

Parseo y Generación de Código

Sistemas de tipos

Unificación

Inferencia de tipos

Licenciatura en Informática con Orientación en Desarrollo de Software
Universidad Nacional de Quilmes

Sistemas de tipos

Tipos vs. *tags*

Dos conceptos que a veces se confunden son los **tipos** y los **tags**:

Tipos (noción estática).

- ▶ Los tipos son etiquetas que se le atribuyen estáticamente a los fragmentos de un programa.
- ▶ Si a un fragmento del programa se le atribuye un tipo, esto representa alguna propiedad que el programa verifica en tiempo de ejecución.
- ▶ Los tipos tienen sentido y existencia en tiempo de compilación.

Por ejemplo, si en Haskell tenemos que `f :: Int -> Bool` sabemos que si ejecutamos `f 3` y el programa termina exitosamente, el resultado va a ser necesariamente `True` o `False`.

`Int -> Bool` es una etiqueta estática que no se manifiesta en tiempo de ejecución (típicamente no “ocupa memoria”), pero nos da una garantía sobre el comportamiento de `f`.

Tipos vs. *tags*

Tags (noción dinámica).

- ▶ Los *tags* son información auxiliar que acompaña a un dato en tiempo de ejecución para indicar de qué clase de información se trata, y distinguirla de otras posibles clases de información.
- ▶ Sirven como patrón para implementar sumas (uniones disjuntas).

Por ejemplo, en JavaScript el dato 42 y el dato "hola" ambos vienen acompañados de un *tag*. En el primer caso, el *tag* indica que se trata de un número. En el segundo caso, el *tag* indica que se trata de una cadena.

```
> typeof(1)
'number'
> typeof("hola")
'string'
```

Los *tags* existen y tienen sentido en tiempo de ejecución (típicamente “ocupan memoria”).

Tipos vs. *tags*

No todos los lenguajes tienen tipos: por ejemplo, SmallTalk es un lenguaje no tipado.

Todos los lenguajes de programación permiten implementar *tags* de una manera o de otra:

- ▶ Left 42 vs. Right "hola"
- ▶ new ValueNumber(42) vs. new ValueString("hola")
- ▶ etc.

A veces los *tags* se llaman “tipos”. No está mal usar esta nomenclatura pero es importante no confundir los dos conceptos.

En esta clase nos van a interesar los tipos en el sentido estático.

Cálculo- λ simplemente tipado

Como herramienta para estudiar tipos usaremos el cálculo- λ simplemente tipado, extendido con distintas construcciones.

El cálculo- λ simplemente tipado extendido con booleanos cuenta con los siguientes tipos y términos:

Tipos.

$$\tau ::= \text{Bool} \mid \tau \rightarrow \tau$$

Términos.

$M ::=$	x	variable
	$ \quad M \ M$	aplicación
	$ \quad \lambda x : \tau. \ M$	abstracción (función anónima)
	$ \quad \text{True}$	verdadero
	$ \quad \text{False}$	falso
	$ \quad \text{if } M \text{ then } M \text{ else } M$	condicional

Nota: las gramáticas de arriba representan la sintaxis abstracta, es decir, generan árboles.

Cálculo- λ simplemente tipado

Ejercicio.

Dibujar el árbol de sintaxis abstracta de:

$$\text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool}$$

Dibujar el árbol de sintaxis abstracta de:

$$\lambda f : \text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool}. \lambda x : \text{Bool}. \lambda y : \text{Bool}. f \ y \ x$$

Dibujar el árbol de sintaxis abstracta de:

$$\lambda x : \text{Bool} \rightarrow \text{Bool}. \text{if } x \text{ then False else } x \text{ True}$$

Nota: El último programa está sintácticamente bien formado, pero nos gustaría decir que no “tipa”.

Cálculo- λ simplemente tipado

- ▶ Las ocurrencias de una variable x que están bajo el alcance de un lambda de la forma “ $\lambda x : \tau. \dots$ ” están **ligadas**.
- ▶ Las ocurrencias de una variable x que no están bajo el alcance de ningún lambda de la forma “ $\lambda x : \tau. \dots$ ” están **libres**.

