Teórica 5 : Inferencia de tipos

Tomás Felipe Melli

May 22, 2025

Índice

1 Introducción

1.1 Qué es la Inferencia de Tipos?

La **inferencia de tipos** es la deducción que hace el *compilador* de cierto lenguaje de programación como Haskell para saber automáticamente **el tipo de una variable o expresión** sin que nosotros tengamos que especificarlo explícitamente. Esto lo logra mediante reglas del lenguaje que le permitirán analizar el contexto entre otras cosas.

1.2 En qué consiste el problema de la inferencia de tipos ?

La idea es formalizar el problema que conlleva determinar automáticamente un tipo válido para dicho término si existe.
Contamos con términos **sin anotaciones de tipos**:

$$U ::= x \mid \lambda x.U \mid U\ U \mid True \mid False \mid if\ U\ then\ U\ else\ U$$

y términos **con anotaciones de tipos**:

$$U ::= x \mid \lambda x : \tau.U \mid U\ U \mid True \mid False \mid if\ U\ then\ U\ else\ U$$

Notamos `erase(M)` al término sin anotaciones de tipos que resulta de borrar las anotaciones de tipos de `M`
Por ejemplo : `erase(`$(\lambda x : Bool.x)$ `True) = (`$\lambda x.x$`) True`

Un término `U` sin anotaciones de tipos es **tipable** si y sólo si, existen :

- **un contexto de tipado** Γ
- **un término con anotaciones de tipos** `M`
- **un tipo** τ

tales que `erase(M) = U` y $\Gamma \vdash$ `M :` τ
Decimos entoces que **el problema de inferencia de tipos consiste en**:

- Dado un término `U`, **determinar si es tipable**
- En caso de que `U` sea tipable :
  - **Hallar un contexto** Γ, **un término** `M` y **un tipo** τ de manera que `erase(M) = U` y $\Gamma \vdash$ `M :` τ

2 Cómo resolvemos entonces el problema de inferencia de tipos ?

La idea del algortimo es **manipular tipos parcialmente conocidos**. Por ejemplo, si tenemos algo de la forma

$$x\ True$$

Sabemos que $x : Bool \to X_1$. La idea es incorporar *incógnitas* $(X_1, X_2, X_3, ...)$ para poder resolver **ecuaciones** con ellas. Donde cada una de estas incógnitas representará **un tipo**.
Por ejemplo : $(X_1 \to X_2) \stackrel{?}{=} ((Bool \to Bool) \to X_2)$ donde una solución puede ser $X_1 := (Bool \to Bool)$ y $X_2 := (Bool \to Bool)$

2.1 Algoritmo de Unificación

El algortimo de unificación básicamente lo que resuelve es **encontrar una sustitución de variables que hace que dos expresiones (términos) sean iguales**.

2.1.1 Unificación

La **unificación** es el problema de resolver sistemas de ecuaciones entre tipos con incógnitas. Suponemos fijo un **conjunto finito de constructores de tipos** :

- Constantes : `Bool, Int,...`
- Unarios : `List` $\bullet$, `Maybe` $\bullet$, ...
- Binarios : $(\bullet \to \bullet), (\bullet \times \bullet), (Either\ \bullet\ \bullet), \ldots$
- Entre otros ...

Los tipos se forman usando incógnitas y constructores :

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

2.1.2 Sustitución

Una **sustitución** es una función que a cada incógnita le asocia un tipo. La notación es :

$$S = \{X_{K_1} := \tau_1, \ldots, X_{K_n} := \tau_n\}$$

Y esto se lee : ***S** es la sustitución tal que $S(X_{K_i}) = \tau_i$ para cada $1 \leq i \leq n$ y $S(X_K) = \tau_K$ para cualquier otra incógnita*
Si τ es un tipo, escribimos $\mathbf{S}(\tau)$ para el resultado de reemplazar cada incógnita de τ por el valor que le otorga **S**.
Veamos un ejemplo : Si $\mathbf{S} = \{X_1 := Bool, X_3 := (X_2 \to X_2)\}$ entonces :

