Semântica Formal
Trabalho Final:
Linguagem de Programação *Esquema*

Alex Gliesch
Daniel Silva
Hyago Sallet

2 de Dezembro de 2013

1 Introdução

O trabalho da disciplina consistiu em descrever a semântica operacional e o sistema de tipos de uma linguagem de programação estaticamente tipada, assim como implementar opcionalmente os itens citados.

A linguagem criada, chamada *Esquema*, é um subconjunto da linguagem funcional Scheme. Ela implementa as operações mais básicas da linguagem, como funções lambda, recursão, definições de variáveis globais, escopos locais, e, obviamente, listas. A grande diferença que *Esquema* tem para Scheme é que ela é estaticamente tipada.

2 Elementos da linguagem

A seguir, descreve-se os principais elementos implementados e definidos para a linguagem.

empty é o tipo de lista vazia, que também é convertível pra qualquer tipo de lista.

($lambda : T_r$ ($x_i : T_i$...) e) é uma definição de função anônima. T_r corresponde ao tipo de retorno, os parâmetros são x_i e seus tipos T_i, sendo que eles podem ocorrer em um número arbitrário, e a operação a ser executada é e.

(if *teste* e_1 e_2) é o teste condicional, que retorna e_1 se *teste* for verdadeiro, e e_2 se teste for falso.

($define$ x e) adiciona uma variável ao contexto global (ou local) com o identificador x e valor e. É interessante notar que, nesta linguagem, o tipo de retorno do $define$ é o tipo da atribuição.

(*local* (e_i...) e_r) permite declarar variáveis locais e_i que existirão somente no contexto da expressão de retorno e_r. Além disso, caso a expressão de retorno for um $define$, ele não é propagado para o contexto de cima.

(*begin* e_i...) executa em ordem todos seus argumentos e retorna o valor do último argumento

(*list* e_i...) cria uma lista com os argumentos e_i passados.

+*, *-*, ***, *<*, *and*, *not* e *empty? são funções *built-in* na linguagem.

3 Tipos básicos

Os tipos básicos são inteiros, booleanos, strings e a lista vazia. A partir destes, podem ser criados tipos compostos, como funções que recebem e retornam tipos básicos, ou listas dos mesmos.

Um dos diferenciais da linguagem é que ela permite inteiros de precisão arbitrária.

4 A biblioteca padrão

Como pode-se observar, a linguagem tem muitas poucas operações *built-in*. Porém, isto não a faz ser menos poderosa: através destas operações básicas, pode-se definir operações mais complexas. Um exemplo disto é a operação $<$, a partir da qual pode-se definir as operações $>, \geq, \leq, =$ e $\neq$.

A biblioteca padrão da linguagem, fornecida junto com o interpretador, oferece implementações das funcionalidades básicas de programação, como comparadores, funções matemáticas, manipulação de listas, *sorting*, e muitos outros.

No entanto, note que, por ela não fazer parte da linguagem, e sim ser uma extensão, ela não tem nenhuma garantia de funcionalidade, estando sucetível a erros, como, por exemplo, funções que nunca terminam. Use a seu próprio risco.

5 Sintaxe

$$
\begin{aligned}
t ::== & \ (t) \\
| & \ n \\
| & \ b \\
| & \ s \\
| & \ empty \\
| & \ (lambda{:}T\ (x_1{:}T_1\ x_2{:}T_2\ ...\ x_n{:}T_n)\ t_r) \\
| & \ (if\ t_1\ t_2\ t_3) \\
| & \ (define\ x\ t_1) \\
| & \ (local\ (t_1\ t_2\ ...\ t_n)\ t_r) \\
| & \ (binop\ t_1\ t_2) \\
| & \ (unop\ t_1) \\
| & \ (n\text{-}ary\ t_1\ t_2\ ...\ t_n)
\end{aligned}
$$

onde:

$n \in \mathbb{Z}$

$b \in \{true, false\}$

$s \in$ conjunto das strings possíveis

$binop \in \{+, -, *, and, cons\}$

$unop \in \{not, empty?, first, rest\}$

$n\text{-}ary \in \{begin, list\}$

x representa um identificador, definido pelo usuário

6 Semântica operacional

Para definir a semântica, foi definido um mapeamento de identificadores para valores α, que representa o contexto global de execução. Quando adiciona-se algum símbolo ao contexto (através de um define, por exemplo), o α é atualizado. Desta maneira, uma configuração é da forma $\langle \alpha, e \rangle$, onde e é uma expressão da linguagem (programa).

