Final PLP

Cristian J. Martinez

22 de Diciembre de 2014

1 Ejercicio 1

Considere la expresión **map foldr** e indique si la misma está bien formada en el sentido que es tipable. Justificar con el árbol de inferencia.

1.1 Solución

Primero defino los axiomas de tipado para **map** y **foldr** porque los voy a necesitar.

$$\frac{}{\Gamma \triangleright \text{map} : (\alpha \to \beta) \to [\alpha] \to [\beta]} \text{ (T-Map)}$$

$$\frac{}{\Gamma \triangleright \text{foldr} : (\alpha \to \beta \to \beta) \to \beta \to [\alpha] \to \beta} \text{ (T-Foldr)}$$

Ahora sí, teniendo esto, puedo definir el árbol de inferencia. Por (T-Map) y tomando $\Gamma = \emptyset$, $\alpha = a \to b \to b$ y $\beta = b \to [a] \to b$, obtenemos:

$$\frac{}{\emptyset \triangleright \text{map} : ((a \to b \to b) \to b \to [a] \to b) \to [a \to b \to b] \to [b \to [a] \to b]} \text{ (T-Map)} \tag{1}$$

Por otro lado, por (T-Foldr) y tomando $\Gamma = \emptyset$, $\alpha = a$ y $\beta = b$, obtenemos:

$$\frac{}{\emptyset \triangleright \text{foldr} : (a \to b \to b) \to b \to [a] \to b} \text{ (T-Foldr)} \tag{2}$$

Finalmente usando esto junto con la regla (T-App) y tomando $\Gamma = \emptyset$, $\sigma = (a \to b \to b) \to b \to [a] \to b$ y $\tau = [a \to b \to b] \to [b \to [a] \to b]$ nos da como resultado:

$$\frac{(1) \qquad (2)}{\emptyset \triangleright \text{map foldr} : [a \to b \to b] \to [b \to [a] \to b]} \text{ (T-App)}$$

Aclaración: para calcular MGU en cada paso hay que utilizar el algoritmo de Martelli-Montanari, por ejemplo para (1) tenemos:

$$\{\alpha \to \beta \doteq (a \to b \to b) \to b \to [a] \to b\}$$

Descomposición

$$\{\alpha \doteq (a \to b \to b), \beta \doteq b \to [a] \to b\}$$

Eliminación de variable, $\sigma_1 = [(a \to b \to b)/\alpha]$

$$\{\beta \doteq b \to [a] \to b\}$$

Eliminación de variable, $\sigma_2 = [(b \to [a] \to b)/\beta]$

$$\square$$

El MGU es $\{(a \to b \to b)/\alpha, (b \to [a] \to b)/\beta\}$

2 Ejercicio 2

Considerar la siguiente extensión al conjunto de términos del cálculo lambda tipado con referencias y booleanos:

```
M ::= ... | for (i:=M,M,M)
```

El conjunto de los tipos y valores es el mismo que antes. Lo único que se modifica es el conjunto de los términos. Una expresión de la forma `for (i:=M1,M2,M3)` representa iteración:

- `i` es la variable de iteración cuyo valor inicial viene dado por el valor de `M1` (que asumimos un `Nat` para simplificar);
- `M2` es la condición de fin; y
- `M3` es el cuerpo de la iteración. Es un comando que se en cada ciclo de la iteración.

Al igual que en los lenguajes imperativos tradicionales, `M1` se ejecuta una sola vez, `M3` se ejecuta sólo si `M2` evaluó a `True`. Por ejemplo, el siguiente programa retorna 255.

```
let x = ref 0
    in for (i:=0, !i <= 7, x := !x + !i * 2; !i = !i+1);
        !x
```

Se solicita lo siguiente:

(a) Dar la regla de tipado de `for`.

(b) Dar la semántica operacional small-step de `for`. Ayuda: le puede ser de utilidad (1) interpretar el términos de `let` y el condicional `if-then-else`, y (2) introducir una construcción auxiliar `for'(M,M)` al lenguaje, si hiciera falta.

(c) Extender el algoritmo de inferencia para `for`.

2.1 Solución

(a) Regla de tipado para `for`.

$$\frac{\Gamma \triangleright \text{M1} : Nat \quad \Gamma, x : Ref_{Nat} \triangleright \text{M2} : Bool \quad \Gamma, x : Ref_{Nat} \triangleright \text{M3} : unit}{\Gamma \triangleright \text{for(i:=M1,M2,M3)} : unit} \text{ (T-For)}$$

(b) Semántica operacional small-step de `for`.
Para este punto es importante darse cuenta que tenemos un problema y es que de alguna manera tenemos que guardar las expresiones `M2` y `M3` originales ya que en cada iteración van a ser evaluadas y si no las guardamos no tenemos forma de volver a evaluarlas una vez que se conviertan en valores.
Existen (al menos) dos opciones, la primera es hacer caso a la sugerencia de interpretar la semántica operacional en términos de `let` y el condicional `if-then-else`, e introducir la construcción `for'(M,M)`. La otra opción es sólo introducir otra construcción `for(M,M,M,M)` que vamos a usar para guardar los términos `M2` y `M3` originales, junto con una copia que utilizaremos para averiguar el valor de sus respectivas evaluaciones.

