

Lambda Cálculo Tipado (3/3)

Eduardo Bonelli

Departamento de Computación, FCEyN, UBA

“There may, indeed, be other applications of the system other than its use as a logic”

Alonzo Church, 1932

13 de septiembre de 2012

Estructura de la clase

Inferencia

- Motivación

- Variables de tipo y sustituciones de tipo

- Especificación del problema

Unificación

- Motivación

- Definición y algoritmo

- Algoritmo de unificación

Algoritmo de inferencia

- Algoritmo de inferencia

- Ejemplos

Inferencia de tipos

- ▶ Problema que consiste en transformar términos **sin** información de tipos o con información de tipos **parcial** en términos **tipables**
- ▶ Para ello debe **inferirse** la información de tipos faltante
- ▶ Beneficio para lenguajes con tipos
 - ▶ el programador puede obviar algunas declaraciones de tipos
 - ▶ en general, evita la sobrecarga de tener que declarar y manipular **todos** los tipos
 - ▶ todo ello sin desmejorar la performance del programa: la inferencia de tipos se realiza en tiempo de **compilación**

Inferencia de tipos

- ▶ Inferencia de tipos es especialmente útil en lenguajes **polimórficos**
- ▶ Nosotros vamos a **restringir** nuestro estudio a inferencia en Lambda Cálculo Tipado (LC)
- ▶ Si bien LC **no** es polimórfico, basta para presentar los conceptos básicos detrás de la inferencia de tipos
- ▶ Diremos más sobre polimorfismo a la ML o Haskell al final de la clase
- ▶ Algunos nombres importantes en la historia de la inferencia de tipos: Curry, Feys, Hindley, Milner

El problema de la inferencia de tipos

Primero modificamos la sintaxis de los términos de LC **eliminando** toda anotación de tipos

$$\begin{array}{l} M ::= x \\ \quad | \text{ true } | \text{ false } | \text{ if } M \text{ then } P \text{ else } Q \\ \quad | 0 | \text{ succ}(M) | \text{ pred}(M) | \text{ iszero}(M) \\ \quad | \lambda x : \sigma . M | M N | \\ \quad | \text{ fix } M \end{array}$$

El problema de la inferencia de tipos

Primero modificamos la sintaxis de los términos de LC **eliminando** toda anotación de tipos

$$\begin{array}{l} M ::= x \\ \quad | \text{ true } | \text{ false } | \text{ if } M \text{ then } P \text{ else } Q \\ \quad | 0 | \text{ succ}(M) | \text{ pred}(M) | \text{ iszero}(M) \\ \quad | \lambda x.M | M N | \\ \quad | \text{ fix } M \end{array}$$

Función de borrado

Llamaremos $\text{ERASE}(\cdot)$ a la función que dado un término de LC **elimina** las anotaciones de tipos de las abstracciones

$$\text{ERASE}(\lambda x : \text{Nat}. \lambda f : \text{Nat} \rightarrow \text{Nat}. f\ x) = \lambda x. \lambda f. f\ x$$

El problema de la inferencia - Definición

Dado un término U sin anotaciones de tipo, hallar un término estándar (i.e. con anotaciones de tipos) M tal que

1. $\Gamma \triangleright M : \sigma$, para algún Γ y σ , y
2. $\text{ERASE}(M) = U$

Ejemplos

- Para $U = \lambda x.x + 5$ tomamos $M = \lambda x : \text{Nat}.x + 5$ (observar que no hay otra posibilidad)

El problema de la inferencia - Definición

Dado un término U sin anotaciones de tipo, hallar un término estándar (i.e. con anotaciones de tipos) M tal que

1. $\Gamma \triangleright M : \sigma$, para algún Γ y σ , y
2. $\text{ERASE}(M) = U$

Ejemplos

- ▶ Para $U = \lambda x. x + 5$ tomamos $M = \lambda x : \text{Nat}. x + 5$ (observar que no hay otra posibilidad)
- ▶ Para $U = \lambda x. \lambda f. f\ x$ tomamos $M_{\sigma, \tau} = \lambda x : \sigma. \lambda f : \sigma \rightarrow \tau. f\ x$ (hay un $M_{\sigma, \tau}$ por cada σ, τ)