Como notação, utilizar-se-á n como um número inteiro, b como um booleano, s uma string, l uma lista, t um termo da linguagem, e v um valor (termo que não progride).

6.1 Semântica da operação de progresso

$$\frac{t_1 \to t_1'}{\langle \alpha, t_1 \rangle \to \langle \alpha', t_1' \rangle} \quad [PROG]$$

6.2 Semântica da operação busca de contexto

$$\frac{x \in Dom(\alpha)\ e\ \alpha(x) = v}{\langle \alpha, x \rangle \to \langle \alpha, v \rangle} \quad [CTX]$$

6.3 Semântica das operações binárias

$$\frac{[[n]] = [[n_1 + n_2]]}{\langle \alpha, (+\ n_1\ n_2) \rangle \to \langle \alpha, n \rangle} \quad [OP+]$$

$$\frac{[[n]] = [[n_1 - n_2]]}{\langle \alpha, (-\ n_1\ n_2) \rangle \to \langle \alpha, n \rangle} \quad [OP-]$$

$$\frac{[[n]] = [[n_1 \times n_2]]}{\langle \alpha, (*\ n_1\ n_2) \rangle \to \langle \alpha, n \rangle} \quad [OP*]$$

$$\frac{[[b]] = [[n_1 < n_2]]}{\langle \alpha, (<\ n_1\ n_2) \rangle \to \langle \alpha, b \rangle} \quad [OP <]$$

$$\frac{[[b]] = [[b_1\ AND\ b_2]]}{\langle \alpha, (and\ b_1\ b_2) \rangle \to \langle \alpha, b \rangle} \quad [AND]$$

$$\frac{l_1 = (v_i^{i \in 1..n}) \quad n \geq 1}{\langle \alpha, (cons\ v_0\ l_1) \rangle \to \langle \alpha, (v_i^{i \in 0..n}) \rangle} \quad [CONS1]$$

$$\frac{}{\langle \alpha, (cons\ v_1\ empty) \rangle \to \langle \alpha, (v_1) \rangle} \quad [CONS2]$$

$$\frac{t_1 \to t_1'}{\langle \alpha, (binop\ t_1\ t_2) \rangle \to \langle \alpha', (binop\ t_1'\ t_2) \rangle} \quad [BINOP1]$$

$$\frac{t_2 \to t_2'}{\langle \alpha, (binop\ v_1\ t_2) \rangle \to \langle \alpha', (binop\ v_1\ t_2') \rangle} \quad [BINOP2]$$

6.4 Semântica das operações unárias

$$\frac{[[b]] = [[not\ b_1]]}{\langle \alpha, (not\ b_1)\rangle \to \langle \alpha, b\rangle} \quad [NOT]$$

$$\frac{v_1 = empty}{\langle \alpha, (empty?\ v_1)\rangle \to \langle \alpha, true\rangle} \quad [EMPTY?1]$$

$$\frac{v_1 \neq empty}{\langle \alpha, (empty?\ v_1)\rangle \to \langle \alpha, false\rangle} \quad [EMPTY?2]$$

$$\frac{l_1 = (t_i^{i\in 1..n}) \quad n \geq 1}{\langle \alpha, (first\ l_1)\rangle \to \langle \alpha, t_1\rangle} \quad [FIRST1]$$

$$\frac{}{\langle \alpha, (first\ empty)\rangle \to \langle \alpha, empty\rangle} \quad [FIRST2]$$

$$\frac{l_1 = (t_i^{i\in 1..n}) \quad n > 1}{\langle \alpha, (rest\ l_1)\rangle \to \langle \alpha, (t_i^{i\in 2..n})\rangle} \quad [REST1]$$

$$\frac{}{\langle \alpha, (rest\ empty)\rangle \to \langle \alpha, empty\rangle} \quad [REST2]$$

$$\frac{l_1 = (t_1)}{\langle \alpha, (rest\ l_1)\rangle \to \langle \alpha, empty\rangle} \quad [REST3]$$

$$\frac{t_1 \to t_1'}{\langle \alpha, (unop\ t_1)\rangle \to \langle \alpha', (unop\ t_1')\rangle} \quad [UNOP]$$

