Uma Linguagem Funcional

(Reynolds, Theories of Programming Languages)

Maria João Frade

HASLab - INESC TEC
Departamento de Informática, Universidade do Minho

2021/2022

Uma Linguagem Funcional

Apresentam-se agora duas linguagens funcionais que estendem o lambda calculus com operações lógicas e aritméticas, expressões condicionais, definições, tuplos, alternativas e listas.

- Uma linguagem estrita, com uma semântica de avaliação aplicativa ("call-by-value").
- Uma linguagem não estrita, com uma semântica de avaliação normal ("call-by-name").

Uma Linguagem Funcional "call-by-value"

Sintaxe abstracta

$$
\begin{array}{rcl}
\langle exp \rangle & ::= & \langle var \rangle \mid \lambda\langle var \rangle.\langle exp \rangle \mid \langle exp \rangle\langle exp \rangle \\
& \mid & 0 \mid 1 \mid 2 \mid \ldots \mid \mathsf{True} \mid \mathsf{False} \\
& \mid & \neg\langle exp \rangle \mid -\langle exp \rangle \mid \langle exp \rangle \ \mathbf{bop} \ \langle exp \rangle \\
& \mid & \mathsf{if}\ \langle exp \rangle\ \mathsf{then}\ \langle exp \rangle\ \mathsf{else}\ \langle exp \rangle \\
& \mid & \langle \langle exp \rangle, \ldots, \langle exp \rangle \rangle \mid \langle exp \rangle.\langle tag \rangle \\
& \mid & @\langle tag \rangle\langle exp \rangle \mid \mathsf{sumcase}\ \langle exp \rangle\ \mathsf{of}\ (\langle exp \rangle, \ldots, \langle exp \rangle) \\
& \mid & \mathsf{nil} \mid \langle exp \rangle :: \langle exp \rangle \mid \mathsf{listcase}\ \langle exp \rangle\ \mathsf{of}\ (\langle exp \rangle, \langle exp \rangle) \\
& \mid & \lambda\langle pat \rangle.\langle exp \rangle \mid \mathsf{let}\ \langle pat \rangle \equiv \langle exp \rangle, \ldots, \langle pat \rangle \equiv \langle exp \rangle\ \mathsf{in}\ \langle exp \rangle \\
& \mid & \mathsf{letrec}\ \langle var \rangle \equiv \lambda\langle pat \rangle.\langle exp \rangle, \ldots, \langle var \rangle \equiv \lambda\langle pat \rangle.\langle exp \rangle\ \mathsf{in}\ \langle exp \rangle \\
& & \\
& & \mathbf{bop} \in \{+, -, *, \mathsf{div}, \mathsf{mod}, =, \neq, <, >, \leq, \geq, \vee, \wedge\} \\
& & \\
\langle tag \rangle & ::= & 1 \mid 2 \mid \ldots \\
\langle pat \rangle & ::= & \langle var \rangle \mid \langle \langle pat \rangle, \ldots, \langle pat \rangle \rangle
\end{array}
$$

Precedências e associatividade dos operadores

- Para evitar o uso excessivo de parêntesis estipulamos a seguinte lista de precedências. A associatividade é à esquerda, excepto nos casos assinalados.
  - .
  - aplicação, @
  - *, div, mod
  - - , +
  - $=, \neq, <, >, \leq, \geq$
  - $\neg$
  - $\wedge$
  - $\vee$
  - :: (associativa à direita)

Formas canónicas

- A linguagem usa uma semântica de avaliação "call-by-value" (CBV).
- Serão dadas regras de inferência para a relação de avaliação $\Rightarrow_E$ (escreveremos abreviadamente $\Rightarrow$).
- $\Rightarrow$ relaciona as expressões com as formas canónicas da linguagem.
- Teremos formas canónicas de diferentes tipos

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

Notação

- Avaliação CBV: $\langle exp \rangle \Rightarrow \langle cfm \rangle$
- Se a avaliação de uma expressão e não terminar ou bloquear, escreveremos simplemente $e\uparrow$.
- Na apresentação das regras da semântica CBV usaremos as seguintes meta-variáveis

e	$\langle exp \rangle_{\text{closed}}$	i	$\mathbb{Z}$
$\widehat{e}$	$\langle exp \rangle$	k, n	$\mathbb{N}$
z	$\langle cfm \rangle$	b	$\mathbb{B}$
v, u	$\langle var \rangle$	p	$\langle pat \rangle$

- $\lfloor i \rfloor$ e $\lfloor b \rfloor$ denotam as formas canónicas das expressões i e b.
  - Ex: se i denota o inteiro 3 e i' o inteiro 2, $\lfloor i + i' \rfloor$ denota o inteiro 5.