Ejercicio. Marcar ocurrencias ligadas y libres de las variables en:

$f (\lambda f : (\text{Bool} \rightarrow \text{Bool}) \rightarrow \text{Bool} \rightarrow \text{Bool}. f(\lambda x : \text{Bool}. x)x)$

Cálculo- λ simplemente tipado

Técnicamente los términos del cálculo- λ no son árboles, porque se permite el **renombrado de variables ligadas**.

Por ejemplo, los dos términos siguientes son iguales:

$$(\lambda x : \text{Bool}. \lambda y : \text{Bool}. f\ x\ y) = (\lambda y : \text{Bool}. \lambda x : \text{Bool}. f\ y\ x)$$

Los términos siguientes **no** son iguales:

$$(\lambda x : \text{Bool}. \lambda y : \text{Bool}. f\ x\ y) \neq (\lambda x : \text{Bool}. \lambda x : \text{Bool}. f\ x\ x)$$

Los términos siguientes **no** son iguales:

$$f\ x\ y \neq f\ y\ x$$

Renombraremos las variables ligadas siempre que sea necesario.

Cálculo- λ simplemente tipado

Un **sistema de tipos** es un conjunto de reglas lógicas que sirven para asignarle tipos a los términos.

Por ejemplo, una regla de tipado podría ser algo de este estilo:

$$\frac{M : \tau \rightarrow \sigma \quad N : \tau}{M N : \sigma}$$

Problema. Nos gustaría tener una regla como esta:

$$\frac{M : \tau}{\lambda x : \sigma. M : \sigma \rightarrow \tau}$$

Pero esta regla se puede abusar fácilmente.

¿Cómo le daríamos tipo a $\lambda x : \text{Bool}. x$?

Problema relacionado. ¿Qué tipo tiene x ?

Cálculo- λ simplemente tipado

Para darle tipos a las variables libres, vamos a contar con un **contexto**.

Un contexto Γ es una lista finita que le asigna tipos a algunas variables:

$$\Gamma = x_1 : \tau_1, x_2 : \tau_2, \dots, x_n : \tau_n$$

donde $n \geq 0$.

Las variables x_1, x_2, \dots, x_n son distintas entre sí.

Escribimos $\Gamma(x)$ para el tipo asociado a la variable x en Γ .

Cálculo- λ simplemente tipado

En lugar de predicar sobre afirmaciones de la forma:

$$M : \tau \quad \text{“el término } M \text{ tiene tipo } \tau\text{”}$$

el sistema de tipos predica sobre afirmaciones de la forma:

$$\Gamma \vdash M : \tau \quad \text{“el término } M \text{ tiene tipo } \tau \text{ en el contexto } \Gamma\text{”}$$

Estas afirmaciones se conocen como **juicios de tipado**. Más explícitamente, un juicio de tipado:

$$x_1 : \tau_1, x_2 : \tau_2, \dots, x_n : \tau_n \vdash M : \tau$$

representa el conocimiento de que el término M tiene tipo τ , asumiendo que las variables x_1, x_2, \dots, x_n tienen tipos $\tau_1, \tau_2, \dots, \tau_n$ respectivamente.

Cálculo- λ simplemente tipado

Estamos en condiciones de dar reglas de tipado para el cálculo- λ simplemente tipado extendido con booleanos:

$$\frac{}{\Gamma \vdash \text{True} : \text{Bool}} \text{T}_{\text{TRUE}}$$

$$\frac{}{\Gamma \vdash \text{False} : \text{Bool}} \text{T}_{\text{FALSE}}$$

$$\frac{\Gamma \vdash M : \text{Bool} \quad \Gamma \vdash N : \tau \quad \Gamma \vdash P : \tau}{\Gamma \vdash \text{if } M \text{ then } N \text{ else } P : \tau} \text{T}_{\text{IF}}$$

Cálculo- λ simplesmente tipado

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \text{TVAR}$$

$$\frac{\Gamma \vdash M : \tau \rightarrow \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma} \text{TAPP}$$

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x : \tau. M : \tau \rightarrow \sigma} \text{TLAM}$$