El problema de la inferencia - Definición

Dado un término U sin anotaciones de tipo, hallar un término estándar (i.e. con anotaciones de tipos) M tal que

1. $\Gamma \triangleright M : \sigma$, para algún Γ y σ , y
2. $\text{ERASE}(M) = U$

Ejemplos

- ▶ Para $U = \lambda x. x + 5$ tomamos $M = \lambda x : \text{Nat}. x + 5$ (observar que no hay otra posibilidad)
- ▶ Para $U = \lambda x. \lambda f. f\ x$ tomamos $M_{\sigma, \tau} = \lambda x : \sigma. \lambda f : \sigma \rightarrow \tau. f\ x$ (hay un $M_{\sigma, \tau}$ por cada σ, τ)
- ▶ Para $U = \lambda x. \lambda f. f\ (f\ x)$ tomamos $M_{\sigma} = \lambda x : \sigma. \lambda f : \sigma \rightarrow \sigma. f\ (f\ x)$ (hay un M_{σ} por cada σ)

El problema de la inferencia - Definición

Dado un término U sin anotaciones de tipo, hallar un término estándar (i.e. con anotaciones de tipos) M tal que

1. $\Gamma \triangleright M : \sigma$, para algún Γ y σ , y
2. $\text{ERASE}(M) = U$

Ejemplos

- ▶ Para $U = \lambda x. x + 5$ tomamos $M = \lambda x : \text{Nat}. x + 5$ (observar que no hay otra posibilidad)
- ▶ Para $U = \lambda x. \lambda f. f\ x$ tomamos $M_{\sigma, \tau} = \lambda x : \sigma. \lambda f : \sigma \rightarrow \tau. f\ x$ (hay un $M_{\sigma, \tau}$ por cada σ, τ)
- ▶ Para $U = \lambda x. \lambda f. f\ (f\ x)$ tomamos $M_{\sigma} = \lambda x : \sigma. \lambda f : \sigma \rightarrow \sigma. f\ (f\ x)$ (hay un M_{σ} por cada σ)
- ▶ Para $U = xx$ no existe ningún M con la propiedad deseada

El problema del chequeo de tipos

chequeo de tipos \neq inferencia de tipos

Chequeo de tipos

Dado un término **estándar** M determinar si existe Γ y σ tales que $\Gamma \triangleright M : \sigma$ es derivable.

- ▶ Es mucho más **fácil** que el problema de la inferencia
- ▶ Consiste simplemente en seguir la **estructura sintáctica** de M para reconstruir una derivación del juicio
- ▶ Es esencialmente equivalente a determinar, **dados** Γ y σ , si $\Gamma \triangleright M : \sigma$ es derivable.

Variables de tipo

- ▶ Dado $\lambda x. \lambda f. f (f x)$, para cada σ ,
 $M_\sigma = \lambda x : \sigma. \lambda f : \sigma \rightarrow \sigma. f (f x)$ es una solución posible
- ▶ ¿De qué manera podemos escribir una **única** expresión que englobe a todas ellas? Usando **variables de tipo**
 - ▶ Todas las soluciones se pueden representar con
$$\lambda x : s. \lambda f : s \rightarrow s. f (f x)$$
 - ▶ “s” es una variable de tipos que representa una expresión de tipos arbitraria
 - ▶ Si bien esta expresión **no** es una solución en sí misma, la **sustitución** de s por cualquier expresión de tipos **sí** arroja una solución válida

Variables de tipo

- ▶ Extendemos las expresiones de tipo de LC con variables de tipo s, t, u, \dots

$$\sigma ::= s \mid Nat \mid Bool \mid \sigma \rightarrow \tau$$

Ejemplos

- ▶ $s \rightarrow t$
- ▶ $Nat \rightarrow Nat \rightarrow t$
- ▶ $Bool \rightarrow t$

Sustitución de tipos (o simplemente sustitución)

Función que mapea variables de tipo en expresiones de tipo.
Usamos letrs S , T , etc para sustituciones.