$$\mathbf{S}((X_1 \to Bool) \to X_3) = ((Bool \to Bool) \to (X_2 \to X_2))$$

2.2 Qué es un problema de unificación ?

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

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

Un **unificador para `E` es una sustitución S tal que**

$$\begin{aligned}
&\mathbf{S}(\tau_1) = S(\sigma_1)\\
&\mathbf{S}(\tau_2) = S(\sigma_2)\\
&\quad\ldots\\
&\mathbf{S}(\tau_n) = S(\sigma_n)
\end{aligned}$$

En general, la solución a un problema de unificación no es única. Ejemplo de solución con infinitas soluciones :

$$\{X_1 = X_2\}$$

En este casom tiene infinitos unificadores :

- $\{X_1 := X_2\}$
- $\{X_2 := X_1\}$
- $\{X_1 := X_3, X_2 := X_3, \}$
- $\{X_1 := Bool, X_2 := Bool\}$
- $\ldots$

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

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

Es decir, $\mathbf{S}_B$ se obtiene **instanciado variables de** $\mathbf{S}_A$. Miremos el siguiente problema de unificación:

$$E = \{(X_1 \to Bool) \stackrel{?}{=} X_2\}$$

Las siguientes sustituciones son unificadores:

- $\mathbf{S}_1 = \{X_1 := \text{Bool},\ X_2 := (\text{Bool} \to \text{Bool})\}$
- $\mathbf{S}_2 = \{X_1 := \text{Int},\ X_2 := (\text{Int} \to \text{Bool})\}$
- $\mathbf{S}_3 = \{X_1 := X_3,\ X_2 := (X_3 \to \text{Bool})\}$
- $\mathbf{S}_4 = \{X_2 := (X_1 \to \text{Bool})\}$

Cuál es la más general?

Para demostrarlo proponemos dos unificadores válidos , en este caso, $\mathbf{S}_1$ *y* $\mathbf{S}_4$:
QVQ existe **S'** tal que

$$S_1 = S' \circ S_4$$

Intentamos sustituir **S'** $= \{X_1 := Bool\}$

$$S_1 = S' \circ S_4$$
$$S_1 = \{X_1 := Bool\} \circ \{X_2 := (X_1 \to \text{Bool})\}$$
$$S_1 = \{X_1 := Bool, X_2 := (Bool \to \text{Bool})\}$$

Por tanto, $\mathbf{S}_4$ es más general, ya que es menos específica. Habría que hacer esto con todos los unificadores para probar cuál es la más abstracta (general).
Notamos **mgu(E)** (*most general unifier*) al unificador más general de `E`, **si existe**.

3 Algortimo de Unificación de Martelli-Montanari

Este algortimo es para resolver problemas de unificación de términos. Formalmente:

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

- Mientras $E \neq \emptyset$, se aplica sucesivamente alguna de las **seis** reglas que vamos a ver más adelante
- La regla puede resultar en una `falla`
- De lo contrario, la regla es de la forma $E \to_S E'$. La resolución del problema E se reduce a resolver otro problema E', aplicando la sustitución $\mathbf{S}$.

Con esto dicho, tenemos **dos escenarios posibles**

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

`Delete` $\quad \{\text{X}_n \stackrel{?}{=} X_n\} \cup E \longrightarrow E$

`Decompose` $\quad \{\text{C}(\tau_1, \ldots, \tau_n) \stackrel{?}{=} C(\sigma_1, \ldots, \sigma_n)\} \cup E \longrightarrow \{\tau_1 \stackrel{?}{=} \sigma_1, \ldots, \tau_n \stackrel{?}{=} \sigma_n\} \cup E$