Cálculo- λ simplemente tipado

Ejercicio. Dar derivaciones para los siguientes juicios:

1. $x : \tau, y : \text{Bool} \rightarrow \text{Bool} \vdash \lambda x : \text{Bool}. \text{if } x \text{ then } x \text{ else } y x : \text{Bool} \rightarrow \text{Bool}$
2. $\vdash \lambda x : \tau. \lambda x : \sigma. x : \tau \rightarrow \sigma \rightarrow \sigma$
3. $\vdash \lambda f : \tau \rightarrow \sigma. \lambda g : \sigma \rightarrow \rho. \lambda x : \tau. g(f\ x) : (\tau \rightarrow \sigma) \rightarrow (\sigma \rightarrow \rho) \rightarrow \tau \rightarrow \rho$

Cálculo- λ simplemente tipado

Comentario. Los sistemas de tipos generalmente vienen acompañados de metateoría que justifica que la elección de las reglas no es *ad hoc* sino que tiene buenas propiedades.

Algunas propiedades posibles:

- ▶ Si un término tiene tipo, su ejecución no falla.
- ▶ Si un término tiene tipo, su ejecución termina.
- ▶ Si un término tiene tipo y se lo ejecuta, el resultado que se obtiene tiene el mismo tipo.
- ▶ Si un término tiene tipo, todos sus subtérminos tienen tipo.

Esto excede los contenidos de la materia.

Unificación

Motivación

¿Se le puede dar tipos a los siguientes términos?

¿Qué tipo tienen?

(Agregar decoraciones en las lambdas y elegir algún contexto Γ apropiado).

$$\lambda x. \lambda y. \lambda z. x z (y z)$$
$$x x$$

Nota. Tener un sistema de tipos con reglas como las de arriba **no** nos da un algoritmo para resolver este problema.

Motivación

Hay varias dificultades para diseñar un algoritmo de inferencia:

- ▶ Para una variable x , no es obvio cómo determinar qué tipo le corresponde.
- ▶ Se debe asignar el mismo tipo a todas las ocurrencias de la misma variable.
- ▶ Para construcciones como la aplicación $M N$ o el condicional **if** M **then** N **else** P , el algoritmo debería asegurarse de que algunos tipos coincidan. Por ejemplo, las dos ramas del **if** deben tener el mismo tipo.

Ejemplos.

$x (x \text{ True})$

if $f x$ **then** $f x$ **else** $x \text{ True}$

Motivación

El algoritmo de inferencia trabaja con tipos de los que se conoce **parcialmente** su forma. Para ello se incorporan **incógnitas** a los tipos.

Por ejemplo:

- ▶ En $x \text{ True}$ sabemos que $x : \text{Bool} \rightarrow ?1$.
- ▶ En **if** x **y then** True **else** False sabemos que $x : ?2 \rightarrow \text{Bool}$.

El algoritmo necesita resolver ecuaciones entre tipos que pueden tener incógnitas.

Por ejemplo:

- ▶ La ecuación $(?1 \rightarrow \text{Bool}) = ((\text{Bool} \rightarrow \text{Bool}) \rightarrow ?2)$ tiene solución tomando $?1 = (\text{Bool} \rightarrow \text{Bool})$ y $?2 = \text{Bool}$.
- ▶ La ecuación $(?1 \rightarrow ?1) = ((\text{Bool} \rightarrow \text{Bool}) \rightarrow ?2)$ tiene solución tomando $?1 = ?2 = (\text{Bool} \rightarrow \text{Bool})$.
- ▶ La ecuación $(?1 \rightarrow \text{Bool}) = ?1$ no tiene solución.

Algoritmo de unificación

El **algoritmo de unificación** es un método para resolver sistemas de ecuaciones entre estructuras con forma de árbol que involucran incógnitas.

Suponemos fijado un conjunto finito de constructores de tipos:

- ▶ Tipos constantes: Bool, Int,
- ▶ Constructores unarios: List, Maybe,
- ▶ Constructores binarios: $\bullet \rightarrow \bullet$, (\bullet, \bullet) , Either $\bullet \bullet$,
- ▶ ...

