Paradigmas de Programación

Compilación
Inferencia de tipos
Máquinas abstractas

1er cuatrimestre de 2024

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Introducción

Inferencia de tipos

Máquinas abstractas

Compiladores

¿Qué es un compilador?

Un compilador es un programa que traduce programas:

Entrada:	programa escrito en un **lenguaje fuente**.
Salida:	programa escrito en un **lenguaje objeto**.

Este proceso de traducción debe **preservar la semántica**.
(O, mejor: aquellos aspectos que nos interesen de la semántica).

Compiladores

¿Para qué queremos un compilador?

Motivación principal

Traducir de lenguajes de alto nivel a lenguajes de bajo nivel.

$$x = \frac{-b+\sqrt{b^2-4ac}}{2a}$$

```
movsd xmm0, QWORD PTR -32[rbp]
mulsd xmm0, xmm0
movsd xmm2, QWORD PTR -24[rbp]
movsd xmm1, QWORD PTR .LC0[rip]
mulsd xmm1, xmm2
mulsd xmm1, QWORD PTR -16[rbp]
subsd xmm0, xmm1
call sqrt@PLT
subsd xmm0, QWORD PTR -32[rbp]
movsd xmm1, QWORD PTR .LC1[rip]
divsd xmm0, xmm1
movsd xmm1, QWORD PTR -24[rbp]
mulsd xmm0, xmm1
movsd QWORD PTR -8[rbp], xmm0
```

Compiladores

Fases típicas de un compilador

	programa fuente
ANÁLISIS SINTÁCTICO	$\downarrow$
	árbol sintáctico
ANÁLISIS SEMÁNTICO	$\downarrow$
	árbol sintáctico con anotaciones
COMPILACIÓN	$\downarrow$
	representación intermedia
OPTIMIZACIÓN	$\downarrow$
	representación intermedia optimizada
GENERACIÓN DE CÓDIGO	$\downarrow$
	programa objeto

Introducción

Inferencia de tipos

Máquinas abstractas

Inferencia de tipos

Notación

Términos **sin** anotaciones de tipos:

$$U ::= x \mid \lambda x.\, U \mid U\, U \mid \mathsf{True} \mid \mathsf{False} \mid \mathsf{if}\ U\ \mathsf{then}\ U\ \mathsf{else}\ U$$

Términos **con** anotaciones de tipos:

$$M ::= x \mid \lambda x : \tau.\, M \mid M\, M \mid \mathsf{True} \mid \mathsf{False} \mid \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M$$

Notamos $\mathtt{erase}(M)$ al término sin anotaciones de tipos que resulta de borrar las anotaciones de tipos de M.

Ejemplo: $\mathtt{erase}((\lambda x : \mathsf{Bool}.\, x)\, \mathsf{True}) = (\lambda x.\, x)\, \mathsf{True}$.

Inferencia de tipos

Definición

Un término U sin anotaciones de tipos es **tipable** sii existen:

un contexto de tipado Γ

un término con anotaciones de tipos M

un tipo τ

tales que $\texttt{erase}(M) = U$ y $\Gamma \vdash M : \tau$.

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
  tales que $\texttt{erase}(M) = U$ y $\Gamma \vdash M : \tau$.

Veremos un algoritmo para resolver este problema.

Inferencia de tipos

El algoritmo se basa en manipular tipos *parcialmente conocidos*.

Ejemplo — tipos parcialmente conocidos

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

Incorporamos *incógnitas* ($?1, ?2, ?3, \ldots$) a los tipos.

Vamos a necesitar resolver *ecuaciones* entre tipos con incógnitas.

Ejemplo — ecuaciones entre tipos

- $(?1 \to \mathsf{Bool}) \stackrel{?}{=} ((\mathsf{Bool} \to \mathsf{Bool}) \to ?2)$
  tiene solución: $?1 := (\mathsf{Bool} \to \mathsf{Bool})$ y $?2 := \mathsf{Bool}$.
- $(?1 \to ?1) \stackrel{?}{=} ((\mathsf{Bool} \to \mathsf{Bool}) \to ?2)$
  tiene solución: $?1 := (\mathsf{Bool} \to \mathsf{Bool})$ y $?2 := (\mathsf{Bool} \to \mathsf{Bool})$.
- $(?1 \to \mathsf{Bool}) \stackrel{?}{=} ?1$
  no tiene solución.