`Swap` $\quad \{\tau \stackrel{?}{=} X_n\} \cup E \longrightarrow \{X_n \stackrel{?}{=} \tau\} \cup E \quad$ si τ no es una incógnita

`Elim` $\quad \{\text{X}_n \stackrel{?}{=} \tau\} \cup E \longrightarrow \{X_n := \tau\}, \quad E' = \{X_n := \tau\}(E) \quad$ si X_n no ocurre en τ

`Clash` $\quad \{\text{C}(\tau_1, \ldots, \tau_n) \stackrel{?}{=} C'(\sigma_1, \ldots, \sigma_m)\} \cup E \longrightarrow$ falla $\quad$ si $C \neq C'$

`Occurs-Check` $\quad \{\text{X}_n \stackrel{?}{=} \tau\} \cup E \longrightarrow$ falla $\quad$ si $X_n \neq \tau$ y X_n ocurre en τ

3.1 Teorema (Corrección del algoritmo)

1. El algoritmo termina para cualquier problema de unificación `E`.
2. Si `E` no tiene solución, el algoritmo llega a una `falla`
3. Si `E` tiene solución, el algoritmo llega a $\emptyset$:

   $$E = E_0 \to_{S_1} E_1 \to_{S_2} E_2 \to \ldots \to_{S_n} E_n = \emptyset$$

   Además, $\mathbf{S} = \mathbf{S}_n \circ \ldots \circ \mathbf{S}_2 \circ \mathbf{S}_1$ es un unificador para `E`. Además, este unificador es **el más general posible** (salvo renombre de incógnitas)

4 Algoritmo de Inferencia de Tipos- Algoritmo $\mathcal{I}$

El algoritmo $\mathcal{I}$ recibe un término `U` sin anotaciones de tipos. Consta de los siguientes pasos :

1. **Rectificación** del término
2. **Anotación** del término con variables de tipo *frescas*
3. **Generación de restricciones**(ecuaciones entre tipos)
4. **Unificación de las restricciones**

4.1 Paso 1 : Rectificación

Un término está **rectificado** si :

1. **No hay dos variables ligadas con el mismo nombre**
2. **No hay una variable ligada con el mismo nombre que una variable libre**

Veamos unos ejemplos de términos rectificados

- $x\ (\lambda x.\ x\ x)\ (\lambda y.\ y\ x)$ no está rectificado, no cumple 2.
- $x\ (\lambda z.\ z\ z)\ (\lambda y.\ y\ x)$ está rectificado.
- $\lambda x.\ \lambda x.\ x\ y$ no está rectificado, no cumple 1.
- $\lambda x.\ \lambda z.\ z\ y$ está rectificado.

Observación : Siempre se puede rectificar un término α`-renombrándolo`

4.2 Paso 2 : Anotación

Tenemos un término `U`, que suponemos ya rectificado. **Producimos un contexto** Γ_0 **y un término** M_0.Para ello :

- El contexto Γ_0 le da tipo a todas las variables libres de U. El tipo de cada variable es incógnita *fresca*
- El término M_0 está anotado de tal modo que `eraser(`M_0`)` $= U$. Todas las anotaciones son incógnitas frescas.

Veamos el siguiente ejemplo de anotación del término.
Dado el término rectificado $U = (\lambda x.y\ x\ x)(\lambda z.w)$ producimos :

- $\Gamma_0 = (y : X_1, w : X_2)$
- $M_0 = (\lambda x : X_3.y\ x\ x)(\lambda z : X_4.w)$

4.3 Paso 3 : Generación de las Restricciones

Tenemos un contexto Γ y un término M con anotaciones de tipos. Recursivamente calculamos :

- **Un tipo** τ**, que corresponde al tipo de** M
- **Un conjunto de ecuaciones** E. Representan restricciones para que M esté bien tipado.

