mid M\ N \mid \lambda x : \sigma.M \\
&\mid x \mid 0 \mid succ(M) \mid pred(M) \mid isZero(M)
\end{aligned} \tag{1}$$

3.0.2. Reglas de tipado

$$\frac{}{\Gamma \rhd true : Bool}\text{(T-True)} \qquad \frac{}{\Gamma \rhd false : Bool}\text{(T-False)}$$

$$\frac{x : \sigma \in \Gamma}{\Gamma \rhd x : \sigma}\text{(T-Var)} \qquad \frac{\Gamma \rhd M : Bool \quad \Gamma \rhd P : \sigma \quad \Gamma \rhd Q : \sigma}{\Gamma \rhd if\ M\ then\ P\ else\ Q : \sigma}\text{(T-If)}$$

$$\frac{\Gamma, x : \sigma \rhd M : \tau}{\Gamma \rhd \lambda x : \sigma.M : \sigma \to \tau}\text{(T-Abs)} \qquad \frac{\Gamma \rhd M : \sigma \to \tau \quad \Gamma \rhd N : \sigma}{\Gamma \rhd M\ N : \tau}\text{(T-App)}$$

$$\frac{}{\Gamma \rhd 0 : Nat}\text{(T-Zero)}$$

$$\frac{\Gamma \rhd M : Nat}{\Gamma \rhd succ(M) : Nat}\text{(T-Succ)} \qquad \frac{\Gamma \rhd M : Nat}{\Gamma \rhd pred(M) : Nat}\text{(T-Pred)}$$

$$\frac{\Gamma \rhd M : Nat}{\Gamma \rhd isZero(M) : Bool}\text{(T-IsZero)}$$

Valores

$V \ ::= \ true \mid false \mid \lambda x : \sigma.M \mid \underline{n}$ donde $\underline{n}$ abrevia $succ^n(0)$

Reglas de semánticas

$$\frac{}{if\ true\ then\ M_1\ else\ M_2 \to M_1}\text{(E-IfTrue)}$$

$$\frac{}{if\ false\ then\ M_1\ else\ M_2 \to M_2}\text{(E-IfFalse)}$$

$$\frac{M_1 \to M_1'}{if\ M_1\ then\ M_2\ else\ M_3 \to if\ M_1'\ then\ M_2\ else\ M_3}\text{(E-If)}$$

$$\frac{M_1 \to M_1'}{M_1\ M_2 \to M_1'\ M_2}\text{(E-App1 / }\mu) \qquad \frac{M_2 \to M_2'}{V_1\ M_2 \to V_1\ M_2'}\text{(E-App2 / }v)$$

$$\frac{}{(\lambda x : \sigma.M)\ V \to M\{x \leftarrow V\}}\text{(E-App3 / }\beta)$$

$$\frac{M_1 \to M_1'}{succ(M_1) \to succ(M_1')}\text{(E-Succ)}$$

$$\frac{}{pred(0) \to 0}\text{(E-PredZero)} \qquad \frac{}{pred(succ(\underline{n})) \to \underline{n}}\text{(E-PredSucc)}$$

$$\frac{M_1 \to M_1'}{pred(M_1) \to pred(M_1')}\text{(E-Pred)}$$

$$\frac{}{isZero(0) \to true}\text{(E-IsZeroZero)} \qquad \frac{}{isZero(succ(\underline{n})) \to false}\text{(E-isZeroSucc)}$$

$$\frac{M_1 \to M_1'}{isZero(M_1) \to isZero(M_1')}\text{(E-isZero)}$$

4. Extensión con memoria λ^{bnu}

Tipos

$$\sigma, \tau \ ::= \ Bool \mid Nat \mid Unit \mid Ref\ \sigma \mid \sigma \to \tau$$