6.5 Semântica das operações n-árias

$$\frac{n \geq 1}{\langle \alpha, (list\ v_i^{i\in 1..n})\rangle \to \langle \alpha, (v_i^{i\in 1..n})\rangle} \quad [LIST]$$

$$\frac{n \geq 1}{\langle \alpha, (begin\ v_i^{i\in 1..n})\rangle \to \langle \alpha, v_n\rangle} \quad [BEGIN]$$

$$\frac{t_k \to t_k' \qquad 1 \leq k \leq n}{\langle \alpha, (n\text{–}ary\ v_i^{i\in 1..k-1}\ t_i^{i\in k..n})\rangle \to \langle \alpha', (n\text{–}ary\ v_i^{i\in 1..k-1}\ t_k'\ t_i^{i\in k+1..n})\rangle} \quad [N-ARY]$$

6.6 Semântica das demais operações

$$\frac{}{\langle \alpha, (if\ true\ t_1\ t_2)\rangle \rightarrow \langle \alpha, t_1\rangle} \quad [IF1]$$

$$\frac{}{\langle \alpha, (if\ false\ t_1\ t_2)\rangle \rightarrow \langle \alpha, t_2\rangle} \quad [IF2]$$

$$\frac{t_1 \rightarrow t_1'}{\langle \alpha, (if\ t_1\ t_2\ t_3)\rangle \rightarrow \langle \alpha', (if\ t_1'\ t_2\ t_3)\rangle} \quad [IF3]$$

$$\frac{t_1 \rightarrow t_1'}{\langle \alpha, (define\ x\ t_1)\rangle \rightarrow \langle \alpha', (define\ x\ t_1')\rangle} \quad [DEFINE1]$$

$$\frac{}{\langle \alpha, (define\ x\ v_1)\rangle \rightarrow \langle \alpha, [x \mapsto v_1], v_1\rangle} \quad [DEFINE2]$$

$$\frac{\langle \alpha, (t_i^{i\in 1..n})\rangle \rightarrow ... \rightarrow \langle \alpha', (v_i^{i\in 1..n})\rangle \quad \langle \alpha', t_r\rangle \rightarrow ... \rightarrow \langle \alpha'', v_r\rangle}{\langle \alpha, (local\ (t_i^{i\in 1..n})\ t_r)\rangle \rightarrow \langle \alpha, v_r\rangle} \quad [LOCAL]$$

$$\frac{}{\langle \alpha, ((lambda{:}T_r\ (x_i{:}T_i^{i\in 1..n})\ t_r)\ v_i^{i\in 1..n})\rangle \rightarrow \langle \alpha, \{v_i/x_i\}^{i\in 1..n}\ t_r\rangle} \quad [APP1]$$

$$\frac{t_k \rightarrow t_k' \qquad 1 \leq k \leq n}{\langle \alpha, ((lambda{:}T_r\ (x_i{:}T_i^{i\in 1..n})\ t_r)\ v_i^{i\in 1..k-1}\ t_i^{i\in k..n})\rangle \rightarrow \langle \alpha, ((lambda{:}T_r\ (x_i{:}T_i^{i\in 1..n})\ t_r)\ v_i^{i\in 1..k-1}\ t_k'\ t_i^{i\in k+1..n})\rangle} \quad [APP2$$

7 Sistema de tipos

A seguir apresenta-se as regras de tipo da linguagem Esquema. Nas regras abaixo, é utilizado a definição de tipo genérico T, que se dá como:

$$T \ ::== \ int \ | \ bool \ | \ string \ | \ [\,] \ | \ T, ..., T \rightarrow T \ | \ [T]$$

Para definir o sistema de tipos, define-se um mapeamento Γ que mapeia um identificador, definido pelo usuário, a um tipo da linguagem. Toda vez que adiciona-se um identificador x ao contexto atual, $\Gamma(x)$ recebe o tipo de x.