Como dijimos que el cálculo es recursivo,definimos el algoritmo :

$$\underbrace{\mathcal{I}(\underbrace{\Gamma}_{contexto} \mid \underbrace{M}_{término})}_{entrada} = \underbrace{(\underbrace{\tau}_{tipo} \mid \underbrace{E}_{restricciones})}_{salida}$$

con la precondición de que Γ le da tipo a todas las variables de M. Veamos las reglas que se aplican según el término :

1. **Constante booleana** $True$:

   $$\mathcal{I}(\Gamma \mid True) = (Bool \mid \emptyset)$$

   Es decir, en cualquier contexto Γ, cuando **queremos inferir el tipo de la constante booleana** $True$, obtenemos : *tipo* : $Bool$ y ninguna *restricción adicional*(ya que el conjunto es vacío).

2. **Constante booleana** $False$:

   $$\mathcal{I}(\Gamma \mid False) = (Bool \mid \emptyset)$$

   Es análogo al anterior.

3. **Variable x**

   $$\mathcal{I}(\Gamma \mid x) = (\tau \mid \emptyset) \quad si\ x : \tau \in \Gamma$$

   Esto es, si estamos en un contexto Γ y **queremos inferir el tipo de la variable** x. Entonces, el tipo resultante debe ser aquel asociado a x en el contexto (o sea, τ), y no se generan restricciones.

4. **Condicional (if M1 then M2 else M3)**:

   $$\mathcal{I}(\Gamma \mid if\ M_1\ then M_2\ else\ M_3) = (\tau_2 \mid \{\tau_1 \stackrel{?}{=} Bool, \tau_2 \stackrel{?}{=} \tau_3\} \cup E_1 \cup E_2 \cup E_3)$$

   donde :

   $$\mathcal{I}(\Gamma \mid M_1) = (\tau_1 \mid E_1)$$
   $$\mathcal{I}(\Gamma \mid M_2) = (\tau_2 \mid E_2)$$
   $$\mathcal{I}(\Gamma \mid M_3) = (\tau_3 \mid E_3)$$

   La idea es inferir los tipos de las **tres partes**. La **condición M1** debe tener tipo $Bool$ eso genera la **restricción** $\tau_1 = Bool$. Luego, las ramas deben tener **el mismo tipo**, y por ello, surge la restricción $\tau_2 = \tau_3$. Ahora bien, el tipo del **if** será entonces aquel de las ramas τ_2 *o* τ_3(son el mismo) y el conjunto de restricciones son, las restricciones de cada subexpresión y las nuevas que impone el if.

5. **Aplicación** :

   $$\mathcal{I}(\Gamma \mid M_1\ M_2) = (X_k \mid \{\tau_1 \stackrel{?}{=} \tau_2 \to X_k\} \cup E_1 \cup E_2)$$

   donde :

   $$\mathcal{I}(\Gamma \mid M_1) = (\tau_1 \mid E_1) \quad \texttt{tipo de la funció n}$$
   $$\mathcal{I}(\Gamma \mid M_2) = (\tau_2 \mid E_2) \quad \texttt{tipo del argumento}$$

   - X_k es una **nueva variable de tipo fresca**
   - Se genera la restricción $\tau_1 = \tau_2 \to X_k$ porque la función tiene que aceptar el tipo del argumento

   En criollo, inferimos el tipo de $M1$ y obtenemos τ_1, inferimos el tipo de $M2$ y obtenemos τ_2 y para que la aplicación tenga sentido, $M1$ **debe ser una función que reciba** $M2$ **como argumento**, que es la restricción principal $\tau_1 = \tau_2 \to X_k$. Como consecuencia, el tipo de toda la aplicación es X_k

6. **Abstracción** $(\lambda x : \tau.M)$

   $$\mathcal{I}(\Gamma \mid \lambda x : \tau.M) = (\tau \to \sigma \mid E)$$

   donde :

   `Se extiende el contexto con` $x : \tau$
   $$\mathcal{I}(\Gamma, x : \tau \mid M) = (\sigma \mid E)$$