- ▶ Una sustitución S puede aplicarse a
 1. una expresión de tipos σ (escribimos $S\sigma$)
 2. un término M (escribimos SM)
 3. un contexto de tipado $\Gamma = \{x_1 : \sigma_1, \dots, x_n : \sigma_n\}$ (escribimos $S\Gamma$ y lo definimos como sigue)

$$S\Gamma \stackrel{\text{def}}{=} \{x_1 : S\sigma_1, \dots, x_n : S\sigma_n\}$$

Sustitución - Nociones adicionales

- ▶ El conjunto $\{t \mid St \neq t\}$ se llama **soporte** de S
- ▶ El soporte representa las variables que S “afecta”
- ▶ Usamos la notación $\{\sigma_1/t_1, \dots, \sigma_n/t_n\}$ para la sustitución con soporte $\{t_1, \dots, t_n\}$ definida de la manera obvia
- ▶ La sustitución cuyo soporte es \emptyset es la **sustitución identidad** (Id)

Instancia de un juicio de tipado

Un juicio de tipado $\Gamma' \triangleright M' : \sigma'$ es una **instancia** de $\Gamma \triangleright M : \sigma$ si existe una sustitución de tipos S tal que

$$\Gamma' \supseteq S\Gamma, M' = SM \text{ y } \sigma' = S\sigma$$

Propiedad

Si $\Gamma \triangleright M : \sigma$ es derivable, entonces cualquier instancia del mismo también

Función de Inferencia $\mathbb{W}(\cdot)$

Definir una función $\mathbb{W}(\cdot)$ que dado un término U **sin anotaciones** verifica

Corrección $\mathbb{W}(U) = \Gamma \triangleright M : \sigma$ implica

- ▶ $\text{ERASE}(M) = U$ y
- ▶ $\Gamma \triangleright M : \sigma$ es derivable

Compleitud Si $\Gamma \triangleright M : \sigma$ es derivable y $\text{ERASE}(M) = U$, entonces

- ▶ $\mathbb{W}(U)$ tiene éxito y
- ▶ produce un juicio $\Gamma' \triangleright M' : \sigma'$ tal que $\Gamma \triangleright M : \sigma$ es instancia del mismo (se dice que $\mathbb{W}(\cdot)$ computa un **tipo principal**)

Inferencia

Unificación

Motivación

Definición y algoritmo

Algoritmo de unificación

Algoritmo de inferencia

Unificación

- ▶ El algoritmo de inferencia analiza un término (sin anotaciones de tipo) a partir de sus subtérminos
- ▶ Una vez obtenida la información inferida para cada uno de los subtérminos debe
 1. (**Consistencia**) Determinar si la información de cada subtérmino es consistente
 2. (**Síntesis**) Sintetizar la información del término original a partir de la información de sus subtérminos

Ejemplo

Consideremos el término $x\ y + x\ (y + 1)$

- ▶ Del análisis de $x\ y$ surge que $x :: s \rightarrow t$ e $y :: s$
- ▶ Del análisis de $x\ (y + 1)$ surge que $x :: Nat \rightarrow u$ e $y :: Nat$
- ▶ Dado que una variable puede tener un sólo tipo debemos **compatibilizar** la información de tipos
 - ▶ El tipo $s \rightarrow t$ debe ser **compatible** o **unificable** con $Nat \rightarrow u$ dado que ambos se refieren a x
 - ▶ El tipo s debe ser **compatible** o **unificable** con Nat dado que ambos se refieren a y

Unificación

- ▶ ¿El tipo $s \rightarrow t$ es compatible o unificable con $Nat \rightarrow u$? Sí
 - ▶ Basta tomar la sustitución $S \stackrel{\text{def}}{=} \{Nat/s, u/t\}$
 - ▶ Y observar que $S(s \rightarrow t) = Nat \rightarrow u = S(Nat \rightarrow u)$
- ▶ ¿El tipo s es compatible o unificable con Nat ? Sí
 - ▶ La sustitución antedicha es tal que $Ss = SNat$

