Uma Linguagem Funcional "call-by-name"

Uma linguagem funcional "call-by-name"

- Veremos agora uma pequena linguagem funcional com uma semântica que segue a ordem normal de avaliação. Ou seja, uma semântica "call-by-name" (CBN).
- A sintaxe da linguagem é idêntica à que vimos para a linguagem CBV, excepto para o caso das definições recursivas.
- A semântica CBN permite a formulação de um operador de ponto fixo

$$\langle exp \rangle ::= \mathsf{rec}\, \langle exp \rangle$$

- A semântica da linguagem CBN é bastante diferente da CBV: a avaliação dos operandos das aplicações, dos componentes dos tuplos e das alternativas é adiada até que seja evidente que esses valores são necessários para determinar o resultado do programa.

Uma linguagem funcional "call-by-name"

- O sistema de tipos para esta linguagem CBN é idêntico ao que vimos para a linguagem CBV. Apenas se acrescenta a seguinte regra de inferência de tipos

$$\frac{\Gamma \vdash e : \theta \to \theta}{\Gamma \vdash \mathsf{rec}\ e : \theta}$$

- Há expressões cuja avaliação CBN termina, embora possam divergir com uma avaliação CBV.
- Será possível definir estruturas de dados infinitas (como por exemplo as "lazy lists").

Formas canónicas

- A linguagem usa uma semântica de avaliação "call-by-name".
- $\Rightarrow$ relaciona as expressões com as formas canónicas da linguagem

$$\langle exp \rangle \Rightarrow \langle cfm \rangle$$

- Teremos formas canónicas de diferentes tipos

$$\begin{array}{rcl} \langle cfm \rangle & ::= & \lambda \langle var \rangle . \langle exp \rangle \\ & | & \ldots \mid -2 \mid -1 \mid 0 \mid 1 \mid 2 \mid \ldots \\ & | & \mathsf{True} \mid \mathsf{False} \\ & | & \langle \langle exp \rangle, \ldots, \langle exp \rangle \rangle \\ & | & @\langle tag \rangle \langle exp \rangle \\ & | & \mathsf{nil} \mid \langle exp \rangle :: \langle exp \rangle \end{array}$$

Semântica de avaliação "call-by-name"

- **Formas canónicas**

$$\frac{}{z \Rightarrow z}$$

- **Aplicação**

$$\frac{e \Rightarrow \lambda v.\widehat{e} \qquad \widehat{e}[e'/v] \Rightarrow z}{e\, e' \Rightarrow z}$$

Semântica de avaliação "call-by-name"

- **Operadores unários**

$$\frac{e \Rightarrow \lfloor i \rfloor}{-e \Rightarrow \lfloor -i \rfloor}$$

- **Operadores binários**

$$\frac{e_1 \Rightarrow \lfloor i_1 \rfloor \qquad e_2 \Rightarrow \lfloor i_2 \rfloor}{e_1 \ \mathbf{bop}\ e_2 \Rightarrow \lfloor i_1 \ \mathbf{bop}\ i_2 \rfloor} \text{ para } \mathbf{bop} \in \{+, -, *, =, \neq, <, >, \leq, \geq\}$$

$$\frac{e_1 \Rightarrow \lfloor i_1 \rfloor \qquad e_2 \Rightarrow \lfloor i_2 \rfloor}{e_1 \ \mathbf{bop}\ e_2 \Rightarrow \lfloor i_1 \ \mathbf{bop}\ i_2 \rfloor} \text{ para } \mathbf{bop} \in \{\mathsf{div}, \mathsf{mod}\} \text{ e } i_2 \neq 0$$

Açúcar sintáctico

- **Operadores booleanos**

$$\begin{aligned} \neg e &\doteq \text{if } e \text{ then False else True} \\ e \vee e' &\doteq \text{if } e \text{ then True else } e' \\ e \wedge e' &\doteq \text{if } e \text{ then } e' \text{ else False} \end{aligned}$$

Desta forma a avaliação de $\vee$ e $\wedge$ é em "curto-circuito".

Semântica de avaliação "call-by-name"

- **Expressões condicionais**

$$\frac{e_1 \Rightarrow \mathsf{True} \qquad e_2 \Rightarrow z}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Rightarrow z}$$

$$\frac{e_1 \Rightarrow \mathsf{False} \qquad e_3 \Rightarrow z}{\text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Rightarrow z}$$

Semântica de avaliação "call-by-name"

- **Tuplos**

$$\frac{e \Rightarrow \langle e_1, \ldots, e_n \rangle \qquad e_k \Rightarrow z}{e.k \Rightarrow z} \text{ para } k \in \{1, \ldots, n\}$$

Semântica de avaliação "call-by-name"

