**Ejercicio 2 ★**

Asumiendo que tenemos los tipos básicos, Top, funciones y registros (sin referencias ni otras extensiones)...

a) ¿Cuáles son los tipos que tienen infinitos subtipos? $\forall \sigma : \sigma <: \text{Top}$ (definición) $\forall$ registros $r : r <: \{\}$

b) ¿Cuáles son los tipos que tienen infinitos supertipos? NO HAY ?

**Ejercicio 4 ★**

Decidir si cada uno de los siguientes enunciados es verdadero o falso. Si es verdadero demostrarlo y si es falso dar un contraejemplo.

a) $T <: S$ si y solo si existe un $A$ tal que $S \to T <: A \to A$. V

b) $\{x: S, y: T\}$ siempre tiene menos supertipos que $S \to T$. F

c) $\{x: S, y: T\}$ nunca tiene menos supertipos que $S \to T$. V (siempre podés agregar un registro hasta quedar)

a)

$\exists A / \; S \to T <: A \to A \iff T <: A \;\; y \;\; A <: S \iff T <: A <: S \overset{A=S}{\iff} T <: \overset{\checkmark \text{REFL}}{S <: S} \iff T <: S$ ✓

b) $S = \text{Top}$ , $T = \text{Top}$

Veamos la supertipo de $\text{Top} \to \text{Top}$

$\text{Top} \to \text{Top} <: S' \to T' \iff S' <: \text{Top} <: T' \iff T' = \text{Top}$

$S' =$ cualquier tipo

Veamos la supertipo de $\{x: \text{Top}, y: \text{Top}\}$

$\{x: \text{Top}, y: \text{Top}\} <: \{x: \text{Top}, y: \text{Top}, z: S', w: T'\} \iff S' =$ cualquier tipo

$T' =$ cualquier tipo

se puede demostrar que los $S', T'$ de abajo forman un $\infty$ más grande que los de arriba.

**Ejercicio 5 ★**

Exhibir tipos $R$, $S$ y $T$ tales que $R <: S$ y $R <: T$ pero no exista una relación de subtipado entre $S$ y $T$.

$S = \{x: \text{Bool}\}$ , $T = \{y: \text{Bool}\}$

$R = [x: \text{Bool}, y: \text{Bool}]$

**Ejercicio 8 ★**

Probar que los siguientes términos tipan si se tienen en cuenta las reglas de subtipado vistas en clase.

a) $\lambda x: Bool.(\lambda y: Nat.\text{suc}(y))\; x$

b) $(\lambda r: \{l_1: Bool,\ l_2: Float\}.\text{if } r.l_1 \text{ then } r.l_2 \text{ else } 5{,}5)\ \{l_1 = \text{true},\ l_2 = -8,\ l_3 = 9{,}0\}$

a)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset \vartriangleright \lambda x: \text{Bool}.(\lambda y: \text{Nat}.\text{suc}(y))\; x : \text{Bool} \to \text{Nat}}\,\text{T-ABS}}{\Gamma = \{x: \text{Bool}\} \vartriangleright (\lambda y: \text{Nat}.\text{suc}(y))\; x : \text{Nat}}\,\text{T-APP}}{}$$

$$\dfrac{\dfrac{\dfrac{\Gamma \vartriangleright \lambda y: \text{Nat}.\text{suc}(y) : \text{Nat} \to \text{Nat}}{\Gamma' = \Gamma, [y: \text{Nat}] \vartriangleright \text{suc}(y) : \text{Nat}}\,\text{T-ABS}}{}\,\text{T-SUC}}{}$$

$$\dfrac{\dfrac{\Gamma' \vartriangleright y : \text{Nat}}{y: \text{Nat} \in \Gamma'}\,\text{T-VAR}}{}$$

$$\dfrac{\dfrac{\Gamma \vartriangleright x : \text{Bool}}{\dfrac{\Gamma \vartriangleright x: \text{Bool}}{x: \text{Bool} \in \Gamma}\,\text{T-VAR} \quad \dfrac{\text{Bool} <: \text{Nat}}{\text{S-Bool Nat}}}\,\text{T-SUBS}}{}$$