El proceso de determinar si existe una sustitución S tal que dos expresiones de tipos σ, τ son unificables (ie. $S\sigma = S\tau$) se llama **unificación**

- ▶ Vamos a estudiar con precisión a la unificación, repasando antes algunos conceptos básicos sobre sustituciones

Composición de sustituciones

La **composición** de S y T , denotada $S \circ T$, es la sustitución que se comporta como sigue:

$$(S \circ T)(\sigma) = S(T\sigma)$$

Ejemplo

Sea $S = \{u \rightarrow \text{Bool}/t, \text{Nat}/s\}$ y $T = \{v \times \text{Nat}/u, \text{Nat}/s\}$, entonces $T \circ S = \{(v \times \text{Nat}) \rightarrow \text{Bool}/t, v \times \text{Nat}/u, \text{Nat}/s\}$

- ▶ Decimos que $S = T$ si tienen el mismo soporte y $St = Tt$ para todo t en el soporte de S
- ▶ $S \circ Id = Id \circ S = S$
- ▶ $S \circ (T \circ U) = (S \circ T) \circ U$

Preorden sobre sustituciones

Una sustitución S es **más general** que T si existe U tal que $T = U \circ S$.

- ▶ La idea es que S es más general que T porque T se obtiene instanciando S

Unificador

Una **ecuación de unificación** es una expresión de la forma $\sigma_1 \doteq \sigma_2$.
Una sustitución S es una **solución** de un conjunto de ecuaciones de unificación $\{\sigma_1 \doteq \sigma'_1, \dots, \sigma_n \doteq \sigma'_n\}$ si $S\sigma_1 = S\sigma'_1 = \dots = S\sigma'_n$

Ejemplos

- ▶ La sust. $\{Bool/v, Bool \times Nat/u\}$ es solución de $\{v \times Nat \rightarrow Nat \doteq u \rightarrow Nat\}$
- ▶ $\{Bool \times Bool/v, (Bool \times Bool) \times Nat/u\}$ también!
- ▶ $\{v \times Nat/u\}$ también!
- ▶ $\{Nat \rightarrow s \doteq t \times u\}$ **no** tiene solución
- ▶ $\{u \rightarrow Nat \doteq u\}$ **no** tiene solución

Unificador más general (MGU)

Una sustitución S es un **MGU** de $\{\sigma_1 \doteq \sigma'_1, \dots, \sigma_n \doteq \sigma'_n\}$ si

1. es solución de $\{\sigma_1 \doteq \sigma'_1, \dots, \sigma_n \doteq \sigma'_n\}$
2. es más general que cualquier otra solución de $\{\sigma_1 \doteq \sigma'_1, \dots, \sigma_n \doteq \sigma'_n\}$

Ejemplos

- ▶ La sust. $\{Bool/v, Bool \times Nat/u\}$ es solución de $\{v \times Nat \rightarrow Nat \doteq u \rightarrow Nat\}$ pero no es un MGU pues es instancia de la solución $\{v \times Nat/u\}$
- ▶ $\{v \times Nat/u\}$ es un MGU del conjunto

Algoritmo de unificación

Teorema

Si $\{\sigma_1 \doteq \sigma'_1, \dots, \sigma_n \doteq \sigma'_n\}$ tiene solución, existe un MGU y además es único salvo renombre de variables

- ▶ Entrada:

- ▶ Conjunto de ecuaciones de unificación $\{\sigma_1 \doteq \sigma'_1, \dots, \sigma_n \doteq \sigma'_n\}$

- ▶ Salida:

- ▶ **MGU** S de $\{\sigma_1 \doteq \sigma'_1, \dots, \sigma_n \doteq \sigma'_n\}$, si tiene solución

