Práctica N° 7 - Subtipado

1. Reglas de subtipado

En esta sección, salvo que se especifique lo contrario, las letras $S, T, S_i, ...$ denotan tipos, y se considera $Bool$, Nat, Int y $Float$ como únicos tipos básicos con las relaciones dadas por:
$Bool <: Nat <: Int <: Float$.

Siempre que se hable de subtipos o supertipos a secas se hará referencia a la versión reflexiva de la relación $<:$. De lo contrario, se aclarará explícitamente que se trata de subtipos/supertipos estrictos.

Ejercicio 1

1. Dar una derivación que pruebe que $\{x : Nat, y : Nat, z : Nat\} <: \{y : Nat\}$. ¿Es única?
2. Dar al menos dos derivaciones de $\{x : Nat, y : Nat\} <: \{\ \}$

Ejercicio 2 ★

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

1. ¿Cuáles son los tipos que tienen infinitos subtipos?
2. ¿Cuáles son los tipos que tienen infinitos supertipos?

Ejercicio 3 ★

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

1. $T <: S$ si y sólo si existe un A tal que $S \to T <: A \to A$.
2. $\{x : S, y : T\}$ siempre tiene menos supertipos que $S \to T$.
3. $\{x : S, y : T\}$ nunca tiene menos supertipos que $S \to T$.

2. Subtipado en el contexto de tipado

Ejercicio 4 ★

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

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

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

Ejercicio 5

Mostrar que el término xx no es tipable en cálculo-λ clásico, pero sí es tipable al considerar las reglas de subtipado según lo visto. Exhibir contexto y tipo para este término.

Ejercicio 6 ★

En este ejercicio trabajaremos con los tipos $Bool <: Nat <: Int <: Float$, funciones y registros.

Puede asumirse que $Float$ tiene la operación $+$, que Int tiene además las operaciones pred y suc, y que $Bool$ cuenta también con la operación if, con las reglas de tipado habituales:

$$\frac{\Gamma \triangleright M : Int}{\Gamma \triangleright \mathsf{suc}(M) : Int}\ (\text{T-Suc}) \qquad \frac{\Gamma \triangleright M : Int}{\Gamma \triangleright \mathsf{pred}(M) : Int}\ (\text{T-Pred})$$

$$\frac{\Gamma \triangleright M : Float \quad \Gamma \triangleright N : Float}{\Gamma \triangleright M + N : Float}\ (\text{T-+}) \quad \frac{\Gamma \triangleright M : Bool \quad \Gamma \triangleright N : \sigma \quad \Gamma \triangleright O : \sigma}{\Gamma \triangleright \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ O : \sigma}\ (\text{T-If})$$

a) Suponer que la regla de subtipado para funciones fuera contravariante en el argumento y en el resultado, es decir:

$$\frac{S <: T \quad U <: V}{T \to V <: S \to U} \text{ (S-Arrow')}$$

Mostrar que esto no sería una buena idea:

- Dar una expresión M del cálculo lambda.
- Explicar brevemente por qué no tiene sentido evaluar M.
- Demostrar que, sin embargo, M tiene tipo.

b) Suponer ahora que la regla de subtipado para funciones fuera covariante en el argumento y en el resultado, es decir:

$$\frac{S <: T \quad U <: V}{S \to U <: T \to V} \text{ (S-Arrow'')}$$

Mostrar que esto tampoco sería una buena idea, de la misma forma que en el item anterior.

Ejercicio 7

a) Suponer que el operador Ref es covariante. Escribir un programa (en un lenguaje a elección) que arroje error/excepción a causa de esto.

b) Hacer lo mismo pero suponiendo que el operador Ref es contravariante.

Ejercicio 8 ★

Supongamos que agregamos al lenguaje el tipo $Comp_\sigma$, para representar *comparadores* de términos de tipo σ. Los comparadores tienen la operación $\mathsf{mejorSegún}$, 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 siguente 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:

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

Ejercicio 9 ★

Extenderemos las reglas habituales de subtipado sobre Top, booleanos, registros, tuplas y funciones.

1. Expresar con reglas de subtipado que el tipo de la *currificación* de una función es equivalente al tipo de dicha función. Es decir, cualquier función *currificada* mantiene el mismo tipo que la misma sin *currificar* y viceversa.

2. Dados los tipos A, B, C, A', B', si $A <: A'$ y $B <: B'$, sean

$$D = \langle \{x : A, y : A'\} \times B \rangle \to C$$

y

$$E = \{x : A'\} \to (B' \to C).$$

Probar, usando las reglas conocidas y las del punto 1, que vale alguna de las dos relaciones $D <: E$ o $E <: D$. Sugerencia: usar transitividad y alguna de la nuevas reglas.