b)

$$\dfrac{\dfrac{\dfrac{}{\emptyset \rhd (\lambda r:\{l_1:B, l_2:F\}.\ \text{if } r.l_1 \text{ then } r.l_2 \text{ else } 5,5)\ \{l_1=\text{true}, l_2=-8, l_3=9.0\} : F}\ \text{T-APP}}{\emptyset \rhd (\lambda r:\{l_1:B, l_2:F\}.\ \text{if } r.l_1 \text{ then } r.l_2 \text{ else } 5,5):\{l_1:B, l_2:F\}\to F} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset \rhd \text{true}:\text{Bool}}\ \text{T-TRUE} \quad \dfrac{}{\emptyset \rhd -8:\text{Int}} \quad \dfrac{}{\emptyset \rhd 9.0:F}}{\emptyset \rhd \{l_1=\text{true}, l_2=-8, l_3=9.0\}:\{l_1:B, l_2:I, l_3:F\}}\ \text{T-RCD} \quad \dfrac{\{l_1,l_2,l_3\}\subseteq\{l_1,l_2\} \quad B<:B}{\{l_1:B, l_2:I, l_3:F\}\leqslant\{l_1:B, l_2:F\}}\ \text{S-RCD}}{\emptyset \rhd \{l_1=\text{true}, l_2=-8, l_3=9.0\}:\{l_1:B, l_2:F\}}\ \text{T-SUBS} \quad \dfrac{}{I<:F}}{\emptyset \rhd (\lambda r:\{l_1:B, l_2:F\}.\ \text{if } r.l_1 \text{ then } r.l_2 \text{ else } 5,5):\{l_1:B, l_2:F\}\to F}\ \text{T-ABS}$$

$$\dfrac{\dfrac{}{\Gamma \rhd r.l_1 : \text{Bool}}\ \dfrac{}{l_1:\text{Bool}} \quad \dfrac{}{\Gamma \rhd 5,5:F} \quad \dfrac{}{\Gamma \rhd r.l_2:F}\ \dfrac{}{l_2:F}}{\Gamma = \{r:\{l_1:B, l_2:F\}\} \rhd \text{if } r.l_1 \text{ then } r.l_2 \text{ else } 5,5 : F}\ \text{T-IF}$$

**Ejercicio 12 ★**

Supongamos que agregamos al lenguaje el tipo $Comp_\sigma$, para representar *comparadores* de términos de tipo $\sigma$. Los comparadores tienen la operación mejorSegún, que indica si el primer término es mejor que el segundo.

$$\frac{\Gamma \rhd M : Comp_\sigma \quad \Gamma \rhd N : \sigma \quad \Gamma \rhd O : \sigma}{\Gamma \rhd \mathsf{mejorSegún}(M, N, O) : Bool}\ \text{(T-Comp)}$$

a) El siguiente término: $\lambda c\colon Comp_{\{x:Int\}}.\mathsf{mejorSegún}(c, \{x=1, y=2\}, \{x=0\})$

¿Debería ser tipable, en términos del principio de sustitutividad? ¿Lo es? En caso afirmativo, dar una derivación que lo pruebe. Pueden asumirse como axiomas:

$$\Gamma \rhd \{x=1, y=2\} : \{x : Int, y : Int\} \qquad \Gamma \rhd \{x=0\} : \{x : Int\}$$

b) Dar la o las reglas de subtipado para comparadores.

c) El siguiente término: $\lambda c\colon Comp_{Float}.(\lambda x\colon Comp_{Nat}.\mathsf{mejorSegún}(x, 3, 4))\ c$

