Parseo y Generación de Código

26 de octubre de 2017

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.

$$\tau \quad ::= \quad \mathsf{Bool} \mid \tau \to \tau$$

Términos.

M	$::=$	x	variable
	$\mid$	$M\,M$	aplicación
	$\mid$	$\lambda x : \tau.\ M$	abstracción (función anónima)
	$\mid$	True	verdadero
	$\mid$	False	falso
	$\mid$	**if** M **then** M **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:

$$\text{Bool} \to \text{Bool} \to \text{Bool}$$

Dibujar el árbol de sintaxis abstracta de:

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

Dibujar el árbol de sintaxis abstracta de:

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

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 "$\lambda x : \tau. \ldots$." están **ligadas**.
- Las ocurrencias de una variable x que no están bajo el alcance de ningún lambda de la forma "$\lambda x : \tau. \ldots$." están **libres**.

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

$$f\,(\lambda f : (\mathsf{Bool} \to \mathsf{Bool}) \to \mathsf{Bool} \to \mathsf{Bool}.\ f(\lambda x : \mathit{Bool}.x)x)$$

Cálculo-λ simplemente tipado

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

Por ejemplo, los dos términos siguientes son iguales:

$$(\lambda x.\lambda y.x\,y) = (\lambda y.\lambda x.y\,x)$$

Los términos siguientes **no** son iguales:

$$(\lambda x.\lambda y.x\,y) \neq (\lambda x.\lambda x.x\,x)$$

Los términos siguientes **no** son iguales:

$$x\,y \neq y\,x$$

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:

$$\frac{M : \tau \to \sigma \qquad N : \tau}{M\,N : \sigma}$$

Problema. Nos gustaría tener una regla como esta:

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

Pero esta regla se puede abusar fácilmente.
¿Cómo le daríamos tipo a $\lambda x : \mathrm{Bool}.x$?

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:

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

donde $n \geq 0$.

Las variables $x_1, x_2, \ldots, x_n$ 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:

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

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

Cálculo-λ simplemente tipado

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

$$\frac{}{\Gamma \vdash \mathsf{True} : \mathsf{Bool}}\ \textsc{tTrue}$$

$$\frac{}{\Gamma \vdash \mathsf{False} : \mathsf{Bool}}\ \textsc{tFalse}$$

$$\frac{\Gamma \vdash M : \mathsf{Bool} \quad \Gamma \vdash N : \tau \quad \Gamma \vdash P : \tau}{\Gamma \vdash \textbf{if } M \textbf{ then } N \textbf{ else } P : \tau}\ \textsc{tIf}$$

Cálculo-λ simplemente tipado

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \textsc{tVar}$$

$$\frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash M\,N : \sigma}\ \textsc{tApp}$$

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x : \tau.\ M : \tau \to \sigma}\ \textsc{tLam}$$

Cálculo-λ simplemente tipado

Ejercicio. Dar derivaciones para los siguientes juicios:

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

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.

$$x\,(x\,\mathsf{True})$$

$$\textbf{if } f\,x \textbf{ then } f\,x \textbf{ else } x\,\mathsf{True}$$

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 $x\,\mathrm{True}$ sabemos que $x : \mathrm{Bool} \to ?1$.
- En **if** $x\,y$ **then** True **else** False sabemos que $x : ?2 \to \mathrm{Bool}$.

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

Por ejemplo:

- La ecuación $(?1 \to \mathrm{Bool}) = ((\mathrm{Bool} \to \mathrm{Bool}) \to ?2)$ tiene solución tomando $?1 = (\mathrm{Bool} \to \mathrm{Bool})$ y $?2 = \mathrm{Bool}$.
- La ecuación $(?1 \to ?1) = ((\mathrm{Bool} \to \mathrm{Bool}) \to ?2)$ tiene solución tomando $?1 = ?2 = (\mathrm{Bool} \to \mathrm{Bool})$.
- La ecuación $(?1 \to \mathrm{Bool}) = ?1$ 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 dado un conjunto de constructores de tipos:

- Tipos constantes: Bool, Int.
- Constructores unarios: List.
- Constructores binarios: $\rightarrow$, $(-,-)$.
- Etc.

Los tipos se forman usando incógnitas y constructores, respetando su aridad:

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

Algoritmo de unificación

Una **sustitución** es una función:

$$S : \texttt{Incógnita} \to \texttt{Tipo}$$

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

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

Una sustitución S se puede extender a una función:

$$\begin{array}{rcl} \hat{S} : \texttt{Tipo} & \to & \texttt{Tipo} \\ \hat{S}(?k) & = & S(?k) \\ \hat{S}(C(\tau_1, \ldots, \tau_n)) & = & C(\hat{S}(\tau_1), \ldots, \hat{S}(\tau_k)) \end{array}$$

Por ejemplo, si

$$S_1 = \{?1 \mapsto \text{Bool},\ ?3 \mapsto (?2 \to ?2)\}$$

calcular $\hat{S_1}((?1 \to \text{Bool}) \to ?3)$.