Unificación

Suponemos fijado un conjunto finito de constructores de tipos:

- Tipos constantes: Bool, Int, ...
- Constructores unarios: (List $\bullet$), (Maybe $\bullet$), ...
- Constructores binarios: $(\bullet \to \bullet)$, $(\bullet \times \bullet)$, (Either $\bullet$ $\bullet$), ...
- (Etcétera).

Los tipos se forman usando incógnitas y constructores:

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

La **unificación** es el problema de resolver sistemas de ecuaciones entre tipos con incógnitas.

Veremos primero un algoritmo de unificación.
Luego lo usaremos para dar un algoritmo de inferencia de tipos.

Unificación

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

Notamos:

$$\{?k_1 := \tau_1, \dots, ?k_n := \tau_n\}$$

a la sustitución $\mathbf{S}$ tal que $\mathbf{S}(?k_i) = \tau_i$ para cada $1 \leq i \leq n$ y $\mathbf{S}(?k) = ?k$ 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 $\mathbf{S}$.

Ejemplo — aplicación de una sustitución a un tipo

Si $\mathbf{S} = \{?1 := \text{Bool},\ ?3 := (?2 \to ?2)\}$, entonces:

$$\mathbf{S}((?1 \to \text{Bool}) \to ?3) = ((\text{Bool} \to \text{Bool}) \to (?2 \to ?2))$$

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 $\mathbf{S}$ tal que:

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

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

$$\ldots$$

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

Unificación

En general, la solución a un problema de unificación no es única.

Ejemplo — problema de unificación con infinitas soluciones

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

tiene infinitos unificadores:

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

Unificación

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 instanciando variables de $\mathbf{S}_A$.

Para el siguiente problema de unificación:

$$E = \{(?1 \to \mathsf{Bool}) \overset{?}{=} ?2\}$$

las siguientes sustituciones son unificadores:

- $\mathbf{S}_1 = \{?1 := \mathsf{Bool},\ ?2 := (\mathsf{Bool} \to \mathsf{Bool})\}$
- $\mathbf{S}_2 = \{?1 := \mathsf{Int},\ ?2 := (\mathsf{Int} \to \mathsf{Bool})\}$
- $\mathbf{S}_3 = \{?1 := ?3,\ ?2 := (?3 \to \mathsf{Bool})\}$
- $\mathbf{S}_4 = \{?2 := (?1 \to \mathsf{Bool})\}$

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

Algoritmo de unificación de Martelli–Montanari

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 $E \rightarrow_{\mathbf{S}} E'$.
  La resolución del problema E se reduce a resolver otro problema E', aplicando la sustitución $\mathbf{S}$.

Hay dos posibilidades:

1. $E = E_0 \rightarrow_{\mathbf{S}_1} E_1 \rightarrow_{\mathbf{S}_2} E_2 \rightarrow \ldots \rightarrow_{\mathbf{S}_n} E_n \rightarrow_{\mathbf{S}_{n+1}} \texttt{falla}$
   En tal caso el problema de unificación E no tiene solución.
2. $E = E_0 \rightarrow_{\mathbf{S}_1} E_1 \rightarrow_{\mathbf{S}_2} E_2 \rightarrow \ldots \rightarrow_{\mathbf{S}_n} E_n = \emptyset$
   En tal caso el problema de unificación E tiene solución.

