

Aula 8

Semânticas operacionais de linguagens funcionais

- Embora o λ -calculus seja um modelo de computação das linguagens de programação funcionais actuais é conveniente estudar os constructores destas linguagens usando modelos mais simples.
- Vamos começar por considerar a linguagem **SFUN** onde um programa é apenas um conjunto de equações que define funções recursivas.
- Para **SFUN** consideramos um sistema de tipos e um algoritmo de atribuição de tipo.
- Para programas tipificáveis daremos semânticas operacionais com chamada por nome e por valor.

SFUN - linguagem funcional (simples) com definições recursivas de funções

Sintaxe

$x \in \mathbf{Var}$ variáveis

$n \in \mathbb{Z}$ inteiros

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

$f_1, f_2, \dots, f_k \in \mathbf{Fvar}$ conjunto de símbolos de funções de aridade a_i , para $1 \leq i \leq k$.

A sintaxe abstracta das expressões é dada pelos seguintes *termos*, t

$$\begin{aligned} t ::= & x \mid n \mid b \mid t_1 \text{ op } t_2 \mid t_1 \text{ bop } t_2 \mid \neg t_1 \mid t_1 \wedge t_2 \mid \\ & \text{if } t_0 \text{ then } t_1 \text{ else } t_2 \mid f_i(t_1, \dots, t_{a_i}) \\ \text{op} ::= & + \mid - \mid \times \mid \div \\ \text{bop} ::= & < \mid = \mid > \end{aligned}$$

Notar que ao contrário das abstrações do λ -calculus as funções em **SFUN** tem uma aridade fixa e só podem ser avaliadas quando são dados todos os argumentos.

Declaração (recursiva) de funções

Um programa em **SFUN** é um conjunto de equações:

$$\begin{aligned} f_1(x_1, \dots, x_{a_1}) &= t_1 \\ &\vdots \\ f_k(x_1, \dots, x_{a_k}) &= t_k \end{aligned}$$

onde para cada $1 \leq i \leq k$,

- t_i são termos
- $Vars(t_i) \subseteq \{x_1, \dots, x_{a_i}\}$
- os f_i são distintos

Exemplo 8.1.

$s(x) = \text{if } x = 0 \text{ then } 0 \text{ else } f(x, 0 - x)$
 $f(x, y) = \text{if } x = 0 \text{ then } 1 \text{ else } (\text{if } y = 0 \text{ then } -1 \text{ else } f(x - 1, y - 1))$

Exemplo 8.2.

$$\begin{aligned} f_1 &= f_1 + 1 \\ f_2(x) &= 1 \\ f_3(x) &= x \times x \end{aligned}$$

Exemplo 8.3.

$$g_1(x) = g_1(x) + 1$$

Neste caso o valor de $g_1(3)$ seria o de $g_1(3) + 1$, que seria o de $(g_1(3) + 1) + 1$, etc.

Comparação com o Haskell/ *lambda-calculus*

- os padrões são só variáveis
- os argumentos são tuplos (pelo que não se pode usar a funções com falta de argumentos)
- não podemos ter funções locais

Sistema de tipos para SFUN

A gramática de **SFUN** permite construir termos como

$$5 \wedge \text{false}$$

que não tem significado. Impondo um sistema de tipos, expressões como esta não serão consideradas.

Pretendemos uma linguagem *fortemente tipificada*, i.e. só termos com tipo serão avaliados. Estes sistemas podem ser usados para verificação estática de termos e programas.

Conjunto de Tipos: Básicos bt e Gerais τ

$$bt ::= \text{int} \mid \text{bool}$$

$$\tau, \sigma ::= bt \mid (bt_1, \dots, bt_n) \rightarrow bt$$

Não há variáveis de tipo e nos tipos funcionais os argumentos têm de ser tipos básicos (bt) e todos definidos.