¿Debería ser tipable, en términos del principio de sustitutividad? ¿Según las reglas dadas, lo es? En caso afirmativo, dar una derivación que lo pruebe. Pueden asumirse como axiomas:

$$\Gamma \rhd 3 : Nat \qquad \Gamma \rhd 4 : Nat$$

a)

$$\dfrac{\dfrac{\dfrac{}{\emptyset \rhd \lambda c:Comp_{\{x:Int\}}.\ \text{mejorSegún}(c,\{x=1,y=2\},\{x=0\}) : Comp_{\{x:I\}}\to \text{Bool}}\ \text{T-ABS}}{\Gamma = \{c:Comp_{\{x:Int\}}\} \rhd \text{mejorSegún}(c,\{x=1,y=2\},\{x=0\}) : \text{Bool}}\ \text{T-COMP}$$

$$\dfrac{}{\Gamma \rhd c : Comp_{\{x:I\}}} \qquad \dfrac{\dfrac{}{\Gamma \rhd \{x=1,y=2\}:\{x:I\}}}{\Gamma \rhd \{x=1,y=2\}:\{x:I, y:I\}}\ \{x:I, y:I\}<:\{x:I\}\ \text{T-SUBS} \qquad \dfrac{}{\Gamma \rhd \{x=0\}:\{x:I\}}$$

te amo mucho! ♡M!

b)

$$\frac{\gamma <: \sigma}{Comp_\sigma <: Comp_\gamma}$$

$$\dfrac{\dfrac{\dfrac{}{\emptyset \rhd \lambda c:Comp_I.\ \text{mejorSegún}(c,\text{true},\text{false}) : Comp_B\to \text{Bool}}\ \text{T-ABS}}{\Gamma = \{c:Comp_I\} \rhd \text{mejorSegún}(c,\text{true},\text{false}) : \text{Bool}}\ \text{T-COMP}$$

$$\dfrac{\dfrac{\Gamma \rhd c : C_B}{\Gamma \rhd c : C_I}\ \dfrac{C_I <: C_B}{B<:I}}{} \qquad \dfrac{}{\Gamma \rhd \text{true} : B} \qquad \dfrac{}{\Gamma \rhd \text{false} : B}$$

c)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset \rhd \lambda c:C_F.(\lambda x:C_N.\ MS(x,3,4))\ c : C_F\to B}\ \text{T-ABS}}{\Gamma = \{c:C_F\} \rhd (\lambda x:C_N.\ MS(x,3,4))\ c : B}\ \text{T-APP}}{\Gamma \rhd \lambda x:C_N.\ MS(x,3,4) : C_N\to B}\ \text{T-ABS} \qquad \dfrac{\dfrac{\Gamma \rhd c : C_N}{\Gamma \rhd c : C_F}\ \text{T-VAR}}{C_F <: C_N}\ \text{S-COMP}}{N<:F}\ \text{S-TRAN}$$

$\Gamma' = \Gamma, \{x:C_N\} \rhd MS(x,3,4)$

⋮

$$\frac{}{N<:I}\qquad \frac{}{I<:F}$$

FALE FACIL

**Ejercicio 16**

Obviando algunos casos patológicos, un compilador de C++ se puede usar para compilar programas escritos en C. En general, algunos lenguajes de programación son subconjuntos de otros. Por ejemplo, todas las expresiones del cálculo-λ básico son a su vez expresiones del cálculo-λ extendido con referencias (aunque lo inverso no es cierto).

Sea un sistema de tipos donde $\mathsf{Prog}_{\mathcal{L}}$ representa el de los programas escritos en el lenguaje de programación $\mathcal{L}$. Se asume dada la relación de subtipado $Prog_x <: Prog_y$, verdadera cuando todo programa escrito en el lenguaje $x$ también es un programa válido en el lenguaje $y$. Este sistema de tipos permite, por ejemplo, derivar juicios como:

$$\mathsf{Prog}_{\mathsf{C}} <: \mathsf{Prog}_{\mathsf{C++}} \qquad \text{o} \qquad \mathsf{Prog}_{\text{c-}\lambda\text{ básico}} <: \mathsf{Prog}_{\text{c-}\lambda\text{ con referencias}}$$

Sea $\mathsf{Comp}(\mathcal{L}_F, \mathcal{L}_O, \mathcal{L}_I)$ el tipo de aquellos compiladores –recordar que los compiladores son programas– que reciben como entrada un programa escrito en $\mathcal{L}_F$ (lenguaje fuente) y lo traducen a $\mathcal{L}_O$ (lenguaje objeto), y además están implementados en $\mathcal{L}_I$ (lenguaje de implementación del compilador).