Algoritmo de unificación de Martelli–Montanari

$$\{x \stackrel{?}{=} x\} \cup E \xrightarrow{\texttt{Delete}} E$$

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

$$\{\tau \stackrel{?}{=} ?n\} \cup E \xrightarrow{\texttt{Swap}} \{?n \stackrel{?}{=} \tau\} \cup E$$

si τ no es una incógnita

$$\{?n \stackrel{?}{=} \tau\} \cup E \xrightarrow{\texttt{Elim}}_{\{?n := \tau\}} E' = \{?n := \tau\}(E)$$

si $?n$ no ocurre en τ

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

si $C \neq C'$

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

si $?n \neq \tau$

y $?n$ ocurre en τ

Algoritmo de unificación de Martelli–Montanari

Teorema (Corrección del algoritmo de Martelli–Montanari)

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 $\emptyset$:

$$E = E_0 \to_{\mathbf{S}_1} E_1 \to_{\mathbf{S}_2} E_2 \to \ldots \to_{\mathbf{S}_n} E_n = \emptyset$$

   Además, $\mathbf{S} = \mathbf{S}_n \circ \ldots \circ \mathbf{S}_2 \circ \mathbf{S}_1$ es un unificador para E.

   Además, dicho unificador es el *más general* posible.
   (Salvo renombre de incógnitas).

Definición (Unificador más general)

Notamos $\mathrm{mgu}(E)$ al unificador más general de E, si existe.

Algoritmo de unificación de Martelli–Montanari

Ejemplo

Calcular unificadores más generales para los siguientes problemas de unificación:

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

Algoritmo $\mathbb{W}$ de inferencia de tipos

El algoritmo $\mathbb{W}$ recibe un término U sin anotaciones de tipos.

Procede recursivamente sobre la estructura de U:

- Puede fallar, indicando que U no es tipable.
- Puede tener éxito.
  En tal caso devuelve una tripla (Γ, M, τ),
  donde $\mathtt{erase}(M) = U$ y $\Gamma \vdash M : \tau$ es válido.

Escribimos $\mathbb{W}(U) \rightsquigarrow \Gamma \vdash M : \tau$ para indicar que el algoritmo de inferencia tiene éxito cuando se le pasa U como entrada y devuelve una tripla (Γ, M, τ).

Algoritmo $\mathbb{W}$ de inferencia de tipos

$$\frac{}{\mathbb{W}(\mathsf{True}) \ \leadsto \ \varnothing \vdash \mathsf{True} : \mathsf{Bool}}$$

$$\frac{}{\mathbb{W}(\mathsf{False}) \ \leadsto \ \varnothing \vdash \mathsf{False} : \mathsf{Bool}}$$

$$\frac{?k \text{ es una incógnita fresca}}{\mathbb{W}(x) \ \leadsto \ x : ?k \vdash x : ?k}$$

Algoritmo $\mathbb{W}$ de inferencia de tipos

$$\begin{array}{rcl} \mathbb{W}(U_1) & \rightsquigarrow & \Gamma_1 \vdash M_1 : \tau_1 \\ \mathbb{W}(U_2) & \rightsquigarrow & \Gamma_2 \vdash M_2 : \tau_2 \\ \mathbb{W}(U_3) & \rightsquigarrow & \Gamma_3 \vdash M_3 : \tau_3 \end{array}$$

$$\frac{\mathbf{S} = \mathrm{mgu}\left(\begin{array}{l} \{\tau_1 \stackrel{?}{=} \mathrm{Bool}, \tau_2 \stackrel{?}{=} \tau_3\} \cup \\ \{\Gamma_i(x) \stackrel{?}{=} \Gamma_j(x) \mid i, j \in \{1,2,3\},\ x \in \Gamma_i \cap \Gamma_j\} \end{array} \right)}{\begin{array}{lcl} \mathbb{W}(\mathrm{if}\ U_1\ \mathrm{then}\ U_2\ \mathrm{else}\ U_3) & \rightsquigarrow & \mathbf{S}(\Gamma_1) \cup \mathbf{S}(\Gamma_2) \cup \mathbf{S}(\Gamma_3) \vdash \\ & & \quad \mathbf{S}(\mathrm{if}\ M_1\ \mathrm{then}\ M_2\ \mathrm{else}\ M_3) : \mathbf{S}(\tau_2) \end{array}}$$