Los tipos son árboles formados usando incógnitas y constructores (respetando su aridad):

$$\tau ::= ?n \mid C(\tau_1, \dots, \tau_n)$$

Algoritmo de unificación

Una **sustitución** es una función:

$$\mathbb{S} : \text{Incógnita} \rightarrow \text{Tipo}$$

tal que $\mathbb{S}(?n) = ?n$ salvo para un número finito de incógnitas.

Notamos $\{?k_1 \mapsto \tau_1, \dots, ?k_n \mapsto \tau_n\}$ para la sustitución \mathbb{S} tal que $\mathbb{S}(?k_i) = \tau_i$ para $i = 1..n$ y $\mathbb{S}(?j) = ?j$ para cualquier otra incógnita.

Una sustitución \mathbb{S} se puede extender a una función:

$$\begin{aligned}\hat{\mathbb{S}} : \text{Tipo} &\rightarrow \text{Tipo} \\ \hat{\mathbb{S}}(?k) &= \mathbb{S}(?k) \\ \hat{\mathbb{S}}(C(\tau_1, \dots, \tau_n)) &= C(\hat{\mathbb{S}}(\tau_1), \dots, \hat{\mathbb{S}}(\tau_n))\end{aligned}$$

Notamos \mathbb{S} también a la función extendida.

Por ejemplo, si

$$\mathbb{S}_1 = \{?1 \mapsto \text{Bool}, ?3 \mapsto (?2 \rightarrow ?2)\}$$

calcular $\mathbb{S}_1((?1 \rightarrow \text{Bool}) \rightarrow ?3)$.

Algoritmo de unificación

Un **problema de unificación** es un conjunto finito E de ecuaciones entre tipos que pueden involucrar incógnitas:

$$E = \{\tau_1 \doteq \sigma_1, \tau_2 \doteq \sigma_2, \dots, \tau_n \doteq \sigma_n\}$$

Un **unificador** para un problema de unificación es una sustitución \mathbb{S} tal que:

$$\mathbb{S}(\tau_1) = \mathbb{S}(\sigma_1)$$

$$\mathbb{S}(\tau_2) = \mathbb{S}(\sigma_2)$$

...

$$\mathbb{S}(\tau_n) = \mathbb{S}(\sigma_n)$$

Algoritmo de unificación

En general la solución a un problema de unificación no tiene por qué ser única:

$$\{?1 \stackrel{\bullet}{=} ?2\}$$

tiene infinitos unificadores:

- ▶ $\{?1 \mapsto ?2\}$
- ▶ $\{?2 \mapsto ?1\}$
- ▶ $\{?1 \mapsto ?3, ?2 \mapsto ?3\}$
- ▶ $\{?1 \mapsto \text{Bool}, ?2 \mapsto \text{Bool}\}$
- ▶ $\{?1 \mapsto (\text{Bool} \rightarrow \text{Bool}), ?2 \mapsto (\text{Bool} \rightarrow \text{Bool})\}$
- ▶ ...

Algoritmo de unificación

Una sustitución \mathbb{S}_A es **más general** que una sustitución \mathbb{S}_B si existe una sustitución \mathbb{S}_C tal que:

$$\mathbb{S}_B = \mathbb{S}_C \circ \mathbb{S}_A$$

es decir, \mathbb{S}_B se puede recuperar a partir de \mathbb{S}_A .

Para el siguiente problema de unificación:

$$E = \{(?1 \rightarrow \text{Bool}) \stackrel{\bullet}{=} ?2\}$$

las siguientes sustituciones son unificadores:

- ▶ $\mathbb{S}_1 = \{?1 \mapsto \text{Bool}, ?2 \mapsto (\text{Bool} \rightarrow \text{Bool})\}$
- ▶ $\mathbb{S}_2 = \{?1 \mapsto \text{Int}, ?2 \mapsto (\text{Int} \rightarrow \text{Bool})\}$
- ▶ $\mathbb{S}_3 = \{?1 \mapsto ?3, ?2 \mapsto (?3 \rightarrow \text{Bool})\}$
- ▶ $\mathbb{S}_4 = \{?2 \mapsto (?1 \rightarrow \text{Bool})\}$

¿Qué relación hay entre ellas? (¿Cuál es más general que cuál?).