7.1 Tipos das variáveis

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T} \quad [TVAR]$$

7.2 Tipos dos literais

$$\frac{[[n]] \in \mathbb{Z}}{\Gamma \vdash n : int} \quad [TINT]$$

$$\frac{[[b]] \in \{true, false\}}{\Gamma \vdash b : bool} \quad [TBOOL]$$

$$\frac{[[s]] \in conjunto\ de\ strings\ poss\acute{i}veis}{\Gamma \vdash s : string} \quad [TSTRING]$$

$$\frac{}{\Gamma \vdash empty : [\,]} \quad [TEMPTY]$$

7.3 Tipos das operações básicas

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash (+\ e_1\ e_2) : int} \quad [TOP+]$$

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash (-\ e_1\ e_2) : int} \quad [TOP-]$$

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash (*\ e_1\ e_2) : int} \quad [TOP*]$$

$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash (<\ e_1\ e_2) : bool} \quad [TOP <]$$

$$\frac{\Gamma \vdash e_1 : bool}{\Gamma \vdash (not\ e_1) : bool} \quad [TNOT]$$

$$\frac{\Gamma \vdash e_1 : bool \quad \Gamma \vdash e_2 : bool}{\Gamma \vdash (and\ e_1\ e_2) : bool} \quad [TAND]$$

$$\frac{}{\Gamma \vdash (empty?\ e_1) : bool} \quad [TEMPTY?]$$

7.4 Tipos das operações sobre listas

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : [T]}{\Gamma \vdash (cons\ e_1\ e_2) : [T]} \quad [TCONS1]$$

$$\frac{\Gamma \vdash e_1 : T}{\Gamma \vdash (cons\ e_1\ empty) : [T]} \quad [TCONS2]$$

$$\frac{\Gamma \vdash e_1 : [T]}{\Gamma \vdash (first\ e_1) : T} \quad [TFIRST1]$$

$$\frac{}{\Gamma \vdash (first\ empty) : [\,]} \quad [TFIRST2]$$

$$\frac{\Gamma \vdash e_1 : [T]}{\Gamma \vdash (rest\ e_1) : [T]} \quad [TREST1]$$

$$\frac{}{\Gamma \vdash (rest\ empty) : [\,]} \quad [TREST2]$$

$$\frac{\Gamma \vdash e_i : T \ \ \forall i \in 1..n \quad n \geq 1}{\Gamma \vdash (list\ e_i^{i \in 1..n}) : [T]} \quad [TLIST]$$

7.5 Tipos das demais operações

$$\frac{\Gamma \vdash e_1 : bool \quad e_2 : T \quad e_3 : T}{\Gamma \vdash (if\ e_1\ e_2\ e_3) : T} \quad [TIF]$$

$$\frac{\Gamma \vdash e_n : T \quad n \geq 1}{\Gamma \vdash (begin\ e_i^{i \in 1..n}) : T} \quad [TBEGIN]$$

$$\frac{\Gamma \vdash e_r : T \quad n \geq 1}{\Gamma \vdash (local\ (e_i^{i \in 1..n})\ e_r) : T} \quad [TLOCAL]$$

$$\frac{\Gamma \vdash e_1 : T}{\Gamma \vdash (define\ x\ e_1) : T} \quad [TDEFINE]$$

$$\frac{\Gamma \vdash e_i : T_i \quad \forall i \in 1..n}{\Gamma \vdash ((lambda{:}T_r\ (x_i{:}T_i^{i \in 1..n})\ e_r)\ e_i^{i \in 1..n}) : T_r} \quad [TAPP]$$

8 Conclusão

Utilizando-se as regras definidas acima, foi possível implementar a linguagem descrita sem grandes dificuldades. A linguagem mostrou-se bastante poderosa e fácil de ser usada, tal como a linguagem em que foi inspirada, Scheme.

Foi possível a implementação de, desde funções simples, como o fatorial, até funções mais complexas, como o *mergesort*. Além disso, por tratar funções como valores, permite utilizar abstrações poderosas, como funções de alta ordem, e definir funções como *map*, $fold$ e $filter$.

Tendo em vista os pontos citados acima, considera-se o trabalho, como um todo, bem-sucedido.