Algoritmo $\mathbb{W}$ de inferencia de tipos

$$\dfrac{\begin{array}{c} \mathbb{W}(U) \;\leadsto\; \Gamma_1 \vdash M : \tau \\ \mathbb{W}(V) \;\leadsto\; \Gamma_2 \vdash N : \sigma \\ ?k \text{ es una incógnita fresca} \\ \mathbf{S} = \mathrm{mgu}\{\tau \stackrel{?}{=} \sigma \to ?k\} \cup \{\Gamma_1(x) \stackrel{?}{=} \Gamma_2(x) : x \in \Gamma_1 \cap \Gamma_2\} \end{array}}{\mathbb{W}(U\,V) \;\leadsto\; \mathbf{S}(\Gamma_1) \cup \mathbf{S}(\Gamma_2) \vdash \mathbf{S}(M\,N) : \mathbf{S}(?k)}$$

Algoritmo $\mathbb{W}$ de inferencia de tipos

$$\frac{\mathbb{W}(U) \;\rightsquigarrow\; \Gamma \vdash M : \tau \qquad \sigma = \begin{cases} \Gamma(x) & \text{si } x \in \Gamma \\ \text{una incógnita fresca } ?k & \text{si no} \end{cases}}{\mathbb{W}(\lambda x.\, U) \;\rightsquigarrow\; \Gamma \ominus \{x\} \vdash \lambda x : \sigma.\ M : \sigma \to \tau}$$

Algoritmo $\mathbb{W}$ de inferencia de tipos

Teorema (Corrección del algoritmo $\mathbb{W}$)

1. Si U no es tipable, $\mathbb{W}(U)$ falla al resolver alguna unificación.
2. Si U es tipable, $\mathbb{W}(U) \rightsquigarrow \Gamma \vdash M : \tau$,
   donde $\mathtt{erase}(M) = U$ y $\Gamma \vdash M : \tau$ es un juicio válido.

   Además, $\Gamma \vdash M : \tau$ es el juicio de tipado más general posible. Más precisamente, si $\Gamma' \vdash M' : \tau'$ es un juicio válido y $\mathtt{erase}(M') = U$, existe una sustitución $\mathbf{S}$ tal que:

$$\begin{array}{rcl} \Gamma' & \supseteq & \mathbf{S}(\Gamma) \\ M' & = & \mathbf{S}(M) \\ \tau' & = & \mathbf{S}(\tau) \end{array}$$

Algoritmo $\mathbb{W}$ de inferencia de tipos

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

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

Introducción

Inferencia de tipos

Máquinas abstractas

Un compilador minimalista

Imaginemos una máquina con tres instrucciones:

$$\texttt{LDI}(n) \quad \texttt{ADD} \quad \texttt{MUL}$$

Un programa ℓ (lista de instrucciones) opera sobre una *pila* π:

$\texttt{LDI}(n) : \ell$	π	$\longrightarrow$	ℓ	$n : \pi$
$\texttt{ADD} : \ell$	$m : n : \pi$	$\longrightarrow$	ℓ	$(n + m) : \pi$
$\texttt{MUL} : \ell$	$m : n : \pi$	$\longrightarrow$	ℓ	$(n * m) : \pi$

Consideremos la siguiente gramática de expresiones aritméticas:

$$E ::= n \mid E + E \mid E * E$$

Una expresión se puede **compilar** a una lista de instrucciones:

$$\begin{array}{rcl} \mathcal{C}\{n\} & = & \texttt{LDI}(n) \\ \mathcal{C}\{E_1 + E_2\} & = & \mathcal{C}\{E_1\}; \mathcal{C}\{E_2\}; \texttt{ADD} \\ \mathcal{C}\{E_1 * E_2\} & = & \mathcal{C}\{E_1\}; \mathcal{C}\{E_2\}; \texttt{MUL} \end{array}$$