- ▶ **falla**, en caso contrario

Algoritmo de Martelli-Montanari

- ▶ Vamos a presentar un algoritmo no-determinístico
- ▶ Consiste en **reglas de simplificación** que simplifican conjuntos de pares de tipos a unificar (*goals*)

$$G_0 \mapsto G_1 \mapsto \dots \mapsto G_n$$

- ▶ Las secuencias que terminan en el goal vacío son **exitosas**; aquellas que terminan en **falla** son **fallidas**
- ▶ Algunas pasos de simplificación llevan un sustitución que representa una solución parcial al problema

$$G_0 \mapsto G_1 \mapsto_{S_1} G_2 \mapsto \dots \mapsto_{S_k} G_n$$

- ▶ Si la secuencia es exitosa el MGU es $S_k \circ \dots \circ S_1$

Reglas del algoritmo de Martelli-Montanari

1. Descomposición

$$\{\sigma_1 \rightarrow \sigma_2 \doteq \tau_1 \rightarrow \tau_2\} \cup G \mapsto \{\sigma_1 \doteq \tau_1, \sigma_2 \doteq \tau_2\} \cup G$$

$$\{Nat \doteq Nat\} \cup G \mapsto G$$

$$\{Bool \doteq Bool\} \cup G \mapsto G$$

2. Eliminación de par trivial

$$\{s \doteq s\} \cup G \mapsto G$$

3. Swap: si σ no es una variable

$$\{\sigma \doteq s\} \cup G \mapsto \{s \doteq \sigma\} \cup G$$

4. Eliminación de variable: si $s \notin FV(\sigma)$

$$\{s \doteq \sigma\} \cup G \mapsto_{\sigma/s} G[\sigma/s]$$

5. Falla

$$\{\sigma \doteq \tau\} \cup G \mapsto \text{falla}, \text{ con } (\sigma, \tau) \in T \cup T^{-1} \text{ y}$$

$$T = \{(Bool, Nat), (Nat, \sigma_1 \rightarrow \sigma_2), (Bool, \sigma_1 \rightarrow \sigma_1)\}$$

6. Occur check: si $s \neq \sigma$ y $s \in FV(\sigma)$

$$\{s \doteq \sigma\} \cup G \mapsto \text{falla}$$

Ejemplo de secuencia exitosa

$$\begin{array}{ll} \vdash^1 & \{(Nat \rightarrow r) \rightarrow (r \rightarrow u) \doteq t \rightarrow (s \rightarrow s) \rightarrow t\} \\ \vdash^3 & \{Nat \rightarrow r \doteq t, r \rightarrow u \doteq (s \rightarrow s) \rightarrow t\} \\ \vdash^4_{Nat \rightarrow r/t} & \{t \doteq Nat \rightarrow r, r \rightarrow u \doteq (s \rightarrow s) \rightarrow t\} \\ \vdash^1 & \{r \rightarrow u \doteq (s \rightarrow s) \rightarrow (Nat \rightarrow r)\} \\ \vdash^4_{s \rightarrow s/r} & \{r \doteq s \rightarrow s, u \doteq Nat \rightarrow r\} \\ \vdash^4_{Nat \rightarrow (s \rightarrow s)/u} & \{u \doteq Nat \rightarrow (s \rightarrow s)\} \\ & \emptyset \end{array}$$

- El MGU es $\{Nat \rightarrow (s \rightarrow s)/u\} \circ \{s \rightarrow s/r\} \circ \{Nat \rightarrow r/t\} = \{Nat \rightarrow (s \rightarrow s)/t, s \rightarrow s/r, Nat \rightarrow (s \rightarrow s)/u\}$

Ejemplo de secuencia fallida

$$\begin{array}{ll} & \{r \rightarrow (s \rightarrow r) \doteq s \rightarrow ((r \rightarrow \text{Nat}) \rightarrow r)\} \\ \mapsto^1 & \{r \doteq s, s \rightarrow r \doteq (r \rightarrow \text{Nat}) \rightarrow r\} \\ \mapsto^4_{s/r} & \{s \rightarrow s \doteq (s \rightarrow \text{Nat}) \rightarrow s\} \\ \mapsto^1 & \{s \doteq s \rightarrow \text{Nat}, s \doteq s\} \\ \mapsto^6 & \text{falla} \end{array}$$