Algoritmo de unificación

El algoritmo de **Martelli-Montanari** es un algoritmo para resolver problemas de unificación.

Dado un problema de unificación E (conjunto de ecuaciones).

- ▶ Mientras $E \neq \emptyset$, se aplica sucesivamente alguna de las seis reglas que se detallan más adelante.
- ▶ La regla puede resultar en una falla.
- ▶ De lo contrario, la regla es de la forma $E \rightarrow_{\mathbb{S}} E'$ y la solución del problema E se reduce a resolver otro problema E' , aplicando la sustitución \mathbb{S} .

Hay dos posibilidades:

- ▶ $E = E_0 \rightarrow_{\mathbb{S}_1} E_1 \rightarrow_{\mathbb{S}_2} E_2 \rightarrow \dots \rightarrow_{\mathbb{S}_n} E_n \rightarrow_{\mathbb{S}_{n+1}} \text{falla}$
En tal caso el problema de unificación E no tiene solución.
- ▶ $E = E_0 \rightarrow_{\mathbb{S}_1} E_1 \rightarrow_{\mathbb{S}_2} E_2 \rightarrow \dots \rightarrow_{\mathbb{S}_n} E_n = \emptyset$
En tal caso el problema de unificación E tiene solución.

Algoritmo de unificación

$$\{x \stackrel{\bullet}{=} x\} \cup E \xrightarrow{\text{Delete}} E$$

$$\{C(\tau_1, \dots, \tau_n) \stackrel{\bullet}{=} C(\sigma_1, \dots, \sigma_n)\} \cup E \xrightarrow{\text{Decompose}} \{\tau_1 \stackrel{\bullet}{=} \sigma_1, \dots, \tau_n \stackrel{\bullet}{=} \sigma_n\} \cup E$$

$$\{\tau \stackrel{\bullet}{=} ?n\} \cup E \xrightarrow{\text{Orient}} \{?n \stackrel{\bullet}{=} \tau\} \cup E$$

si τ no es una incógnita

$$\{?n \stackrel{\bullet}{=} \tau\} \cup E \xrightarrow{\text{Elim}}_{\{?n \mapsto \tau\}} E' = \{?n \mapsto \tau\}(E)$$

si $?n$ no ocurre en τ

$$\{C(\tau_1, \dots, \tau_n) \stackrel{\bullet}{=} C'(\sigma_1, \dots, \sigma_m)\} \cup E \xrightarrow{\text{Clash}} \text{falla}$$

si $C \neq C'$

$$\{?n \stackrel{\bullet}{=} \tau\} \cup E \xrightarrow{\text{Occurs-Check}} \text{falla}$$

si $?n \neq \tau$
y $?n$ ocurre en τ

Algoritmo de unificación

Si un problema de unificación E tiene solución:

$$E = E_0 \rightarrow_{S_1} E_1 \rightarrow_{S_2} E_2 \rightarrow \dots \rightarrow_{S_n} E_n = \emptyset$$

un unificador para el problema se puede obtener como composición de los reemplazos que el algoritmo hace cada vez que se aplica una regla **Elim**:

$$S = S_n \circ \dots \circ S_2 \circ S_1$$

Más aún, el unificador que se obtiene es el unificador **más general** posible (salvo renombre de incógnitas).

Escribimos $\text{mgu}(E)$ para denotar el unificador más general de E .

Algoritmo de unificación

Ejercicio. Calcular unificadores más generales para los siguientes problemas de unificación:

- ▶ $\{?1 \stackrel{\bullet}{=} (?2 \rightarrow ?2), ?2 \stackrel{\bullet}{=} (?1 \rightarrow ?1)\}$
- ▶ $\{(?2 \rightarrow (?1 \rightarrow ?1)) \stackrel{\bullet}{=} ((\text{Bool} \rightarrow \text{Bool}) \rightarrow (?1 \rightarrow ?2))\}$

Inferencia de tipos

Algoritmo de inferencia de tipos

Definimos los términos **sin anotaciones de tipos**:

$U ::=$	x	variable
	$U U$	aplicación
	$\lambda x. U$	abstracción (función anónima)
	True	verdadero
	False	falso
	if U then U else U	condicional

El algoritmo de inferencia de tipos es un algoritmo recursivo que dado un término sin anotaciones U devuelve un juicio de tipado $\Gamma \vdash M : \tau$ válido, y tal que M es una variante de U decorada con anotaciones de tipos.