   Con esto sabemos que una **abstracción siempre tiene el tipo función**. Conocemos el tipo del parámetro τ y queremos inferir el tipo del cuerpo σ bajo el contexto extendido. Como consecuencia el tipo final es $\tau \to \sigma$. Con esto, las únicas restricciones que aparecen son las del cuerpo **M**

4.4 Paso 4 : Unificación de las restricciones

Hasta ahora tenemos : un término original U sin tipos explícitos, lo rectificamos y lo llamamos M_0, ajustamos el contexto y lo llamamos Γ_0, aplicamos el algoritmo I para obtener $\mathcal{I}(\Gamma_0 \mid M_0) = (\tau \mid E)$. O sea, τ es el tipo inferido del término con variables tipo y E el conjunto de restricciones. En este paso, queremos **resolver las restricciones** E **para obtener los tipos concretos**. Para lograrlo tenemos que :

1. Calcular el unificador más general **S = mgu(E)**
2. Si **no existe** el unificador, el término U **no es tipable**
3. Si **existe** el unificador, el término U **es tipable** y vale el siguiente juicio :

   $$\mathbf{S}(\Gamma_0) \vdash \mathbf{S}(M_0) : \mathbf{S}(\tau)$$

   Es decir, como pudimos aplicar la sustitución **S** al contexto, al término y al tipo y lograr este juicio, concluimos que el término bajo el contexto actualizado tiene un tipo concreto y ese es $\mathbf{S}(\tau)$

4.5 Corrección

El teorema de corrección del algoritmo I nos asegura que el algoritmo es correcto. Con esto :

- Si el término U **no es tipable**, es porque **realmente no se puede construir ningún tipo válido para él** ya que no hay unificador para E.
- Si el algoritmo dice que U es un **término es tipable**, entonces **existe una derivación válida en el sistema de tipos** : $\mathbf{S}(\Gamma_0) \vdash \mathbf{S}(M_0) : \mathbf{S}(\tau)$. Más aún, **el juicio de tipado es el más general posible para** U. Más precisamente, si $\Gamma' \vdash M' : \tau$ es un juicio válido y `erase(M') = U`, existe una sustitución **S'** tal que :

  $$\Gamma' \supseteq \mathbf{S'}(\Gamma_0)$$
  $$M' = \mathbf{S'}(M_0)$$
  $$\tau' = \mathbf{S'}(\tau)$$

  donde además **S** es más general que **S'**

5 Corrección del Algoritmo de Unificación

5.1 Terminación

Dado un conjunto de ecuaciones de unificación E, definimos :

- n_1 : cantidad de variables distintas en E
- n_2 : tamaño de E, calculado como $\sum_{(\tau \stackrel{?}{=} \sigma) \in E} |\tau| + |\sigma|$
- n_3 : cantidad de ecuaciones de la forma $\tau \stackrel{?}{=} x$ en E

Podemos observar que las reglas que no producen falla **achican la tripla** n_1, n_2, n_3 de acuerdo con el orden *lexicográfico*

	n_1	n_2	n_3
`Elim`	$>$		
`Decompose`	$=$	$>$	
`Delete`	$\geq$	$>$	
`Swap`	$=$	$=$	$>$

5.2 Corrección

Recordamos lo siguiente :

1. Una **sustitución** es una función **S** que le asocia un término $\mathbf{S}(x)$ a cada variable x.
2. **S** es un **unificador** de E si para cada $(\tau \stackrel{?}{=} \sigma) \in E$ se tiene $\mathbf{S}(\tau) = \mathbf{S}(\sigma)$
3. **S** es **más general** que **S'** si existe **T** tal que $\mathbf{S'} = \mathbf{T} \circ \mathbf{S}$
4. **S** es un `mgu` de E si **S** es un unificador de E y para todo unificador **S'** de E se tiene que **S** es más general que **S'**. Técnicamente, nos interesan los `mgu` **idempotentes**, es decir, $\mathbf{S}(\mathbf{S}(\tau)) = \mathbf{S}(\tau)$ para todo término τ