Un compilador minimalista

Ejemplo — compilación

$$
\begin{aligned}
& \mathcal{C}\{(2 * 3) + (4 * 5)\} \\
= \quad & \texttt{LDI}(2) : \texttt{LDI}(3) : \texttt{MUL} : \texttt{LDI}(4) : \texttt{LDI}(5) : \texttt{MUL} : \texttt{ADD}
\end{aligned}
$$

	código	pila
	$\texttt{LDI}(2) : \texttt{LDI}(3) : \texttt{MUL} : \texttt{LDI}(4) : \texttt{LDI}(5) : \texttt{MUL} : \texttt{ADD}$	$[]$
$\rightarrow$	$\texttt{LDI}(3) : \texttt{MUL} : \texttt{LDI}(4) : \texttt{LDI}(5) : \texttt{MUL} : \texttt{ADD}$	2
$\rightarrow$	$\texttt{MUL} : \texttt{LDI}(4) : \texttt{LDI}(5) : \texttt{MUL} : \texttt{ADD}$	$3 : 2$
$\rightarrow$	$\texttt{LDI}(4) : \texttt{LDI}(5) : \texttt{MUL} : \texttt{ADD}$	6
$\rightarrow$	$\texttt{LDI}(5) : \texttt{MUL} : \texttt{ADD}$	$4 : 6$
$\rightarrow$	$\texttt{MUL} : \texttt{ADD}$	$5 : 4 : 6$
$\rightarrow$	$\texttt{ADD}$	$20 : 6$
$\rightarrow$	$[]$	26

Un compilador para el cálculo-λ con booleanos

Veremos cómo definir un compilador:

- Lenguaje fuente: términos del cálculo-λ con booleanos.
- Lenguaje objeto: código para la máquina abstracta SECD.

El compilador implementa la estrategia call-by-value.
La evaluación de una aplicación $M\,N$ es de izquierda a derecha.

Una **máquina abstracta** es una abstracción de la arquitectura real.
Es sencillo traducir su código a un lenguaje de bajo nivel.

La máquina SECD

La máquina SECD trabaja con los siguientes tipos de datos:

Código	ℓ	$::= [\,] \mid \mathtt{i} : \ell$	lista de instrucciones
Valores	v	$::= \mathtt{tt} \mid \mathtt{ff} \mid \langle \ell, e \rangle$	booleanos y clausuras
Entornos	e	$::= [\,] \mid v : e$	lista de valores
Pilas	π	$::= [\,] \mid v : \pi$	lista de valores
Dumps	d	$::= [\,] \mid \langle \ell, \pi, d \rangle : d$	

Las instrucciones de la máquina son:

$\mathtt{LDB}(b)$	$(b \in \{\mathtt{tt}, \mathtt{ff}\})$	carga un booleano en la pila
$\mathtt{MKCLO}(\ell)$		crea una clausura en la pila
$\mathtt{LD}(n)$	$(n \in \mathbb{N})$	carga un valor del entorno en la pila
$\mathtt{AP}$		invoca a una función
$\mathtt{RET}$		retorna de una función
$\mathtt{TEST}(\ell_1, \ell_2)$		ejecuta condicionalmente ℓ_1 ó ℓ_2

Un estado de la máquina es una 4-upla (ℓ, π, e, d). Lo notamos así:

código	pila	entorno	dump

La máquina SECD — transiciones

Los estados son de la forma $\begin{array}{|c|c|c|c|}\hline \text{código} & \text{pila} & \text{entorno} & \text{dump} \\ \hline\end{array}$.