- Opción 1 (usando todas las sugerencias):

  En primer lugar hacemos uso de la semántica operacional de `let` para evaluar `M1` hasta llegar a un valor y que luego se actualice el mismo en el store μ.

$$\frac{}{\text{for(i:=M1,M2,M3)}|\mu \rightarrow \text{let i = M1 in for'(M2,M3)}|\mu}$$

  Y por último, hacemos uso de la semántica operacional de `if-then-else` para evaluar `M2` hasta llegar a un valor y, en caso de que ese valor sea `True` evaluar `M3` y a continuación volver a evaluar `for'(M2,M3)`; o en caso de que el valor sea `False` terminar la iteración y 'retornar' `unit`.

$$\frac{}{\text{for'(M2,M3)}|\mu \rightarrow \text{if M2 then (M3;for'(M2,M3)) else unit}|\mu}$$

- Opción 2:

  Primero evalúo `M1` hasta llegar a un valor y actualizo la referencia `i` en el store.

$$\frac{M1|\mu \rightarrow M1'|\mu'}{for(i := M1, M2, M3)|\mu \rightarrow for(i := M1', M2, M3)|\mu'}$$

$$\frac{}{for(i := V, M2, M3)|\mu \rightarrow for'(M2, M2, M3, M3)|\mu[i \mapsto V]}$$

  Ahora necesito evaluar una de las copias de `M2` para saber si tengo que realizar una iteración más o no (siempre evalúo la misma copia y la otra la dejo para poder 'reinicializar' su valor una vez evaluada). Si el resultado de evaluar la copia es `True`, tengo que evaluar una copia del comando `M3` de acuerdo al comportamiento esperado de `for`.

$$\frac{M2|\mu \rightarrow M2'|\mu'}{for'(M2, M2, M3, M3)|\mu \rightarrow for'(M2, M2', M3, M3)|\mu'}$$

$$\overline{for'(M2, False, M3, M3)|\mu \to unit|\mu}$$

$$\frac{M3|\mu \to M3'|\mu'}{for'(M2, True, M3, M3)|\mu \to for'(M2, True, M3, M3')|\mu'}$$

$$\overline{for'(M2, True, M3, unit)|\mu \to for'(M2, M2, M3, M3)|\mu}$$

(c) Algoritmo de inferencia para `for`.

$\mathbb{W}(\texttt{for(i:=M1,M2,M3)}) = S\Gamma_1 \bigcup S\Gamma'_2 \bigcup S\Gamma'_3 \triangleright S(\texttt{for(i:=M1,M2,M3)}) : S(unit)$

- $\mathbb{W}(M1) = \Gamma_1 \triangleright M1 : \sigma_1$
- $\mathbb{W}(M2) = \Gamma_2 \triangleright M2 : \sigma_2$
- $\mathbb{W}(M3) = \Gamma_3 \triangleright M3 : \sigma_3$

$$\rho = \begin{cases} \sigma & \text{si } \{x : \sigma\} \in \Gamma_2 \\ \text{s} & \text{sino, con s variable fresca} \end{cases}$$

$\Gamma'_2 = \Gamma_2 \backslash \{x : \rho\}$

$$\tau = \begin{cases} \sigma & \text{si } \{x : \sigma\} \in \Gamma_3 \\ \text{s} & \text{sino, con s variable fresca} \end{cases}$$

$\Gamma'_3 = \Gamma_3 \backslash \{x : \tau\}$

$S = \mathrm{MGU}(\{\sigma_1 \doteq Nat, \sigma_2 \doteq Bool, \sigma_3 \doteq Unit\, \rho \doteq Ref_{Nat}, \rho \doteq \tau\} \bigcup \{\sigma_i \doteq \sigma_j | \{x : \sigma_i\} \in \Gamma_i, \{x : \sigma_j\} \in \Gamma_j, i \neq j\})$

3 Ejercicio 3

Escribir en prolog la función `subcadenas(-S,+C)` que dada una cadena devuelve todas sus subcadenas. Por ejemplo, `subcadenas(S, [1,2,3])` debería devolver las subcadenas `[]`, `[1]`, `[2]`, `[3]`, `[1,2]`, `[2,3]`, `[1,3]` y `[1,2,3]`.

3.1 Solución

```
subcadenas([],[]).
subcadenas(Xs,[Y|Ys]) :- subcadenas(Xs,Ys).
subcadenas([X|Xs],[X|Ys]) :- subcadenas(Xs,Ys).
```