Propiedades del algoritmo

Teorema

- ▶ El algoritmo de Martelli-Montanari siempre termina
- ▶ Sea G un conjunto de pares G
 - ▶ si G tiene un unificador, el algoritmo termina exitosamente y retorna un MGU
 - ▶ si G no tiene unificador, el algoritmo termina con **falla**

Inferencia

Unificación

Algoritmo de inferencia

Algoritmo de inferencia

Ejemplos

Algoritmo de inferencia

- ▶ Vamos a presentar un algoritmo de inferencia para LC
- ▶ El objetivo es definir $\mathbb{W}(U)$ por recursión sobre la estructura de U
- ▶ Primero presentamos la cláusulas que definen a $\mathbb{W}(U)$ sobre las constantes y las variables, luego pasamos a las demás construcciones
- ▶ Utilizaremos el algoritmo de unificación

Algoritmo de inferencia (caso constantes y variables)

$$\begin{aligned}\mathbb{W}(0) &\stackrel{\text{def}}{=} \emptyset \triangleright 0 : \textit{Nat} \\ \mathbb{W}(\textit{true}) &\stackrel{\text{def}}{=} \emptyset \triangleright \textit{true} : \textit{Bool} \\ \mathbb{W}(\textit{false}) &\stackrel{\text{def}}{=} \emptyset \triangleright \textit{false} : \textit{Bool} \\ \mathbb{W}(x) &\stackrel{\text{def}}{=} \{x : s\} \triangleright x : s, \quad s \text{ variable fresca}\end{aligned}$$

Algoritmo de inferencia (caso *succ*)

- ▶ Sea $\mathbb{W}(U) = \Gamma \triangleright M : \tau$
- ▶ Sea $S = MGU\{\tau \doteq Nat\}$
- ▶ Entonces

$$\mathbb{W}(\textcolor{red}{succ}(U)) \stackrel{\text{def}}{=} S\Gamma \triangleright S \textcolor{red}{succ}(M) : Nat$$

- ▶ Nota: Caso *pred* es similar

Algoritmo de inferencia (caso *iszero*)

- ▶ Sea $\mathbb{W}(U) = \Gamma \triangleright M : \tau$
- ▶ Sea $S = MGU\{\tau \doteq Nat\}$
- ▶ Entonces

$$\mathbb{W}(\textcolor{red}{iszero}(\textcolor{red}{U})) \stackrel{\text{def}}{=} S\Gamma \triangleright S \textcolor{blue}{iszero}(M) : \textcolor{blue}{Bool}$$

Algoritmo de inferencia (caso ifThenElse)

► Sea

- $\mathbb{W}(U) = \Gamma_1 \triangleright M : \rho$
- $\mathbb{W}(V) = \Gamma_2 \triangleright P : \sigma$
- $\mathbb{W}(W) = \Gamma_3 \triangleright Q : \tau$

► Sea

$$S = MGU\{\sigma_1 \doteq \sigma_2 \mid x : \sigma_1 \in \Gamma_i \wedge x : \sigma_2 \in \Gamma_j, i \neq j\} \\ \cup \\ \{\sigma \doteq \tau, \rho \doteq Bool\}$$

► Entonces

$$\mathbb{W}(\text{if } U \text{ then } V \text{ else } W) \\ \stackrel{\text{def}}{=} S\Gamma_1 \cup S\Gamma_2 \cup S\Gamma_3 \triangleright S(\text{if } M \text{ then } P \text{ else } Q) : S\sigma$$

Algoritmo de inferencia (caso aplicación)

- ▶ Sea

- ▶ $\mathbb{W}(U) = \Gamma_1 \triangleright M : \tau$

- ▶ $\mathbb{W}(V) = \Gamma_2 \triangleright N : \rho$

- ▶ Sea

$$S = \text{MGU}\{\sigma_1 \doteq \sigma_2 \mid x : \sigma_1 \in \Gamma_1 \wedge x : \sigma_2 \in \Gamma_2\} \cup$$