Términos

$$M \ ::= \ \ldots \mid unit \mid ref\ M \mid\ !M \mid M \ := \ N! \mid l$$

Axiomas y reglas de tipado

$$\frac{}{\Gamma|\Sigma \rhd unit : Unit}(\text{T-Unit}) \qquad \frac{\Gamma|\Sigma \rhd M_1 : \sigma}{\Gamma|\Sigma \rhd ref\ M_1 : Ref\ \sigma}(\text{T-Ref})$$

$$\frac{\Gamma|\Sigma \rhd M_1 : Ref\ \sigma}{\Gamma \rhd !M_1 : \sigma}(\text{T-DeRef})$$

$$\frac{\Gamma|\Sigma \rhd M_1 : Ref\ \sigma \qquad \Gamma|\Sigma \rhd M_2 : \sigma}{\Gamma \rhd M_1 \ := \ M_2 : Unit}(\text{T-Assing})$$

$$\frac{\Sigma(l) = \sigma}{\Gamma|Signa \rhd l : Ref\ \sigma}(\text{T-Loc})$$

Valores

$$V \ ::= \ \ldots \mid unit \mid l$$

Axiomas y reglas semánticas

$$\frac{M_1|\mu \to M_1'|\mu'}{M_1\ M_2|\mu \to M_1'\ M_2|\mu'}(\text{E-App1}) \qquad \frac{M_2|\mu \to M_2'|\mu'}{V_1\ M_2|\mu \to V_1\ M_2'|\mu'}(\text{E-App2})$$

$$\frac{}{(\lambda x : \sigma.M)\ V|\mu \to M\{x \leftarrow V\}|\mu'}(\text{E-AppAbs})$$

$$\frac{M_1|\mu \to M_1'|\mu'}{!M_1|\mu \to !M_1'|\mu'}(\text{E-DeRef}) \qquad \frac{\mu(l) = V}{!l|\mu \to V|\mu}(\text{E-DerefLoc})$$

$$\frac{M_1|\mu \to M_1'|\mu'}{M_1 \ := \ M_2\ |\mu \to M_1' \ := \ M_2|\mu'}(\text{E-Assign1})$$

$$\frac{M_2|\mu \to M_2'|\mu'}{V \ := \ M_2|\mu \to V \ := \ M_2'|\mu'}(\text{E-Assign2})$$

$$\frac{}{l \ := \ V|\mu \to unit|\mu[l \to V]}\text{(E-Assign)}$$

$$\frac{M_1|\mu \to M_1'|\mu'}{ref\ M_1|\mu \to ref\ M_1'|\mu'}\text{(E-Ref)} \qquad \frac{l \notin Dom(\mu)}{ref\ V|\mu \to l|\mu \oplus (l \to V)}\text{(E-RefV)}$$

5. Extensión con recursión $\lambda^{..r}$

Términos

$$M \ := \ \ldots \mid fix\ M$$

Regla de tipado

$$\frac{\Gamma \rhd M : \sigma \to \sigma}{\Gamma \rhd fix\ M : \sigma}\text{(T-Fix)}$$

Reglas de evaluación

$$\frac{M_1 \to M_1'}{fix\ M_1 \to fix\ M_1'}\text{(E-Fix)}$$

$$\frac{}{fix\ (\lambda x : \sigma.M) \to M\{x \leftarrow fix\ \lambda x : \sigma.M\}}\text{(E-FixBeta)}$$

6. Extensión con Declaraciones Locales ($\lambda^{...let}$)

Con esta extensión, agregamos al lenguaje el término $let\ x : \sigma = M\ in\ N$, que evalúa M a un valor, liga x a V y, luego, evalúa N. Este término solo mejora la legibilidad de los programas que ya podemos definir con el lenguaje hasta ahora definido.

Términos

$$M \ ::= \ \ldots \mid let\ x : \sigma = M\ in\ N$$

Axiomas y reglas de tipado

$$\frac{\Gamma \rhd M : \sigma_1 \qquad \Gamma, x : \sigma_1 \rhd N : \sigma_2}{\Gamma \rhd let\ x : \sigma_1 = M\ in\ N : \sigma_2}\text{(T-Let)}$$

Axiomas y reglas de evaluación

$$\frac{M_1 \to M_1'}{let\ x : \sigma = M_1\ in\ M_2 \to let\ x : \sigma = M_1'\ in\ M_2}\text{(E-Let)}$$

$$\frac{}{let\ x : \sigma = V_1\ in\ M_2 \to M_2\{x \leftarrow V_1\}}\text{(E-LetV)}$$

6.0.1. Construcción *let* recursivo (Letrec)

Una construcción alternativa para definir funciones recursivas es

$$letrec\ f : \sigma \to \sigma = \lambda x : \sigma.M\ in\ N$$

Y $letRec$ se puede definir en base a let y fix (definido en **??**) de la siguiente forma:

$$let\ f : \sigma \to \sigma = (fix\ \lambda f : \sigma \to \sigma.\lambda x : \sigma.M)\ in\ N$$