- **Alternativas**

$$\frac{e \Rightarrow @k\, e' \qquad e_k\, e' \Rightarrow z}{\mathsf{sumcase}\ e\ \mathsf{of}\ (e_1, \ldots, e_n) \Rightarrow z} \text{ para } k \in \{1, \ldots, n\}$$

Semântica de avaliação "call-by-name"

- **Listas**

$$\frac{e \Rightarrow \mathsf{nil} \qquad e_{\mathsf{e}} \Rightarrow z}{\mathsf{listcase}\ e\ \mathsf{of}\ (e_{\mathsf{e}}, e_{\mathsf{ne}}) \Rightarrow z}$$

$$\frac{e \Rightarrow e' :: e'' \qquad e_{\mathsf{ne}}\, e'\, e'' \Rightarrow z}{\mathsf{listcase}\ e\ \mathsf{of}\ (e_{\mathsf{e}}, e_{\mathsf{ne}}) \Rightarrow z}$$

Semântica de avaliação "call-by-name"

- **Recursividade**

$$\frac{e\,(\mathsf{rec}\, e) \Rightarrow z}{\mathsf{rec}\, e \Rightarrow z}$$

Açúcar sintáctico

- Padrões, definições "let" e listas podem ser definidas como açúcar sintáctico do mesmo modo que na linguagem CBV.
- O operador de ponto fixo pode ser usado para definições "letrec" mais gerais do que tinhamos na linguagem CBV

$$\langle exp \rangle ::= \text{letrec } \langle pat \rangle \equiv \langle exp \rangle, \ldots, \langle pat \rangle \equiv \langle exp \rangle \text{ in } \langle exp \rangle$$

- Onde

$$\begin{aligned}&\mathsf{FV}(\text{letrec } p_1 \equiv e_1, \ldots, p_n \equiv e_n \text{ in } e)\\ &\quad = (\mathsf{FV}(e_1) \cup \ldots \cup \mathsf{FV}(e_n) \cup \mathsf{FV}(e)) - (\mathsf{FV}(p_1) \cup \ldots \cup \mathsf{FV}(p_n))\end{aligned}$$

Açúcar sintáctico

- **Letrec**

$$\text{letrec } p_1 \equiv e_1 \text{ in } e \quad \doteq \quad \text{let } p_1 \equiv \mathsf{rec}\,(\lambda p_1.\, e_1) \text{ in } e$$

$$\begin{aligned}&\text{letrec } p_1 \equiv e_1, \ldots, p_n \equiv e_n \text{ in } e\\ &\quad \doteq \quad \text{let } \langle p_1, \ldots, p_n \rangle \equiv \mathsf{rec}\,(\lambda \langle p_1, \ldots, p_n \rangle.\, \langle e_1, \ldots, e_n \rangle) \text{ in } e\end{aligned}$$

Exemplo

Considere a seguinte função que testa se duas listas são iguais

$$\text{eqlist} : \mathbf{List\ Int} \to \mathbf{List\ Int} \to \mathbf{Bool}$$

e compare a avaliação CBN e CBV do seguinte programa

$$\begin{aligned}&\text{letrec eqlist} \equiv \lambda l_1.\, \lambda l_2.\\ &\quad \text{listcase } l_1 \text{ of } (\\ &\qquad \text{listcase } l_2 \text{ of } (\text{True}, \lambda h_2.\lambda t_2.\, \text{False}),\\ &\qquad \lambda h_1.\lambda t_1.\, \text{listcase } l_2 \text{ of } (\text{False}, \lambda h_2.\lambda t_2.\, h_1 = h_2 \ \wedge\ \text{eqlist}\, t_1\, t_2)\\ &\quad)\\ &\text{in} \quad \text{eqlist } (1 :: 2 :: 3 :: \text{nil})\ (3 :: 2 :: 1 :: \text{nil})\end{aligned}$$

Estruturas de dados infinitas

A semântica CBN permite definir estruturas de dados infinitas, como por exemplo as seguintes *"lazy lists"*

A lista infinita de zeros

$$\text{letrec zerolist} \equiv 0 :: \text{zerolist } \ldots$$

ou simplesmente, $\mathsf{rec}\,(\lambda l.0 :: l)$

A lista infinita de números naturais

$$\begin{aligned}&\text{letrec } \ \text{map} \equiv \lambda f.\, \lambda l.\, \text{listcase } l \text{ of } (\text{nil}, \lambda h.\lambda t.\, f\, h :: \text{map}\, f\, t)\\ &\text{in} \quad \ \mathsf{rec}\,(\lambda l.\, 0 :: \text{map}\,(\lambda x.x + 1)\, l)\end{aligned}$$

Apresente uma definição alternativa para a lista de naturais chamada natlist.

Como se comporta a função eqlist com listas infinitas?