$\{\tau \doteq \rho \rightarrow t\}$ con t una variable fresca

- ▶ Entonces

$$\mathbb{W}(UV) \stackrel{\text{def}}{=} S\Gamma_1 \cup S\Gamma_2 \triangleright S(MN) : St$$

Algoritmo de inferencia (caso abstracción)

- ▶ Sea $\mathbb{W}(U) = \Gamma \triangleright M : \rho$
- ▶ Si el contexto tiene información de tipos para x (i.e. $x : \tau \in \Gamma$ para algún τ), entonces

$$\mathbb{W}(\lambda x. U) \stackrel{\text{def}}{=} \Gamma \setminus \{x : \tau\} \triangleright \lambda x : \tau. M : \tau \rightarrow \rho$$

- ▶ Si el contexto no tiene información de tipos para x (i.e. $x \notin \text{Dom}(\Gamma)$) elegimos una variable fresca s y entonces

$$\mathbb{W}(\lambda x. U) \stackrel{\text{def}}{=} \Gamma \triangleright \lambda x : s. M : s \rightarrow \rho$$

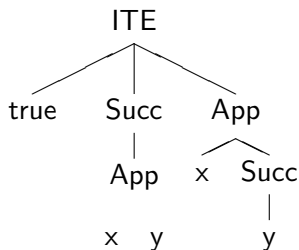
Algoritmo de inferencia (caso *fix*)

- ▶ Sea $\mathbb{W}(U) = \Gamma \triangleright M : \tau$
- ▶ Sea $S = MGU\{\tau \doteq t \rightarrow t\}$, t variable fresca

$$\mathbb{W}(\text{fix}(U)) \stackrel{\text{def}}{=} S\Gamma \triangleright S \text{fix}(M) : St$$

Ejemplo

- ▶ Vamos a mostrar cómo inferir el tipo de *if true then succ(x y) else x (succ(y))*
- ▶ Aplicaremos el algoritmo, paso por paso



Ejemplo (1/4)

*if **true** then succ(x y) else x (succ(y))*

$$\mathbb{W}(\text{true}) = \emptyset \triangleright \text{true} : \text{Bool}$$

Ejemplo (2/4)

if true then succ(x y) else x (succ(y))

$$\mathbb{W}(x) = \{x : s\} \triangleright x : s$$

$$\mathbb{W}(y) = \{y : t\} \triangleright y : t$$

$$\mathbb{W}(x y) = \{x : t \rightarrow r, y : t\} \triangleright x y : r$$

donde $S = MGU(\{s \doteq t \rightarrow r\}) = \{t \rightarrow r/s\}$

$$\mathbb{W}(\text{succ}(x y)) = \{x : t \rightarrow \text{Nat}, y : t\} \triangleright \text{succ}(x y) : \text{Nat}$$

donde $S = MGU(\{r \doteq \text{Nat}\}) = \{\text{Nat}/r\}$

Ejemplo (3/4)

if true then succ(x y) else $x(\text{succ}(y))$

$$\mathbb{W}(y) = \{y : v\} \triangleright y : v$$

$$\mathbb{W}(\text{succ}(y)) = \{y : \text{Nat}\} \triangleright \text{succ}(y) : \text{Nat}$$

$$\text{donde } S = \text{MGU}(\{v \doteq \text{Nat}\}) = \{\text{Nat}/v\}$$

$$\mathbb{W}(x) = \{x : u\} \triangleright x : u$$

$$\mathbb{W}(x \text{ succ}(y)) = \{x : \text{Nat} \rightarrow w, y : \text{Nat}\} \triangleright x \text{ succ}(y) : w$$

$$\text{donde } \text{MGU}(\{u \doteq \text{Nat} \rightarrow w\}) = \{\text{Nat} \rightarrow w/u\}$$

Ejemplo (4/4)

$$M = \text{if } \text{true} \text{ then } \text{succ}(x\ y) \text{ else } x\ (\text{succ}(y))$$