7. Extensión con Registros $\lambda^{\dots r}$

Tipos

$$\sigma, \tau \ ::= \ \dots \mid \{l_i : \sigma_i \ ^{i \in 1..n}\}$$

El tipo $\{l_i : \sigma_i^{i \in 1..n}\}$ representan las estructuras con n atributos tipados, por ejemplo: $\{nombre : String, edad : Nat\}$

Términos

$$M \ ::= \ \dots \mid \{l_i = M_i \ ^{i \in 1..n}\} \mid M.l$$

Los términos significan:

- El registro $\{l_i = M_i \ ^{i \in 1..n}\}$ evalua $\{l_i = V_i \ ^{i \in 1..n}\}$ donde V_i es el valores al que evalúa M_i para $i \in 1..n$.
- $M.l$: Proyecta el valor de la etiqueta l del registro M

Axiomas y reglas de tipado

$$\frac{\Gamma \rhd M_i : \sigma_i \text{ para cada } i \in 1..n}{\Gamma \rhd \{l_i = M_i \ ^{i \in 1..n}\} : \{l_i : \sigma_i \ ^{i \in 1..n}\}} \text{(T-RCD)}$$

$$\frac{\Gamma \rhd M : \{l_i : \sigma_i \ ^{i \in 1..n}\} \qquad j \in 1..n}{\Gamma \rhd M.l_j : \sigma_j} \text{(T-Proj)}$$

Valores

$$V \ ::= \ \dots \mid \{l_i = V_i \ ^{i \in 1..n}\}$$

Axiomas y reglas de evaluación

$$\frac{j \in 1..n}{\{l_i = V_i \ ^{i \in 1..n}\}.l_j \to V_j} \text{(E-ProjRcd)}$$

$$\frac{M \to M'}{M.l \to M'.l} \text{(E-Proj)}$$

$$\frac{M_j \to M'_j}{\{l_i = V_i \ ^{i \in 1..j-1}, l_j = M_j, l_i = M_i \ ^{i \in j+1..n}\} \to \{l_i = V_i \ ^{i \in 1..j-1}, l_j = M'_j, l_i = M_i \ ^{i \in j+1..n}\}} \text{(E-RCD)}$$

8. Extensión con tuplas

Tipos

$$\sigma, \tau \ ::= \ldots \mid \sigma \times \tau$$

Términos

$$M,\ N\ ::=\ \ldots \mid\ < M, N > \ \mid \pi_1(M) \mid \pi_2(M)$$

Axiomas y reglas de tipado

$$\frac{\Gamma \rhd M : \sigma \qquad \Gamma \rhd N : \tau}{\Gamma \rhd < M, N >: \sigma \times \tau}(\text{T-Tupla})$$

$$\frac{\Gamma \rhd M : \sigma \times \tau}{\Gamma \rhd \pi_1(M) : \sigma}(\text{T-}\pi_1) \qquad \frac{\Gamma \rhd M : \sigma \times \tau}{\Gamma \rhd \pi_2(M) : \tau}(\text{T-}\pi_2)$$

Valores

$$V\ ::=\ \ldots \mid\ < V, V >$$

Axiomas y reglas de evaluación

$$\frac{M \to M'}{< M, N > \to < M', N >}(\text{E-Tuplas}) \qquad \frac{N \to N'}{< V, N > \to < V, N' >}(\text{E-Tuplas1})$$

$$\frac{M \to M'}{\pi_1(M) \to \pi_1(M')}(\text{E-}\pi_1) \qquad \frac{}{\pi_1(< V_1, V_2 >) \to V_1}(\text{E-}\pi_1')$$

$$\frac{M \to M'}{\pi_2(M) \to \pi_2(M')}(\text{E-}\pi_2) \qquad \frac{}{\pi_2(< V_1, V_2 >) \to V_2}(\text{E-}\pi_2')$$

9. Extensión con árboles binarios

Tipos

$$\sigma, \tau \ ::= \ \ldots \ | \ AB_\sigma$$

Términos

$$M, \ N \ ::= \ \ldots \ | \ \mathrm{Nil}_\sigma \ | \ \mathrm{Bin}(M, N, O) \ | \ \mathrm{raiz}(M) \ | \ \mathrm{der}(M) \ | \ \mathrm{izq}(M) \ | \ \mathrm{esNil}(M)$$