5.3 Lemas

5.3.1 Lema 1 - Corrección de la regla `Delete`

Básicamente esta regla nos dice que si hay una ecuación del tipo $x = x$ (trivial) la podemos quitar y no vamos a molestar a la solución

$$\mathbf{S} \texttt{ mgu de } E \implies \mathbf{S} \texttt{ mgu de } \{x \stackrel{?}{=} x\} \cup E$$

5.3.2 Lema 2 - Corrección de la regla `Swap`

La regla nos dice que podemos *"intercambiar lados"*, formalmente, la **la igualdad es simétrica**, es decir, podemos unificar $A = B$ o $B = A$ dando el mismo resultado

$$\mathbf{S}\ \texttt{mgu de}\ \{\tau \stackrel{?}{=} \sigma\} \cup E \implies \mathbf{S}\ \texttt{mgu de}\ \{\sigma \stackrel{?}{=} \tau\} \cup E$$

5.3.3 Lema 3 - Corrección de la regla `Decompose`

La regla nos dice que podemos **dividir una ecuación compleja en subecuaciones más simples**, si los constructores (funciones o tipos) coinciden. Esto vale ya que para que dos valores de la forma $C(\ldots)$ sean iguales, todos sus componentes deben ser iguales. Con esto en mente...

$$\mathbf{S}\ \texttt{mgu de}\ \{\tau_1 \stackrel{?}{=} \sigma_1, \ldots, \tau_n \stackrel{?}{=} \sigma_n\} \cup E \implies \mathbf{S}\ \texttt{mgu de}\ \{C(\tau_1, \ldots, \tau_n) \stackrel{?}{=} C(\sigma_1, \ldots, \sigma_n)\} \cup E$$

5.3.4 Lema 4 - Corrección de la regla `Elim`

Esta regla nos dice que podemos eliminar una ecuación asignando a x el valor τ y **sustituirla en todo el sistema**. Esto vale ya que si $x = \tau$ y luego unificás el resto de E con esto, no hay pérdida de información (lo que no se puede hacer es cuando **x** *aparece en* τ ahí hay quilombo porque entramos en un bucle infinito)

$$\mathbf{S}\ \texttt{mgu de}\ E\{x := \tau\}\ y\ x \notin \tau \implies \mathbf{S} \circ \{x := \tau\}\ \texttt{mgu de}\ E$$

Nos aclaran que *si después de aplicar la sustitución* **S** *la variable* x *ya queda reemplazada por* τ*, entonces sustituir* x *por* τ ***antes*** *de aplicar* ***S*** *no cambia el resultado.* Formalmente :

$$\texttt{Si}\ \mathbf{S}(x) = \tau\ \texttt{entonces}\ \mathbf{S}(\sigma\{x := \tau\}) = \mathbf{S}(\sigma)$$

5.4 Demo

Probemos la corrección del algoritmo en caso de éxito.

Sea

$$E_0 \xrightarrow{S_1} E_1 \xrightarrow{S_2} E_2 \to \ldots \xrightarrow{S_n} E_n = \emptyset.$$

Veamos que $S_n \circ \ldots \circ S_1$ es un m.g.u. de E.

Por inducción en n:

1. Si $n = 0$, la sustitución identidad es un m.g.u. de $\emptyset$.
2. Si $n > 0$, se tiene:

$$E_0 \xrightarrow{S_1} E_1, \quad E_1 \xrightarrow{S_2} \ldots \xrightarrow{S_n} E_n = \emptyset.$$

Por HI, $S_n \circ \ldots \circ S_2$ es un m.g.u. de E_1.

Aplicando alguno de los lemas anteriores, se concluye que

$$S_n \circ \ldots \circ S_2 \circ S_1$$

es un m.g.u. de E_0.