Contamos ya con las siguientes reglas de tipado y subtipado:

$$\frac{\Gamma \triangleright P : \mathsf{Prog}_{\mathcal{L}_F} \quad \Gamma \triangleright C : \mathsf{Comp}(\mathcal{L}_F, \mathcal{L}_O, \mathcal{L}_I)}{\Gamma \triangleright \mathsf{compilar}(C, P) : \mathsf{Prog}_{\mathcal{L}_O}}\, T - Comp \qquad \frac{}{\mathsf{Comp}(\mathcal{L}_F, \mathcal{L}_O, \mathcal{L}_I) <: \mathsf{Prog}_{\mathcal{L}_I}}\, S - CompProg$$

Interesa completar la siguiente regla de subtipado:

$$\frac{???}{\mathsf{Comp}(\mathcal{L}_F, \mathcal{L}_O, \mathcal{L}_I) <: \mathsf{Comp}(\mathcal{L}'_F, \mathcal{L}'_O, \mathcal{L}'_I)}\, S - Comp$$

Justificar la regla propuesta en términos del principio de sustitutividad.

*Sugerencia:* pensar primero cuál es la relación entre dos compiladores cuando se cambia sólo uno de los lenguajes y los demás quedan fijos. Por ejemplo, pensar cuál debería ser la relación entre $\mathcal{L}_F$ y $\mathcal{L}'_F$, fijando $\mathcal{L}_O$ y $\mathcal{L}_I$.

$$\frac{L_F' <: L_F \qquad L_O <: L_O' \qquad L_I <: L_I'}{\mathsf{Comp}(L_F, L_O, L_I) <: \mathsf{Comp}(L_F', L_O', L_I')}$$

**Ejercicio 17**

Se desea poder utilizar las proyecciones sobre pares de manera más general, sin tener que pedir nada sobre la componente que no se está proyectando. Por ejemplo, para poder evaluar la expresión $\mathsf{suc}\pi_1(M)$ solo es necesario que $M$ sea una tupla con primera componente de tipo $Nat$. No nos interesa el tipo de la segunda componente. Para esto, definimos dos nuevos tipos: $\sigma ::= \cdots \mid \times_1(\sigma) \mid \times_2(\sigma)$ donde $\times_1(\sigma)$ es el tipo de los pares cuya primera componente es de tipo $\sigma$, y $\times_2(\sigma)$ es el análogo para la segunda componente.

Se modifican las reglas de tipado paras las proyecciones de la siguiente manera:

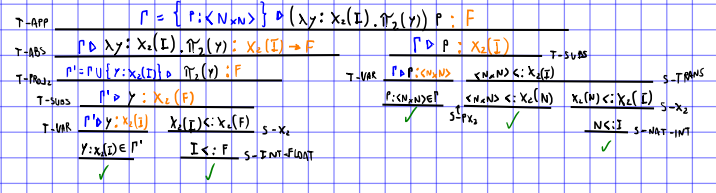
$$\frac{\Gamma \triangleright M : \times_1(\sigma)}{\Gamma \triangleright \pi_1(M) : \sigma}\, T - Proj1 \qquad \frac{\Gamma \triangleright M : \times_2(\tau)}{\Gamma \triangleright \pi_2(M) : \tau}\, T - Proj2$$

a) Escribir reglas de subtipado adecuadas para que la proyección funcione correctamente, relacionando los nuevos tipos entre sí y con los tipos de pares ya existentes. Justificar.