Axiomas y reglas de tipado

$$\frac{}{\Gamma \rhd \mathrm{Nil}_\sigma : AB_\sigma}(\text{T-Nil}) \qquad \frac{\Gamma \rhd M : AB_\sigma \quad \Gamma \rhd N : \sigma \quad \Gamma \rhd O : AB_\sigma}{\Gamma \rhd \mathrm{Bin}(M, N, O) : AB_\sigma}(\text{T-Bin})$$

$$\frac{\Gamma \rhd M : AB_\sigma}{\Gamma \rhd \mathrm{raiz}(M) : \sigma}(\text{T-raiz}) \qquad \frac{\Gamma \rhd M : AB_\sigma}{\Gamma \rhd \mathrm{der}(M) : AB_\sigma}(\text{T-der})$$

$$\frac{\Gamma \rhd M : AB_\sigma}{\Gamma \rhd \mathrm{izq}(M) : AB_\sigma}(\text{T-izq}) \qquad \frac{\Gamma \rhd M : AB_\sigma}{\Gamma \rhd \mathrm{isNil}(M) : Bool}(\text{T-isNil})$$

Valores

$$V \ ::= \ \ldots \ | \ \mathrm{Nil} \ | \ \mathrm{Bin}(V, V, V)$$

Axiomas y reglas de evaluación

$$\frac{M \to M'}{\mathrm{Bin}(M, N, O) \to \mathrm{Bin}(M', N, O)}(\text{E-Bin1}) \qquad \frac{N \to N'}{\mathrm{Bin}(V, N, O) \to \mathrm{Bin}(V, N', O)}(\text{E-Bin2})$$

$$\frac{O \to O'}{\mathrm{Bin}(V_1, V_2, O) \to \mathrm{Bin}(V_1, V_2, O')}(\text{E-Bin3})$$

$$\frac{M \to M'}{\mathrm{raiz}(M) \to \mathrm{raiz}(M')}(\text{E-Raiz1}) \qquad \frac{}{\mathrm{raiz}(\mathrm{Bin}(V_1, V_2, V_3)) \to V_2}(\text{E-Bin3})$$

$$\frac{M \to M'}{\mathrm{der}(M) \to \mathrm{der}(M')}(\text{E-Der1}) \qquad \frac{}{\mathrm{der}(\mathrm{Bin}(V_1, V_2, V_3)) \to V_3}(\text{E-Der2})$$

$$\frac{M \to M'}{\mathrm{izq}(M) \to \mathrm{izq}(M')}(\text{E-Izq1}) \qquad \frac{}{\mathrm{izq}(\mathrm{Bin}(V_1, V_2, V_3)) \to V_1}(\text{E-Izq2})$$

$$\frac{}{\mathrm{isNil}(M) \to \mathrm{izq}(M')}(\text{E-isNil1}) \qquad \frac{}{\mathrm{isNil}(\mathrm{Bin}(V_1, V_2, V_3)) \to false}(\text{E-isNilBin})$$

$$\frac{}{\mathrm{isNil}(\mathrm{Bin}(V_1, V_2, V_3)) \to true}(\text{E-isNilNil})$$

10. Algoritmo de Martelli-Montanari

1. **Descomposición**

   $\{\sigma_1 \to \sigma_2 \doteq \tau_1 \to \tau_2\} \cup G \mapsto \{\sigma_1 \doteq \tau_1,\ \sigma_2 \doteq \tau_2\} \cup G$

2. **Eliminación de par trivial**

   $\{Nat \doteq Nat\} \cup G \mapsto G$

   $\{Bool \doteq Bool\} \cup G \mapsto G$

   $\{\mathrm{s} \doteq \mathrm{s}\} \cup G \mapsto G$

3. **Swap** Si σ no es una variable,

   $\{\sigma \doteq \mathrm{s}\} \cup G \mapsto \{\mathrm{s} \doteq \sigma\} \cup G$

4. **Eliminación de variable** Si $s \notin FV(\sigma)$

   $\{\mathrm{s} \doteq \sigma\} \cup G \mapsto_{\sigma/s} G[\sigma/s]$

5. **Falla**

   $\{\sigma \doteq \tau\} \cup G \mapsto$ `falla`, con $(\sigma, \tau) \in T \cup T^{-1}$ y $T = \{(Bool, Nat), (Nat, \sigma_1 \to \sigma_2), (Bool, \sigma_1 \to \sigma_2\}$. Acá, la notación T^{-1} se refiere al conjunto con cada tupla de T invertida.