- ▶ $\mathbb{W}(\text{true}) = \emptyset \triangleright \text{true} : \text{Bool}$
- ▶ $\mathbb{W}(\text{succ}(x\ y)) = \{x : t \rightarrow \text{Nat}, y : t\} \triangleright \text{succ}(x\ y) : \text{Nat}$
- ▶ $\mathbb{W}(x\ \text{succ}(y)) = \{x : \text{Nat} \rightarrow w, y : \text{Nat}\} \triangleright x\ \text{succ}(y) : w$

$$\mathbb{W}(M) = \{x : \text{Nat} \rightarrow \text{Nat}, y : \text{Nat}\} \triangleright M : \text{Nat}$$

$$\begin{aligned} \text{donde } S &= \text{MGU}(\{t \rightarrow \text{Nat} \doteq \text{Nat} \rightarrow w, t \doteq \text{Nat}, \text{Nat} \doteq w\}) = \\ &= \{\text{Nat}/t, \text{Nat}/w\} \end{aligned}$$

Un ejemplo de falla

$M = \text{if } \text{true} \text{ then } x \underline{2} \text{ else } x \text{ true}$

$$\mathbb{W}(x) = \{x : s\} \triangleright x : s$$

$$\mathbb{W}(\underline{2}) = \emptyset \triangleright \underline{2} : \text{Nat}$$

$$\mathbb{W}(x \underline{2}) = \{x : \text{Nat} \rightarrow t\} \triangleright x \underline{2} : t$$

$$\mathbb{W}(x) = \{x : u\} \triangleright x : u$$

$$\mathbb{W}(\text{true}) = \emptyset \triangleright \text{true} : \text{Bool}$$

$$\mathbb{W}(x \text{ true}) = \{x : \text{Bool} \rightarrow v\} \triangleright x \underline{2} : v$$

$$\mathbb{W}(M) = \text{falla}$$

no existe el $MGU(\{\text{Nat} \rightarrow t \doteq \text{Bool} \rightarrow v\})$

Complejidad

- ▶ Tanto la unificación como la inferencia para LC se puede hacer en **tiempo lineal**
- ▶ El tipo principal asociado a un término sin anotaciones puede ser **exponencial** en el tamaño del término

Considerar inferir el tipo de $P^n M$ con $P : s \rightarrow s \times s$ y $M : \sigma$

- ▶ ¿Esto no contradice lo antedicho?
- ▶ **No**. Se pueden representar usando dags en cuyo caso el tamaño del tipo principal de U será $O(n)$
- ▶ **NB**: En la presencia de polimorfismo la inferencia es exponencial

Let-Polymorphism

- ▶ Los lenguajes funcionales como ML, Haskell, etc. permiten tipos polimórficos de la forma

$$\forall s_1 \dots s_n. \sigma \text{ (}\sigma \text{ sin cuantificadores)}$$

- ▶ Este tipo de polimorfismo restringido se llama **predicativo**
- ▶ En particular no se pueden definir funciones que tomen a otras funciones **polimórficas** como argumento

Let-Polymorphism

```
Prelude> (\f-> (f True, f 3)) (\x -> 5)
ERROR - Illegal Haskell 98 class constraint in inferred type
*** Expression : (\f -> (f True,f 3)) (\x -> 5)
*** Type       : Num Bool => (Integer,Integer)
```

```
Prelude> (\f-> (f True, f 3)) id
ERROR - Illegal Haskell 98 class constraint in inferred type
*** Expression : (\f -> (f True,f 3)) id
*** Type       : Num Bool => (Bool,Bool)
```

Let-Polymorphism

- ▶ Para poder declarar y usar funciones polimórficas se introduce la construcción `let`

```
Prelude> let g = \x->5 in (g True, g 3)  
(5,5)
```

- ▶ Polimorfismo predicativo con declaraciones `let` polimórficas forman el núcleo (básico) del sistema de tipos de ML y Haskell
- ▶ La inferencia de tipos para este sistema es muy similar a aquella vista hoy
- ▶ Para más detalles consultar capítulo 11 del texto de Mitchell o capítulo 22 del texto de Pierce