Lambda Cálculo Tipado (3/3)

Eduardo Bonelli / Alejandro Ríos

Departamento de Computación, FCEyN, UBA

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

Alonzo Church, 1932

19 de abril de 2012

Estructura de la clase

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 à 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

$$
\begin{array}{rcl}
M & ::= & x \\
 & | & \mathit{true} \mid \mathit{false} \mid \mathit{if}\ M\ \mathit{then}\ P\ \mathit{else}\ Q \\
 & | & 0 \mid \mathit{succ}(M) \mid \mathit{pred}(M) \mid \mathit{iszero}(M) \\
 & | & \lambda x : \sigma.M \mid M\, N \mid \\
 & | & \mathit{fix}\, M
\end{array}
$$

Denotamos este conjunto de términos con $\Lambda_{\mathcal{T}}$

El problema de la inferencia de tipos

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

$$
\begin{array}{rcl}
M & ::= & x \\
 & | & \mathit{true} \mid \mathit{false} \mid \mathit{if}\ \ M\ \ \mathit{then}\ \ P\ \ \mathit{else}\ \ Q \\
 & | & \mathit{0} \mid \mathit{succ}(M) \mid \mathit{pred}(M) \mid \mathit{iszero}(M) \\
 & | & \lambda x.M \mid M\,N \mid \\
 & | & \mathit{fix}\,M
\end{array}
$$

Denotamos este conjunto de términos con Λ

Función de borrado

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

$\textsc{Erase} : \Lambda_{\mathcal{T}} \longrightarrow \Lambda$ se define de la manera esperada.

Ejemplo

$\textsc{Erase}(\lambda x : \mathit{Nat}.\lambda f : \mathit{Nat} \to \mathit{Nat}.f\ x) = \lambda x.\lambda f.f\ x$

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. $\Gamma \rhd M : \sigma$, para algún Γ y σ, y
2. $\textsc{Erase}(M) = U$

Ejemplos

- Para $U = \lambda x.x + 5$ tomamos $M = \lambda x : Nat.x + 5$ (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. $\textsc{Erase}(M) = U$

Ejemplos

- Para $U = \lambda x.x + 5$ tomamos $M = \lambda x : Nat.x + 5$ (observar que no hay otra posibilidad)
- Para $U = \lambda x.\lambda f.f\,x$ tomamos $M_{\sigma,\tau} = \lambda x : \sigma.\lambda f : \sigma \to \tau.f\,x$ (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. $\textsc{Erase}(M) = U$

Ejemplos

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

Ejemplos

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

Variables de tipo

- Dado $\lambda x.\lambda f.f\,(f\,x)$, para cada σ,
  $M_\sigma = \lambda x : \sigma.\lambda f : \sigma \to \sigma.f\,(f\,x)$ es un 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
    $$\lambda x : s.\lambda f : s \to s.f\,(f\,x)$$
  - "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 $s, t, u, \ldots$

$$\sigma \quad ::= \quad s \mid Nat \mid Bool \mid \sigma \to \tau$$

- Denotamos con $\mathcal{V}$ al conjunto de variables de tipo
- Denotamos con $\mathcal{T}$ al conjunto de tipos así definidos

Ejemplos

- $s \to t$
- $Nat \to Nat \to t$
- $Bool \to t$

Sustitución de tipos (o simplemente sustitución)

- Función que mapea variables de tipo en expresiones de tipo.

  Usamos S, T, etc. para sustituciones.

  Formalmente, $S : \mathcal{V} \longrightarrow \mathcal{T}$

  Sólo nos interesan las S tales que $\{t \in \mathcal{V} \mid St \neq t\}$ es finito.

Sustitución de tipos (o simplemente sustitución)

- Función que mapea variables de tipo en expresiones de tipo.

  Usamos S, T, etc. para sustituciones.

  Formalmente, $S : \mathcal{V} \longrightarrow \mathcal{T}$

  Sólo nos interesan las S tales que $\{t \in \mathcal{V} \mid St \neq t\}$ es finito.