6. **Occur Check** Si $s \neq \sigma$ y $s \in FV(\sigma)$

   $\{\mathrm{s} \doteq \sigma\} \cup G \mapsto$ `falla`

11. Función $\mathbb{W}$

Constantes y variables

$$\mathbb{W}(true) \stackrel{def}{=} \varnothing \rhd true : Bool$$

$$\mathbb{W}(false) \stackrel{def}{=} \varnothing \rhd false : Bool$$

$$\mathbb{W}(x) \stackrel{def}{=} \{x : s\} \rhd x : s, \ \ s \text{ variable fresca}$$

$$\mathbb{W}(0) \stackrel{def}{=} \varnothing \rhd 0 : Nat$$

Caso *succ*

$\mathbb{W}(succ(U)) \stackrel{def}{=} S\Gamma \rhd S\ succ(M) : Nat$

- $\mathbb{W}(U) = \Gamma \rhd M : \tau$
- $S = MGU\{\tau \doteq Nat\}$

Caso *pred*

$\mathbb{W}(pred(U)) \stackrel{def}{=} S\Gamma \rhd S\ pred(M) : Nat$

- $\mathbb{W}(U) = \Gamma \rhd M : \tau$
- $S = MGU\{\tau \doteq Nat\}$

Caso *isZero*

$\mathbb{W}(isZero(U)) \stackrel{def}{=} S\Gamma \rhd S\ isZero(M) : Bool$

- $\mathbb{W}(U) = \Gamma \rhd M : \tau$
- $S = MGU\{\tau \doteq Nat\}$

Caso *ifThenElse*

$\mathbb{W}(\textit{if } U \textit{ then } V \textit{ else } W) \stackrel{def}{=} S\Gamma_1 \cup S\Gamma_2 \cup S\Gamma_3 \rhd S\ (\textit{if } M \textit{ then } P \textit{ else } Q) : S\sigma$

- $\mathbb{W}(U) = \Gamma_1 \rhd M : \rho$
- $\mathbb{W}(V) = \Gamma_2 \rhd P : \sigma$
- $\mathbb{W}(W) = \Gamma_3 \rhd Q : \tau$
- $S = MGU\{\sigma_1 \doteq \sigma_2 \mid x : \sigma_1 \in \Gamma_i \ \wedge \ x : \sigma_2 \in \Gamma_j,\ i \neq j\} \cup \{\sigma \doteq \tau \ \ \rho \doteq Bool\}$

Caso aplicación

$\mathbb{W}(U\ V) \stackrel{def}{=} S\Gamma_1 \cup S\Gamma_2 \rhd S\ (M\ N) : St$

- $\mathbb{W}(U) = \Gamma_1 \rhd M : \tau$
- $\mathbb{W}(V) = \Gamma_2 \rhd N : \rho$
- $S = MGU\{\sigma_1 \doteq \sigma_2 \mid x : \sigma_1 \in \Gamma_i \ \wedge \ x : \sigma_2 \in \Gamma_j,\ i \neq j\} \cup \{\tau \doteq \rho \to t\}$ con t variable fresca

Caso abstracción

Sea $\mathbb{W}(U) = \Gamma \rhd M : \rho$, si Γ tiene información de tipos para x, es decir $x : \tau \in \Gamma$ para algún τ, entonces:

$$\mathbb{W}(\lambda x.U) \stackrel{def}{=} \Gamma \backslash \{x : \tau\} \rhd \lambda x : \tau.M : \tau \to \rho$$

Si Γ no tiene información de tipos para x ($x \notin \mathrm{Dom}(\Gamma)$), entonces elegimos una variable fresca s y

$$\mathbb{W}(\lambda x.U) \stackrel{def}{=} \Gamma \rhd \lambda x : s.M : s \to \rho$$

Caso *fix*

$\mathbb{W}(\textit{fix}\ (U)) \stackrel{def}{=} S\Gamma \rhd S\ \textit{fix}\ (M) : St$

- $\mathbb{W}(U) = \Gamma_1 \rhd M : \tau$
- $S = MGU\{\tau \doteq t \to t\}$ con t variable fresca