Semântica de avaliação "call-by-value"

- **Formas canónicas**

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

- **Aplicação**

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

Semântica de avaliação "call-by-value"

- **Operadores unários**

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

$$\frac{e \Rightarrow \lfloor b \rfloor}{\neg e \Rightarrow \lfloor \neg b \rfloor}$$

Semântica de avaliação "call-by-value"

- **Operadores binários**

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

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

$$\frac{e_1 \Rightarrow \lfloor b_1 \rfloor \qquad e_2 \Rightarrow \lfloor b_2 \rfloor}{e_1 \ \mathbf{bop} \ e_2 \Rightarrow \lfloor b_1 \ \mathbf{bop} \ b_2 \rfloor} \text{ para } \mathbf{bop} \in \{\vee, \wedge\}$$

Semântica de avaliação "call-by-value"

- **Expressões condicionais**

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

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

Semântica de avaliação "call-by-value"

- **Tuplos**

$$\frac{e_1 \Rightarrow z_1 \quad \cdots \quad e_n \Rightarrow z_n}{\langle e_1, \ldots, e_n \rangle \Rightarrow \langle z_1, \ldots, z_n \rangle}$$

$$\frac{e \Rightarrow \langle z_1, \ldots, z_n \rangle}{e.k \Rightarrow z_k} \text{ para } k \in \{1, \ldots, n\}$$

Semântica de avaliação "call-by-value"

- **Alternativas**

$$\frac{e \Rightarrow z}{@k\, e \Rightarrow @k\, z}$$

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

Semântica de avaliação "call-by-value"

- **Listas**

$$\frac{}{\text{nil} \Rightarrow \text{nil}} \qquad \frac{e \Rightarrow z \qquad e' \Rightarrow z'}{e :: e' \Rightarrow z :: z'}$$

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

Açúcar sintáctico

Embora possam ser definidas regras de avaliação para as definições e padrões, é mais simples ver essas construções como açúcar sintáctico.

- **Definições**

$$\text{let } p_1 \equiv e_1, \ldots, p_n \equiv e_n \text{ in } e \;\doteq\; (\lambda p_1 \ldots \lambda p_n . e)\, e_1 \ldots e_n$$

- **Padrões**

$$\lambda\langle p_1, \ldots, p_n\rangle . e \;\doteq\; \lambda v . \text{let } p_1 \equiv v.1, \ldots, p_n \equiv v.n \text{ in } e$$

Açúcar sintáctico

$$\begin{aligned}
&\text{let } \langle x, y\rangle \equiv w, z \equiv 2 * n \text{ in } x + y + z \\
&\doteq\; (\lambda\langle x, y\rangle . \lambda z . x + y + z)\, w\, (2 * n) \\
&\doteq\; (\lambda v . \text{let } x \equiv v.1, y \equiv v.2 \text{ in } \lambda z . x + y + z)\, w\, (2 * n) \\
&\doteq\; (\lambda v . (\lambda x . \lambda y . \lambda z . x + y + z)\, (v.1)\, (v.2))\, w\, (2 * n)
\end{aligned}$$

Açúcar sintáctico

Uma definição alternativa para listas

- As listas podem ser vistas como estruturas de dados construidas à custa das alternativas e dos tuplos.
- Nesta abordagem uma lista é um valor alternativo que pode ser:
  - a etiqueta 1 seguida do tuplo vazio; ou
  - a etiqueta 2 seguida do par com a cabeça e cauda da lista.

$$\begin{array}{rcl} \text{nil} & \doteq & @1\,\langle\rangle \\ e :: e' & \doteq & @2\,\langle e, e'\rangle \\ \text{listcase } e \text{ of } (e_{\mathsf{e}}, e_{\mathsf{ne}}) & \doteq & \text{sumcase } e \text{ of } (\lambda\langle\rangle.e_{\mathsf{e}}, \lambda\langle x, y\rangle.e_{\mathsf{ne}}\,x\,y) \\ & & \text{com } x, y \notin \mathsf{FV}(e_{\mathsf{ne}}) \end{array}$$

Recursividade

- Numa definição let $v \equiv e$ in e' as ocorrências de v em e' são ligadas, mas eventuais ocorrências de v em e são livres, dado que

$$\text{let } v \equiv e \text{ in } e' \doteq (\lambda v.\, e')\, e$$

- Ou seja

$$\mathsf{FV}(\lambda p.\, e) = \mathsf{FV}(e) - \mathsf{FV}(p)$$

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

- Portanto, numa definição let $v \equiv e$ in e', v não pode representar uma função recursiva.