- Una sustitución S puede aplicarse (de manera natural) a

Sustitución de tipos (o simplemente sustitución)

- Función que mapea variables de tipo en expresiones de tipo.

  Usamos S, T, etc. para sustituciones.

  Formalmente, $S : \mathcal{V} \longrightarrow \mathcal{T}$

  Sólo nos interesan las S tales que $\{t \in \mathcal{V} \mid St \neq t\}$ es finito.

- Una sustitución S puede aplicarse (de manera natural) a
  1. una expresión de tipos σ (escribimos $S\sigma$)

Sustitución de tipos (o simplemente sustitución)

- Función que mapea variables de tipo en expresiones de tipo.

  Usamos S, T, etc. para sustituciones.

  Formalmente, $S : \mathcal{V} \longrightarrow \mathcal{T}$

  Sólo nos interesan las S tales que $\{t \in \mathcal{V} \mid St \neq t\}$ es finito.

- Una sustitución S puede aplicarse (de manera natural) a
  1. una expresión de tipos σ (escribimos $S\sigma$)
  2. un término M (escribimos SM)

Sustitución de tipos (o simplemente sustitución)

- Función que mapea variables de tipo en expresiones de tipo.

  Usamos S, T, etc. para sustituciones.

  Formalmente, $S : \mathcal{V} \longrightarrow \mathcal{T}$

  Sólo nos interesan las S tales que $\{t \in \mathcal{V} \mid St \neq t\}$ es finito.

- Una sustitución S puede aplicarse (de manera natural) 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 $\{\sigma_1/t_1, \ldots, \sigma_n/t_n\}$ para la sustitución con soporte $\{t_1, \ldots, t_n\}$ definida de la manera obvia
- La sustitución cuyo soporte es $\emptyset$ es la sustitución identidad y la notamos Id

Instancia de un juicio de tipado

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

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

Propiedad

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

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

- $\textsc{Erase}(M) = U$ y
- $\Gamma \triangleright M : \sigma$ es derivable

Completitud Si $\Gamma \triangleright M : \sigma$ es derivable y $\textsc{Erase}(M) = U$, 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)

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 $x :: s \to t$ e $y :: s$
- Del análisis de $x\,(y+1)$ surge que $x :: Nat \to u$ e $y :: Nat$
- Dado que una variable puede tener un sólo tipo debemos compatibilizar la información de tipos
  - El tipo $s \to t$ debe ser compatible o unificable con $Nat \to u$ 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 $s \to t$ es compatible o unificable con $Nat \to u$? Sí
  - Basta tomar la sustitución $S \stackrel{\text{def}}{=} \{Nat/s, u/t\}$
  - Y observar que $S(s \to t) = Nat \to u = S(Nat \to u)$
- ¿El tipo s es compatible o unificable con Nat? Sí
  - La sustitución antedicha es tal que $Ss = S\,Nat$

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 el proceso de 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 $S = \{u \to Bool/t, Nat/s\}$ y $T = \{v \times Nat/u, Nat/s\}$, entonces $T \circ S = \{(v \times Nat) \to Bool/t, v \times Nat/u, Nat/s\}$

- 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 $\{\sigma_1 \doteq \sigma'_1, \ldots, \sigma_n \doteq \sigma'_n\}$ si $S\sigma_1 = S\sigma'_1, \ldots, S\sigma_n = S\sigma'_n$

Ejemplos

- La sustitución $\{Bool/v, Bool \times Nat/u\}$ es solución de $\{v \times Nat \to Nat \doteq u \to Nat\}$
- $\{Bool \times Bool/v, (Bool \times Bool) \times Nat/u\}$ también!
- $\{v \times Nat/u\}$ también!
- $\{Nat \to s \doteq t \times u\}$ no tiene solución
- $\{u \to Nat \doteq u\}$ no tiene solución

Unificador más general (MGU)