Escribimos $U \rightsquigarrow \Gamma \vdash M : \tau$ para denotar que el resultado de aplicar el algoritmo de inferencia sobre U es $\Gamma \vdash M : \tau$.

Algoritmo de inferencia de tipos

Constantes.

$$\frac{}{\text{True} \rightsquigarrow \emptyset \vdash \text{True} : \text{Bool}} \text{I-TRUE}$$

$$\frac{}{\text{False} \rightsquigarrow \emptyset \vdash \text{False} : \text{Bool}} \text{I-FALSE}$$

Variable.

$$\frac{?k \text{ es una inc3agnita fresca}}{x \rightsquigarrow x : ?k \vdash x : ?k} \text{I-VAR}$$

Algoritmo de inferencia de tipos

Condicional.

$$\frac{\begin{array}{l} U \rightsquigarrow \Gamma_0 \vdash M : \tau \\ V_1 \rightsquigarrow \Gamma_1 \vdash N_1 : \sigma_1 \\ V_2 \rightsquigarrow \Gamma_2 \vdash N_2 : \sigma_2 \end{array} \quad \mathbb{S} = \text{mgu} \left(\begin{array}{l} \{\tau \stackrel{\bullet}{=} \text{Bool}, \sigma_1 \stackrel{\bullet}{=} \sigma_2\} \cup \\ \{\Gamma_i(x) \stackrel{\bullet}{=} \Gamma_j(x) : i, j \in \{0, 1, 2\}, x \in \Gamma_i \cap \Gamma_j\} \end{array} \right)}{\text{if } U \text{ then } V_1 \text{ else } V_2 \rightsquigarrow \frac{\mathbb{S}(\Gamma_1 \cup \Gamma_2 \cup \Gamma_3) \vdash}{\mathbb{S}(\text{if } M \text{ then } N_1 \text{ else } N_2) : \mathbb{S}(\sigma_1)}} \text{I-IF}$$

Algoritmo de inferencia de tipos

Aplicación.

$$U \rightsquigarrow \Gamma_1 \vdash M : \tau$$

$$V \rightsquigarrow \Gamma_2 \vdash N : \sigma$$

$?k$ es una incógnita fresca

$$\frac{\mathbb{S} = \text{mgu}\{\tau \stackrel{\bullet}{=} \sigma \rightarrow ?k\} \cup \{\Gamma_1(x) \stackrel{\bullet}{=} \Gamma_2(x) : x \in \Gamma_1 \cap \Gamma_2\}}{U V \rightsquigarrow \mathbb{S}(\Gamma_1 \cup \Gamma_2 \vdash M N : ?k)} \text{I-APP}$$

Algoritmo de inferencia de tipos

Abstracción.

$$\frac{U \rightsquigarrow \Gamma \vdash M : \tau \quad \sigma = \begin{cases} \Gamma(x) & \text{si } x \in \Gamma \\ \text{una incógnita fresca } ?k & \text{si no} \end{cases}}{\lambda x. U \rightsquigarrow \Gamma \setminus \{x\} \vdash \lambda x : \sigma. M : \sigma \rightarrow \tau} \text{I-LAM}$$

Algoritmo de inferencia de tipos

- ▶ El algoritmo de inferencia le da tipo a un término U si y sólo si U es tipable.
- ▶ En tal caso, le da el tipo más general posible.
- ▶ En caso contrario, el algoritmo de inferencia falla.

Algoritmo de inferencia de tipos

Ejercicio. Aplicar el algoritmo de inferencia sobre los siguientes términos:

- ▶ $\lambda x. \lambda y. y\ x$
- ▶ **if** x **then** $f\ x\ (g\ x)$ **else** $g\ (f\ x\ x)$
- ▶ $(\lambda x. x\ x)(\lambda x. x\ x)$