12. Subtipado

$$\frac{}{Nat <: Float}\text{(S-NatFloat)} \qquad \frac{}{Int <: Float}\text{(S-IntFloat)} \qquad \frac{}{Bool <: Nat}\text{(S-BoolNat)}$$

$$\frac{\sigma' <: \sigma \quad \tau <: \tau'}{\sigma \to \tau <: \sigma' \to \tau'}\text{(S-Func)}$$

$$\frac{}{\sigma <: \sigma}\text{(S-Refl)} \qquad \frac{\sigma <: \tau \quad \tau <: \rho}{\sigma <: \rho}\text{(S-Trans)}$$

$$\frac{\sigma <: \tau \quad \tau <: \sigma}{Ref\ \tau <: Ref\ \sigma}$$

$$\frac{\sigma <: \tau}{Source\ \sigma <: Source\ \tau}\text{(S-Source)} \qquad \frac{\tau <: \sigma}{Sink\ \sigma <: Sink\ \tau}\text{(S-Sink)}$$

$$\frac{}{Ref\ \tau <: Source\ \tau}\text{(S-RefSource)} \qquad \frac{}{Ref\ \tau <: Sink\ \tau}\text{(S-RefSink)}$$

12.1. Reglas de reduccion con subtipado

$$\frac{x : \sigma \in \Gamma}{\Gamma \mapsto x : \sigma}\text{(T-Var)}$$

$$\frac{\Gamma, x : \sigma \mapsto M : \tau}{\Gamma \mapsto \lambda x : \sigma.M : \sigma \to \tau}\text{(T-Abs)} \qquad \frac{\Gamma \mapsto M : \sigma \to \tau \quad \Gamma \mapsto N : \rho \quad \rho <: \sigma}{\Gamma \mapsto M\ N : \tau}\text{(T-App)}$$

13. Objetos

13.0.1. Sintaxis

$$\begin{array}{lcll} a,b & ::= & x & \text{Variables} \\ & | & [l_i = \varsigma(x_i)b_i^{i\in 1..n}] & \text{Objetos} \\ & | & a.l & \text{Selección/ Envío de mensajes} \\ & | & a.l \Leftarrow \varsigma(x)b & \text{Redefinición de un método.} \end{array}$$

13.1. Variables libres

$$\begin{array}{ll} \mathrm{fv}(\varsigma(x)b) & = \mathrm{fv}(b)\backslash\{x\} \\ \mathrm{fv}(x) & = \{x\} \\ \mathrm{fv}([l_i = \varsigma(x_i)b_i^{i\in 1..n}]) & = \bigcup^{1\in 1..n} \mathrm{fv}(\varsigma(x)b) \\ \mathrm{fv}(a.l) & = \mathrm{fv}(a) \\ \mathrm{fv}(a.l \Leftarrow \varsigma(x)b) & = \mathrm{fv}(a.l) \cup \mathrm{fv}(\varsigma(x)b) \end{array}$$

13.2. Sustitución

$$\begin{array}{lll} x\{x \leftarrow c\} & = c & \\ y\{x \leftarrow c\} & = y & \text{si } x \neq y \\ ([l_i = \varsigma(x_i)b_i^{i\in 1..n}])\{x \leftarrow c\} & = [l_i = (\varsigma(x_i)b_i)\{x \leftarrow c\}^{i\in 1..n}] & \\ (a.l)\{x \leftarrow c\} & = (a\{x \leftarrow c\}).l & \\ (a.l \Leftarrow \varsigma(x)b)\{x \leftarrow c\} & = (a\{x \leftarrow c\}).l \Leftarrow (\varsigma(x)b)\{x \leftarrow c\} & \\ (\varsigma(y)b)\{x \leftarrow c\} & = (\varsigma(y')(b\{y \leftarrow y'\}\{x \leftarrow c\})) & \text{si } y' \notin \mathrm{fv}(\varsigma(y)b) \cup \mathrm{fv}(c) \cup \{x\} \end{array}$$

13.3. Semantica operacional

$$V \ ::= \ [l_i = \varsigma(x_i)b_i^{1\in 1..n}]$$

$$\frac{}{v \longrightarrow v}[\text{Obj}]$$

$$\frac{a \longrightarrow v' \quad v' \equiv [l_i = \varsigma(x_i)b_i^{i\in 1..n}] \quad b_j\{x_j \leftarrow v'\} \longrightarrow v \quad j \in 1..n}{a.l_j \longrightarrow v}[\text{Sel}]$$

$$\frac{a \longrightarrow [l_i = \varsigma(x_i)b_i^{i\in 1..n}] \quad j \in 1..n}{a.l_j \Leftarrow \varsigma(x)b \longrightarrow [l_j = \varsigma(x)b,\ l_i = \varsigma(x_i)b_i^{i\in 1..n-\{j\}}]}[\text{Upd}]$$