Algoritmo de unificación

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

$$E = \{\tau_1 \stackrel{\bullet}{=} \sigma_1, \tau_2 \stackrel{\bullet}{=} \sigma_2, \ldots, \tau_n \stackrel{\bullet}{=} \sigma_n\}$$

Un **unificador** para un problema de unificación es una sustitución S tal que:

$$\hat{S}(\tau_1) = \hat{S}(\sigma_1)$$
$$\hat{S}(\tau_2) = \hat{S}(\sigma_2)$$
$$\ldots$$
$$\hat{S}(\tau_n) = \hat{S}(\sigma_n)$$

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:

- $\{?1 \mapsto ?2\}$
- $\{?2 \mapsto ?1\}$
- $\{?1 \mapsto ?3, ?2 \mapsto ?3\}$
- $\{?1 \mapsto \mathsf{Bool}, ?2 \mapsto \mathsf{Bool}\}$
- $\{?1 \mapsto (\mathsf{Bool} \to \mathsf{Bool}), ?2 \mapsto (\mathsf{Bool} \to \mathsf{Bool})\}$
- $\ldots$

Algoritmo de unificación

Hay unificadores más generales que otros:

$$\{(?1 \to \mathsf{Bool}) \stackrel{\bullet}{=} ?2\}$$

Las sustituciones:

- $S_1 = \{?1 \mapsto \mathsf{Bool}, ?2 \mapsto (\mathsf{Bool} \to \mathsf{Bool})\}$
- $S_2 = \{?1 \mapsto ?3, ?2 \mapsto (?3 \to \mathsf{Bool})\}$

son unificadores para el problema de unificación de arriba.

Pero S_2 es **más general** que S_1 porque:

$$S_1 = \{?3 \mapsto \mathsf{Bool}\} \circ S_2$$

es decir, S_1 se puede recuperar a partir de S_2.

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 $E \neq \varnothing$, 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 $E \to_S E'$ y la solución del problema E se reduce a resolver otro problema E', aplicando la sustitución S.

Hay dos posibilidades:

- $E = E_0 \to_{S_1} E_1 \to_{S_2} E_2 \to \ldots \to_{S_n} E_n \to_{S_{n+1}}$ `falla`
  En tal caso el problema de unificación E no tiene solución.
- $E = E_0 \to_{S_1} E_1 \to_{S_2} E_2 \to \ldots \to_{S_n} E_n = \varnothing$
  En tal caso el problema de unificación E tiene solución.

Algoritmo de unificación

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

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

$$\{\tau \stackrel{\bullet}{=} ?n\} \cup E \xrightarrow{\textbf{Orient}} \{?n \stackrel{\bullet}{=} \tau\} \cup E$$
si τ no es una incógnita

$$\{?n \stackrel{\bullet}{=} \tau\} \cup E \xrightarrow[?n \mapsto \tau]{\textbf{Eliminate}} E[?n \mapsto \tau]$$
si $?n$ no ocurre en τ

$$\{C(\tau_1, \ldots, \tau_n) \stackrel{\bullet}{=} C'(\sigma_1, \ldots, \sigma_m)\} \cup E \xrightarrow{\textbf{Clash}} \texttt{falla}$$
si $C' \neq C'$

$$\{?n \stackrel{\bullet}{=} \tau\} \cup E \xrightarrow{\textbf{Occurs-Check}} \texttt{falla}$$
si $?n \neq \tau$ y $?n$ ocurre en τ

Algoritmo de unificación

Si el problema tiene solución:

$$E = E_0 \to_{s_1} E_1 \to_{s_2} E_2 \to \ldots \to_{s_n} E_n = \varnothing$$

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

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

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

Escribimos mgu(E) para denotar el unificador más general de E.

Algoritmo de unificación

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

- $\{?1 \stackrel{\bullet}{=} (?2 \to ?2), ?2 \stackrel{\bullet}{=} (?1 \to ?1)\}$
- $\{(?2 \to (?1 \to ?1)) \stackrel{\bullet}{=} ((\mathsf{Bool} \to \mathsf{Bool}) \to (?1 \to ?2))\}$

Inferencia de tipos

Algoritmo de inferencia de tipos

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

U	$::=$	x	variable
	$\mid$	$U\ U$	aplicación
	$\mid$	$\lambda x.\, U$	abstracción (función anónima)
	$\mid$	True	verdadero
	$\mid$	False	falso
	$\mid$	**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.

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

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

Variable.

$$\frac{?k \text{ es una variable de tipos fresca}}{x \rightsquigarrow x : ?k \vdash x : ?k}\ \textsc{I-VAR}$$

Algoritmo de inferencia de tipos

Condicional.

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

Algoritmo de inferencia de tipos

Aplicación.

$$\frac{\begin{array}{c} M \rightsquigarrow \Gamma_1 \vdash M' : \tau \\ N \rightsquigarrow \Gamma_2 \vdash N' : \sigma \\ ?k \text{ es una variable de tipos fresca} \\ \mathbb{S} = \mathsf{mgu}\{\tau \stackrel{\bullet}{=} \sigma \to ?k\} \cup \{\Gamma_1(x) \stackrel{\bullet}{=} \Gamma_2(x) : x \in \Gamma_1 \cap \Gamma_2\} \end{array}}{M\,N \rightsquigarrow \mathbb{S}(\Gamma_1 \cup \Gamma_2 \vdash M'\,N' : ?k)}\ \text{I-APP}$$

Algoritmo de inferencia de tipos

Abstracción.

$$\frac{M \rightsquigarrow \Gamma \vdash M' : \tau \qquad \sigma = \begin{cases} \Gamma(x) & \text{si } x \in \Gamma \\ \text{una variable fresca } ?k & \text{si no} \end{cases}}{\lambda x.\, M \rightsquigarrow \Gamma \setminus \{x\} \vdash \lambda x : \sigma.\, M' : \sigma \to \tau}\ \text{I-LAM}$$

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** $g\, (f\, x\, x)$
- $(\lambda x.\, x\, x)(\lambda x.\, x\, x)$