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

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

Ejemplos

- La sustitución $\{Bool/v, Bool \times Nat/u\}$ es solución de $\{v \times Nat \to Nat \doteq u \to Nat\}$ 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 $\{\sigma_1 \doteq \sigma'_1, \ldots, \sigma_n \doteq \sigma'_n\}$ tiene solución, existe un MGU y además es único salvo renombre de variables

- Entrada:
  - Conjunto de ecuaciones de unificación $\{\sigma_1 \doteq \sigma'_1, \ldots, \sigma_n \doteq \sigma'_n\}$
- Salida:
  - MGU S de $\{\sigma_1 \doteq \sigma'_1, \ldots, \sigma_n \doteq \sigma'_n\}$, 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*)

$$G_0 \mapsto G_1 \mapsto \ldots \mapsto G_n$$

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

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

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

Reglas del algoritmo de Martelli-Montanari

1. **Descomposición**
   $\{\sigma_1 \to \sigma_2 \doteq \tau_1 \to \tau_2\} \cup G \mapsto \{\sigma_1 \doteq \tau_1, \sigma_2 \doteq \tau_2\} \cup G$
   $\{Nat \doteq Nat\} \cup G \mapsto G$
   $\{Bool \doteq Bool\} \cup G \mapsto G$
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 $s \notin FV(\sigma)$
   $\{s \doteq \sigma\} \cup G \mapsto_{\{\sigma/s\}} \{\sigma/s\}G$
5. **Falla**
   $\{\sigma \doteq \tau\} \cup G \mapsto \texttt{falla}$, con $(\sigma, \tau) \in T \cup T^{-1}$ y
   $T = \{(Bool, Nat), (Nat, \sigma_1 \to \sigma_2), (Bool, \sigma_1 \to \sigma_1)\}$
6. **Occur check**: si $s \neq \sigma$ y $s \in FV(\sigma)$
   $\{s \doteq \sigma\} \cup G \mapsto \texttt{falla}$

Ejemplo de secuencia exitosa

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

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

Ejemplo de secuencia fallida

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

Propiedades del algoritmo

Teorema

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

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 $\mathbb{W}(.)$ sobre las constantes y las variables, luego pasamos a las demás construcciones
- Utilizaremos el algoritmo de unificación

Algoritmo de inferencia (caso constantes y variables)

$$
\begin{array}{rcl}
\mathbb{W}(\mathit{0}) & \stackrel{\mathrm{def}}{=} & \emptyset \rhd \mathit{0} : \mathit{Nat} \\
\mathbb{W}(\mathit{true}) & \stackrel{\mathrm{def}}{=} & \emptyset \rhd \mathit{true} : \mathit{Bool} \\
\mathbb{W}(\mathit{false}) & \stackrel{\mathrm{def}}{=} & \emptyset \rhd \mathit{false} : \mathit{Bool} \\
\mathbb{W}(x) & \stackrel{\mathrm{def}}{=} & \{x : s\} \rhd x : s, \quad s \text{ variable fresca}
\end{array}
$$

Algoritmo de inferencia (caso *succ*)

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

$$\mathbb{W}(succ(U)) \quad \stackrel{\mathrm{def}}{=} \quad S\Gamma \rhd S\, succ(M) : Nat$$

- 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

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

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

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

Algoritmo de inferencia (caso aplicación)

- Sea
  - $\mathbb{W}(U) \quad = \quad \Gamma_1 \rhd M : \tau$
  - $\mathbb{W}(V) \quad = \quad \Gamma_2 \rhd N : \rho$
- Sea

$$S = MGU(\{\sigma_1 \doteq \sigma_2 \mid x : \sigma_1 \in \Gamma_1 \wedge x : \sigma_2 \in \Gamma_2\} \\ \cup \\ \{\tau \doteq \rho \to t\}) \quad \text{con } t \text{ una variable fresca}$$

- Entonces

$$\mathbb{W}(U\,V) \quad \stackrel{\text{def}}{=} \quad S\Gamma_1 \cup S\Gamma_2 \rhd S(M\,N) : St$$