Indefinido: $[a = \varsigma(x)x.a].a$

13.3.1. Codificacion de funciones

$$
\begin{aligned}
[[x]] &\stackrel{def}{=} x \\
[[M\ N]] &\stackrel{def}{=} [[M]].arg := \ [[N]] \\
[[\lambda x.M]] &\stackrel{def}{=} [val = \varsigma(y)[[M]]\{x \leftarrow y.arg\},\ arg = \varsigma(y)y.arg]
\end{aligned}
$$

14. Resolución

14.1. Lógica propocisional

$$\frac{C_1 = \{A_1, \ldots, A_m, L\} \qquad C_2 = \{B_1, \ldots, B_m, \overline{L}\}}{C = \{A_1, \ldots, A_m, B_1, \ldots, B_n\}}$$

14.2. Lógica de primer orden

Transformar la formula:

1. Eliminar las implicaciones, es decir, si aparece una clausula de la forma $(A \supset B)$, reescribirla como $(\neg A \vee B)$.
2. Pasar a **forma normal negada**.
3. Pasar a **forma normal prenexa**.
4. Pasar a **forma normal de Skolem**.
5. Pasar a **forma normal conjuntiva**.
6. **Distribuir** cuantificadores universales.

14.2.1. Skolemización

Sea A una sentencia rectificada en forma normal negada, la **forma normal de Skolem de A** (**SK(A)**) se define recursivamente como sigue:

Sea A' cualquier subfórmula de A,

- Si A' es una fórmula atómica o su negación, $\mathbf{SK}(A') = A'$.
- Si A' es de la forma $(B \star C)$ con $\star \in \{\wedge, \vee\}$, entonces $\mathbf{SK}(A') = (\mathbf{SK}(B) \star \mathbf{SK}(C))$.
- Si A' es de la forma $\forall x.B$, entonces $\mathbf{SK}(A') = \forall x.\mathbf{SK}(B)$.
- Si A' es de la forma $\exists x.B$ y $\{x, y_1, \ldots, y_m\}$ son las variables libres de B, entonces:
  1. Si $m > 0$, crear un **símbolo de función de Skolem**, f_x de aridad m y definir:

  $$\mathbf{SK}(A') = \mathbf{SK}(B\{x \leftarrow f(y_1, \ldots, y_m)\})$$

  2. Si $m = 0$, crear una nueva **constante de Skolem** c_x y

  $$\mathbf{SK}(A') = \mathbf{SK}(B\{x \leftarrow c_x\})$$

14.3. Reglas de resolucion de primer orden

$$\frac{\{B_1, \ldots, B_k, A_1, \ldots, A_n\} \qquad \{\neg D_1, \ldots, \neg D_k, A_1, \ldots, A_n\}}{\sigma(\{A_1, \ldots A_m, C_1, \ldots, C_n\})}$$

donde σ es el **unificador más general** (MGU) de $\{B_1, \ldots, B_k, \neg D_1, \ldots, \neg D_k\}$ y $\sigma(\{A_1, \ldots A_m, C_1, \ldots, C_n\})$ es el **resolvente**.

14.4. Regla de resolucion binaria y factorizacion

$$\frac{\{B_1, A_1, \ldots, A_n\} \quad \{\neg D_1, A_1, \ldots, A_n\}}{\sigma(\{A_1, \ldots A_m, C_1, \ldots, C_n\})}$$

$$\frac{\{B_1, \ldots, B_k, A_1, \ldots, A_n\}}{\sigma(\{B_1, A_1, \ldots A_m\})}$$

15. Prolog predicados

Predicados: =, sort, msort, length, nth1, nth0, member, append, last, between, is_list, list_to_set, is_set, union, intersection, subset, subtract, select, delete, reverse, atom, number, numlist, sumlist, flatten, help

Operaciones extra-lógicas : is, \ =, ==, =:=, = \ =, >, <, =<, >=, abs, max, min, gcd, var, nonvar, ground, trace, notrace

Metapredicados: bagof, setof, maplist, include, not, forall, assert, retract, listing