Recursividade

- A definição de funções recursivas é feita através da construção

$$\text{letrec } \langle var\rangle \equiv \lambda\langle pat\rangle.\langle exp\rangle, \ldots, \langle var\rangle \equiv \lambda\langle pat\rangle.\langle exp\rangle \text{ in } \langle exp\rangle$$

Note como as retrições sintáticas apenas deixam definir recursivamente funções (isto é característico do CBV).

- Assim,

$$\begin{array}{l} \mathsf{FV}(\text{letrec } v_1 \equiv \lambda p_1.e_1, \ldots, v_n \equiv \lambda p_n.e_n \text{ in } e) \\ \quad = (\mathsf{FV}(\lambda p_1.e_1) \cup \ldots \cup \mathsf{FV}(\lambda p_n.e_n) \cup \mathsf{FV}(e)) - \{v_1, \ldots, v_n\} \end{array}$$

Semântica de avaliação "call-by-value"

- **Recursividade**

$$\frac{(\lambda v_1.\, \ldots \lambda v_n.\, e)\,(\lambda u_1.\, e_1^*) \cdots (\lambda u_n.\, e_n^*) \Rightarrow z}{\text{letrec } v_1 \equiv \lambda u_1.e_1, \ldots, v_n \equiv \lambda u_n.e_n \text{ in } e \Rightarrow z}$$

onde cada e_i^* representa letrec $v_1 \equiv \lambda u_1.e_1, \ldots, v_n \equiv \lambda u_n.e_n$ in e_i e $v_1, \ldots, v_n \notin \{u_1, \ldots, u_n\}$

Note que $(\lambda u_i.\, e_i^*)$ são formas canónicas. Portanto, os letrec's interiores não serão avaliados a não ser que a aplicação de $(\lambda u_i.\, e_i^*)$ a um argumento seja avaliada, ou seja, a não ser que a chamada recursiva seja avaliada.

Podemos ver letrec $v_1 \equiv \lambda u_1.e_1, \ldots, v_n \equiv \lambda u_n.e_n$ in $e \Rightarrow z$ como a avaliação CBV da expressão e no contexto contendo as definições das funções $v_1, \ldots, v_n$, mutuamente recursivas.

Exemplos

Alguns exemplos

- A função factorial

$$\text{letrec fact} \equiv \lambda n.\, \text{if } n = 0 \text{ then } 1 \text{ else } n * \text{fact}\,(n-1) \;\ldots$$

- A concatenação de listas

$$\text{letrec append} \equiv \lambda x.\, \lambda y.\, \text{listcase } x \text{ of } (y, \lambda h.\, \lambda t.\, h :: \text{append}\, t\, y) \;\ldots$$

- A função *map*

$$\text{letrec map} \equiv \lambda f.\, \lambda l.\, \text{listcase } l \text{ of } (\text{nil}, \lambda h.\, \lambda t.\, f\, h :: \text{map}\, f\, t) \;\ldots$$

- A função *foldr*

$$\text{letrec foldr} \equiv \lambda f.\, \lambda z.\, \lambda l.\, \text{listcase } l \text{ of } (z, \lambda h.\, \lambda t.\, f\, h\, (\text{foldr}\, f\, z\, t)) \;\ldots$$

Exercícios

Apresente a avaliação de letrec fact $\equiv \ldots$ in $(\text{fact}\,1)$ até à sua forma canónica.

Tendo definido funções de ordem superior, podemos definir novas funções utilizando definições não recursivas. Por exemplo:

$$\text{let append} \equiv \lambda x.\, \lambda y.\, \text{foldr}\,(\lambda h.\, \lambda r.\, h :: r)\, y\, x \;\ldots$$
$$\text{let inc} \equiv \lambda l.\, \text{map}\,(\lambda x.\, x+1)\, l \;\ldots$$

Apresente definições alternativas para as funções fact e map.

Defina as seguintes funções (com recursividade explícita)

- Valor absoluto de um inteiro.
- Comprimento de uma lista.
- Testar se uma lista de inteiros está ordenada.
- Fusão de listas ordenadas.