Algoritmo de inferencia (caso abstracción)

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

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

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

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

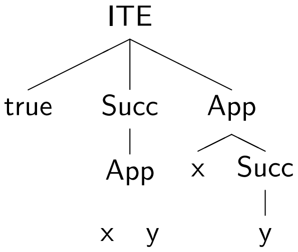
Algoritmo de inferencia (caso *fix*)

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

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

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)

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

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

Ejemplo (2/4)

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

$$\begin{array}{rcl}
\mathbb{W}(x) & = & \{x : s\} \triangleright x : s \\
\mathbb{W}(y) & = & \{y : t\} \triangleright y : t \\
\mathbb{W}(x\,y) & = & \{x : t \to r, y : t\} \triangleright x\,y : r \\
 & & \text{donde } S = MGU(\{s \doteq t \to r\}) = \{t \to r/s\} \\
\mathbb{W}(\mathit{succ}(x\,y)) & = & \{x : t \to Nat, y : t\} \triangleright \mathit{succ}(x\,y) : Nat \\
 & & \text{donde } S = MGU(\{r \doteq Nat\}) = \{Nat/r\}
\end{array}$$

Ejemplo (3/4)

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

$$\begin{array}{rcl}
\mathbb{W}(y) & = & \{y : v\} \rhd y : v \\
\mathbb{W}(\mathit{succ}(y)) & = & \{y : \mathit{Nat}\} \rhd \mathit{succ}(y) : \mathit{Nat} \\
 & & \text{donde } S = \mathit{MGU}(\{v \doteq \mathit{Nat}\}) = \{\mathit{Nat}/v\} \\
\mathbb{W}(x) & = & \{x : u\} \rhd x : u \\
\mathbb{W}(x\,\mathit{succ}(y)) & = & \{x : \mathit{Nat} \to w, y : \mathit{Nat}\} \rhd x\,\mathit{succ}(y) : w \\
 & & \text{donde } \mathit{MGU}(\{u \doteq \mathit{Nat} \to w\}) = \{\mathit{Nat} \to w/u\}
\end{array}$$

Ejemplo (4/4)

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

- $\mathbb{W}(\mathit{true}) = \emptyset \rhd \mathit{true} : \mathit{Bool}$
- $\mathbb{W}(\mathit{succ}(x\,y)) = \{x : t \to \mathit{Nat}, y : t\} \rhd \mathit{succ}(x\,y) : \mathit{Nat}$
- $\mathbb{W}(x\,\mathit{succ}(y)) = \{x : \mathit{Nat} \to w, y : \mathit{Nat}\} \rhd x\,\mathit{succ}(y) : w$

$$\mathbb{W}(M) \quad = \quad \{x : \mathit{Nat} \to \mathit{Nat}, y : \mathit{Nat}\} \rhd M : \mathit{Nat}$$

donde $S = MGU(\{t \to \mathit{Nat} \doteq \mathit{Nat} \to w,\ t \doteq \mathit{Nat}, \mathit{Nat} \doteq w\}) =$
$= \{\mathit{Nat}/t, \mathit{Nat}/w\}$

Un ejemplo de falla

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

$$\begin{array}{rcl}
\mathbb{W}(x) & = & \{x : s\} \rhd x : s \\
\mathbb{W}(\underline{2}) & = & \emptyset \rhd \underline{2} : Nat \\
\mathbb{W}(x\,\underline{2}) & = & \{x : Nat \to t\} \rhd x\,\underline{2} : t \\
\\
\mathbb{W}(x) & = & \{x : u\} \rhd x : u \\
\mathbb{W}(true) & = & \emptyset \rhd true : Bool \\
\mathbb{W}(x\,true) & = & \{x : Bool \to v\} \rhd x\,true : v \\
\\
\mathbb{W}(M) & = & falla
\end{array}$$

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

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 $P : s \to s \times s$ 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

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

- 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