$$\begin{array}{|c|c|c|c|}\hline \mathtt{LDB}(b) : \ell & \pi & e & d \\ \hline\end{array} \longrightarrow \begin{array}{|c|c|c|c|}\hline \ell & b : \pi & e & d \\ \hline\end{array}$$

$$\begin{array}{|c|c|c|c|}\hline \mathtt{MKCLO}(\ell') : \ell & \pi & e & d \\ \hline\end{array} \longrightarrow \begin{array}{|c|c|c|c|}\hline \ell & \langle \ell', e \rangle : \pi & e & d \\ \hline\end{array}$$

$$\begin{array}{|c|c|c|c|}\hline \mathtt{LD}(n) : \ell & \pi & e & d \\ \hline\end{array} \longrightarrow \begin{array}{|c|c|c|c|}\hline \ell & v : \pi & e & d \\ \hline\end{array}$$

$$(\text{si } e[n] = v)$$

$$\begin{array}{|c|c|c|c|}\hline \mathtt{AP} : \ell & v : \langle \ell', e' \rangle : \pi & e & d \\ \hline\end{array} \longrightarrow \begin{array}{|c|c|c|c|}\hline \ell' & [\,] & v : e' & \langle \ell, \pi, e \rangle : d \\ \hline\end{array}$$

$$\begin{array}{|c|c|c|c|}\hline \mathtt{RET} : \ell & v : \pi & e & \langle \ell', \pi', e' \rangle : d \\ \hline\end{array} \longrightarrow \begin{array}{|c|c|c|c|}\hline \ell' & v : \pi' & e' & d \\ \hline\end{array}$$

$$\begin{array}{|c|c|c|c|}\hline \mathtt{TEST}(\ell_1, \ell_2) : \ell & \mathtt{tt} : \pi & e & d \\ \hline\end{array} \longrightarrow \begin{array}{|c|c|c|c|}\hline \ell_1 ; \ell & \pi & e & d \\ \hline\end{array}$$

$$\begin{array}{|c|c|c|c|}\hline \mathtt{TEST}(\ell_1, \ell_2) : \ell & \mathtt{ff} : \pi & e & d \\ \hline\end{array} \longrightarrow \begin{array}{|c|c|c|c|}\hline \ell_2 ; \ell & \pi & e & d \\ \hline\end{array}$$

La máquina SECD

Ejemplo de ejecución de la máquina SECD

$\mathtt{LD}(1) : \mathtt{LD}(0) : \mathtt{AP}$	$[\,]$	$\mathtt{tt} : \langle \mathtt{LD}(0) : \mathtt{RET}, e\rangle$	$[\,]$

$\longrightarrow$

$\mathtt{LD}(0) : \mathtt{AP}$	$\langle \mathtt{LD}(0) : \mathtt{RET}, e\rangle$	$\mathtt{tt} : \langle \mathtt{LD}(0) : \mathtt{RET}, e\rangle$	$[\,]$

$\longrightarrow$

$\mathtt{AP}$	$\mathtt{tt} : \langle \mathtt{LD}(0) : \mathtt{RET}, e\rangle$	$\mathtt{tt} : \langle \mathtt{LD}(0) : \mathtt{RET}, e\rangle$	$[\,]$

$\longrightarrow$

$\mathtt{LD}(0) : \mathtt{RET}$	$[]$	$\mathtt{tt} : e$	$\langle [], [], \mathtt{tt} : \langle \mathtt{LD}(0) : \mathtt{RET}, e\rangle\rangle : [\,]$

$\longrightarrow$

$\mathtt{RET}$	$\mathtt{tt}$	$\mathtt{tt} : e$	$\langle [], [], \mathtt{tt} : \langle \mathtt{LD}(0) : \mathtt{RET}, e\rangle\rangle : [\,]$

$\longrightarrow$

$[]$	$\mathtt{tt}$	$\mathtt{tt} : \langle \mathtt{LD}(0) : \mathtt{RET}, e\rangle$	$[\,]$