b) Utilizando las reglas de tipado y subtipado (nuevas y preexistentes), derivar el siguiente juicio de tipado: $\{p : \langle Nat \times Nat \rangle\} \triangleright (\lambda y : \times_2(Int).\pi_2(y))\, p : Float$

$$\frac{\sigma <: \sigma'}{\times_1(\sigma) <: \times_1(\sigma')}\, S\text{-}\times_1 \qquad \frac{}{\langle \sigma, \gamma \rangle <: \times_1(\sigma)}\, S\text{-}P\times_1$$

$$\frac{\sigma <: \sigma'}{\times_2(\sigma) <: \times_2(\sigma')}\, S\text{-}\times_2 \qquad \frac{}{\langle \tau, \sigma \rangle <: \times_2(\sigma)}\, S\text{-}P\times_2$$



**Ejercicio 18**

Se desea extender el cálculo lambda tipado para tener un mayor control sobre el proceso de reducción. Para esto, se introducen expresiones capaces de detener la reducción de un término, o de continuar una reducción que estaba detenida.

El conjunto de tipos será: $\sigma ::= \ldots \mid \mathtt{det}(\sigma)$ donde $\mathtt{det}(\sigma)$ es el tipo de los términos que resultan de detener la reducción de términos de tipo $\sigma$.

Al conjunto de términos se agregan $\mathtt{detener}(M)$ y $\mathtt{continuar}(M)$, y al de valores: $\mathtt{detener}(M)$.

El comportamiento de estas expresiones es el siguiente: sea $M$ un término tipable cualquiera, $\mathtt{detener}(M)$ detiene la reducción de $M$. Es decir, no reduce por más que $M$ pueda reducirse. Por otro lado, si $N$ es un término detenido, $\mathtt{continuar}(N)$ reanuda la reducción de $N$. Por ejemplo:
$\mathtt{continuar}((\lambda x\colon \mathtt{det}(Nat).\mathtt{continuar}(x))\ \mathtt{detener}(\mathsf{pred}(\mathsf{suc}(0)))) \to \mathtt{continuar}(\mathtt{detener}(\mathsf{pred}(\mathsf{suc}(0)))) \to \mathsf{pred}(\mathsf{suc}(0)) \to 0$.

Contamos con las siguientes reglas de tipado y semántica:

$$\frac{\Gamma \rhd M\colon \sigma}{\Gamma \rhd \mathtt{detener}(M)\colon \mathtt{det}(\sigma)}\ (\text{T-Det}) \qquad \frac{\Gamma \rhd M\colon \mathtt{det}(\sigma)}{\Gamma \rhd \mathtt{continuar}(M)\colon \sigma}\ (\text{T-Cont})$$

$$\frac{M \to M'}{\mathtt{continuar}(M) \to \mathtt{continuar}(M')}\ (\text{E-Cont}) \qquad \frac{}{\mathtt{continuar}(\mathtt{detener}(M)) \to M}\ (\text{E-ContDet})$$

Se desea que además, las funciones que esperan argumentos detenidos, puedan recibir argumentos del tipo correspondiente sin detener. En ese caso, en lugar de reducir el argumento hasta obtener un valor, lo detienen. Esto permite definir funciones que toman parámetros por nombre (call-by-name).

Para lograr este comportamiento, se reemplaza la regla $\nu$ (o E-App2) por las dos reglas siguientes:

$$\frac{\emptyset \rhd V\colon \sigma \to \tau \quad \emptyset \rhd M\colon \sigma \quad \neg(\emptyset \rhd V\colon \mathtt{det}(\sigma) \to \tau) \quad M \to M'}{V\ M \to V\ M'}\ (\nu') \qquad \frac{\emptyset \rhd V\colon \mathtt{det}(\sigma) \to \tau \quad \emptyset \rhd M\colon \sigma}{V\ M \to V\ \mathtt{detener}(M)}\ (\text{E-AppDet})$$

**Nota:** la regla $\nu'$ podría escribirse de manera más simple en un lenguaje sin subtipado, quitando la premisa $\neg(\emptyset \rhd V\colon \mathtt{det}(\sigma) \to \tau)$. Aquí la escribimos de esta manera para mantener el determinismo ante la posibilidad de que un mismo término tenga distintos tipos a la vez.