3. Mostrar con un contraejemplo que las reglas de subtipado definidas en a) no son adecuadas en lo que respecta a la preservación de tipos por reducción. Es decir, mostrar dos términos M y M' tales que $M \to M'$, M es tipable con las nuevas reglas y M' no lo es. Pista: pensar en proyección y/o aplicación parcial.

Ejercicio 10

Considerar un tipo *Animal*, más una jerarquía de subtipos que (por brevedad) limitaremos aquí a *Vaca* y *León*, más un tipo paramétrico *AlimentoPara*(σ) que identifica valores que pueden ser ingeridos por *todos* los de tipo σ.

Además de las T-Sub y S-Trans usuales se han definido estas reglas de tipado y de subtipado:

$$\frac{}{\textit{León} <: \textit{Animal}}\ \text{S-León} \qquad \frac{}{\textit{Vaca} <: \textit{Animal}}\ \text{S-Vaca}$$

$$\frac{}{\textit{Vaca} <: \textit{AlimentoPara}(\textit{León})}\ \text{S-VacaLeón} \qquad \frac{}{\Gamma \rhd \textit{Clarabelle} : \textit{Vaca}}\ \text{T-Clara}$$

$$\frac{\Gamma \rhd M : \sigma \quad \Gamma \rhd N : \text{AlimentoPara}(\sigma)}{\Gamma \rhd \text{comer}(M, N) : \sigma}\ \text{T-Comer}$$

1. Suponer que alguien nos propone incorporar la siguiente regla de subtipado:

$$\frac{\sigma <: \sigma'}{\textit{AlimentoPara}(\sigma) <: \textit{AlimentoPara}(\sigma')}\ \text{S-Alim}$$

Argumentar que no es buena idea mostrando que, con esta regla, Clarabelle se come a sí misma.
Es decir, dar una derivación que pruebe que el término *comer(Clarabelle, Clarabelle)* resultaría tipable.

Sugerencia: notar que el cálculo pretende capturar cierta noción de cadenas alimentarias (quién come a quién). Antes de hacer cuentas, pensar *cómo* lo intenta, y por qué. Imaginar la misma idea en contextos con fauna más diversa también puede ser útil para comprender las ventajas y detectar los problemas del enfoque.

2. Informalmente, el conjunto de valores de tipo *AlimentoPara*(σ) es el caracterizado por la propiedad:

$$y \in \textit{AlimentoPara}(\sigma) \qquad \Leftrightarrow \qquad \forall x \in \sigma\, .\, x \text{ puede comer } y$$

Sin embargo, el problema que ilustra el punto a) es que la regla permite que las vacas coman alimento para leones. Proponer una nueva versión de S-Alim. Explicar brevemente por qué tiene más sentido que la original.

Ejercicio 11 ★

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:

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

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:

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

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 $\mathcal{L}_I$.

Ejercicio 12

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 $\texttt{Succ}(\pi_1(M))$ 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 σ, y $\times_2(\sigma)$ es el análogo para la segunda componente.

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

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

1. 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.
2. Utilizando las reglas de tipado y subtipado (nuevas y preexistentes), derivar el siguiente juicio de tipado: $\{p : \langle Nat \times Nat\rangle\} \triangleright (\lambda y\colon \times_2(Int).\pi_2(y))\, p\colon Float$

3. Subtipado: aplicaciones concretas

Ejercicio 13

En Smalltalk se entiende como tipo de un objeto el conjunto de mensajes que responde.
Extendemos el conjunto de tipos para considerar los conjuntos de mensajes como tipos:

$$\sigma ::= \ldots \mid \{m_1, \ldots, m_n\}$$

donde cada m_i es un símbolo que representa el nombre de un mensaje.

1. Axiomatizar una relación de subtipado entre los nuevos tipos que tenga sentido respecto del principio de sustitutividad.
2. ¿Cuál es la relación entre la subclasificación de Smalltalk y la relación de subtipado definida?

Ejercicio 14 ★

Dado el siguiente código en Java (que también funcionaría en C# con los cambios sintácticos adecuados):

```
public class A {
        public void foo(Object o) {
                System.out.println("Object " + o.toString());
        }

        public void foo(Integer i) {
                System.out.println("Integer " + i.toString());
        }

        public void foo(String s) {
                System.out.println("String " + s.toString());
        }
}

public class Entry {
        public static void main(String[] args) {
                Object[] v = new Object[] { new Integer(1), "Hola", Boolean.FALSE };
                A a = new A();
                for (Object o : v) {
                        a.foo(o);
                }
        }
}
```

¿Bajo qué reglas cada una de las siguientes salidas sería la correcta? ¿Cuál es la salida que efectivamente se obtiene en Java, teniendo en cuenta que utiliza la regla S-Arrow vista en clase?

a)
```
Object 1
Object Hola
Object false
```

b)
```
Integer 1
String Hola
Object false
```