Compilación del cálculo-λ a la máquina SECD

Definición del compilador

Dada una lista de variables $\omega = [z_1, \ldots, z_n]$ y un término M del cálculo-λ, definimos una lista de instrucciones $\mathcal{C}_\omega\{M\}$ por recursión estructural sobre M:

$$
\begin{array}{lcl}
\mathcal{C}_\omega\{\mathsf{True}\} & = & \mathtt{LDB}(\mathtt{tt}) \\
\mathcal{C}_\omega\{\mathsf{False}\} & = & \mathtt{LDB}(\mathtt{ff}) \\
\mathcal{C}_\omega\{\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P\} & = & \mathcal{C}_\omega\{M\}; \mathtt{TEST}(\mathcal{C}_\omega\{N\}, \mathcal{C}_\omega\{P\}) \\
\mathcal{C}_\omega\{x\} & = & \mathtt{LD}(i) \\
 & & \text{si } i \text{ es el menor índice t.q. } \omega[i] = x \\
\mathcal{C}_\omega\{\lambda x.\, M\} & = & \mathtt{MKCLO}(\mathcal{C}_{x:\omega}\{M\}; \mathtt{RET}) \\
\mathcal{C}_\omega\{M\, N\} & = & \mathcal{C}_\omega\{M\}; \mathcal{C}_\omega\{N\}; \mathtt{AP}
\end{array}
$$

Compilación del cálculo-λ a la máquina SECD

Ejemplo de compilación

$$
\begin{aligned}
& \mathcal{C}_{[]}\{(\lambda x.\,\lambda y.\,x)\,\mathsf{True}\,\mathsf{False}\} \\
= \; & \mathcal{C}_{[]}\{(\lambda x.\,\lambda y.\,x)\,\mathsf{True}\};\mathcal{C}_{[]}\{\mathsf{False}\};\mathtt{AP} \\
= \; & \mathcal{C}_{[]}\{\lambda x.\,\lambda y.\,x\};\mathcal{C}_{[]}\{\mathsf{True}\};\mathtt{AP};\mathcal{C}_{[]}\{\mathsf{False}\};\mathtt{AP} \\
= \; & \mathtt{MKCLO}(\mathcal{C}_{[x]}\{\lambda y.\,x\};\mathtt{RET});\mathcal{C}_{[]}\{\mathsf{True}\};\mathtt{AP};\mathcal{C}_{[]}\{\mathsf{False}\};\mathtt{AP} \\
= \; & \mathtt{MKCLO}(\mathtt{MKCLO}(\mathcal{C}_{[y,x]}\{x\};\mathtt{RET});\mathtt{RET});\mathcal{C}_{[]}\{\mathsf{True}\};\mathtt{AP};\mathcal{C}_{[]}\{\mathsf{False}\};\mathtt{AP} \\
= \; & \mathtt{MKCLO}(\mathtt{MKCLO}(\mathtt{LD}(1);\mathtt{RET});\mathtt{RET});\mathcal{C}_{[]}\{\mathsf{True}\};\mathtt{AP};\mathcal{C}_{[]}\{\mathsf{False}\};\mathtt{AP} \\
= \; & \mathtt{MKCLO}(\mathtt{MKCLO}(\mathtt{LD}(1);\mathtt{RET});\mathtt{RET});\mathtt{LDB}(\mathtt{tt});\mathtt{AP};\mathtt{LDB}(\mathtt{ff});\mathtt{AP}
\end{aligned}
$$

Compilación del cálculo-λ a la máquina SECD

Teorema (Corrección del compilador)

Sea M un término cerrado del cálculo-λ y v un valor booleano.
Son equivalentes:

1. $M \to^* v$
2. $\boxed{\mathcal{C}_{[\,]}\{M\} \mid [\,] \mid [\,] \mid [\,]} \longrightarrow^* \boxed{[\,] \mid v : \pi \mid e \mid d}$
   para ciertos π, e, d.

Otras máquinas abstractas

Origen de la máquina SECD:

P. J. Landin. *The mechanical evaluation of expressions*. 1964.

Se puede extender para incorporar otras características.

Hay otros esquemas de compilación a máquinas abstractas:

1. Máquina de Krivine para evaluación call-by-name.
2. Máquina ZINC de Leroy.
3. Máquina de Sestoft, para evaluación “lazy” (call-by-need).
4. Máquina de Crégut, para evaluar términos con variables libres.

...

¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ¿ ? ? ? ? ? ? ? ? ?