Exemplo 8.4. *Exemplos de tipos são:* $\text{int} \rightarrow \text{int}$, $(\text{int}, \text{int}) \rightarrow \text{int}$, $(\text{bool}, \text{bool}) \rightarrow \text{int}$, $(\text{bool}, \text{int}, \text{bool}) \rightarrow \text{bool}$, etc.

Atribuição de Tipo

- Uma expressão da forma $t : \tau$ é um **atribuição de tipo** que associa ao termo t o tipo τ .
- Um *contexto* ou *ambiente de variáveis* é um conjunto finito de atribuições de tipo a variáveis (de termos) distintas:

$$\Gamma = x_1 : \tau_1, \dots, x_n : \tau_n$$

$$\text{dom}(\Gamma) = \{x \mid x : \tau \in \Gamma\}$$

Se $x : \tau \in \Gamma$ escrevemos que $\Gamma(x) = \tau$.

- Um *ambiente de funções* atribui a cada símbolo de função f_i , com $a_i = \text{ar}(f_i)$, um tipo funcional:

$$\varepsilon(f_i) = (\sigma_1, \dots, \sigma_{a_i}) \rightarrow \sigma$$

- Um termo t é *bem-tipificado* se existem Γ e τ tal que

$$\Gamma \vdash_{\varepsilon} t : \tau$$

Regras do sistema de tipos para SFUN

$$\frac{}{\Gamma \vdash_{\varepsilon} b : \text{bool}} \text{ (b)}$$

$$\frac{}{\Gamma \vdash_{\varepsilon} n : \text{int}} \text{ (n)}$$

$$\frac{\Gamma(x) = \sigma}{\Gamma \vdash_{\varepsilon} x : \sigma} \text{ (var)}$$

$$\frac{\Gamma \vdash_{\varepsilon} t_1 : \text{int} \quad \Gamma \vdash_{\varepsilon} t_2 : \text{int}}{\Gamma \vdash_{\varepsilon} t_1 \text{ op } t_2 : \text{int}} \text{ (op)}$$

$$\frac{\Gamma \vdash_{\varepsilon} t_1 : \text{int} \quad \Gamma \vdash_{\varepsilon} t_2 : \text{int}}{\Gamma \vdash_{\varepsilon} t_1 \text{ bop } t_2 : \text{bool}} \text{ (bop)}$$

$$\frac{\Gamma \vdash_{\varepsilon} t_1 : \text{bool} \quad \Gamma \vdash_{\varepsilon} t_2 : \text{bool}}{\Gamma \vdash_{\varepsilon} t_1 \wedge t_2 : \text{bool}} \text{ (and)}$$

$$\frac{\Gamma \vdash_{\varepsilon} t : \text{bool}}{\Gamma \vdash_{\varepsilon} \neg t : \text{bool}} \text{ (not)}$$

$$\frac{\Gamma \vdash_{\varepsilon} t_0 : \text{bool} \quad \Gamma \vdash_{\varepsilon} t_1 : \sigma \quad \Gamma \vdash_{\varepsilon} t_2 : \sigma}{\Gamma \vdash_{\varepsilon} \text{if } t_0 \text{ then } t_1 \text{ else } t_2 : \sigma} \text{ (if)}$$

$$\frac{\Gamma \vdash_{\varepsilon} t_1 : \sigma_1 \cdots \quad \Gamma \vdash_{\varepsilon} t_{a_i} : \sigma_{a_i} \quad \varepsilon(f_i) = (\sigma_1, \dots, \sigma_{a_i}) \rightarrow \sigma}{\Gamma \vdash_{\varepsilon} f_i(t_1, \dots, t_{a_i}) : \sigma} \text{ (fn)}$$

Exemplos

- $x : \text{int} \vdash_{\varepsilon} x \times x : \text{int}$

$$\text{(var)} \frac{\frac{}{x : \text{int} \vdash_{\varepsilon} x : \text{int}} \quad \frac{}{x : \text{int} \vdash_{\varepsilon} x : \text{int}} \text{(var)}}{x : \text{int} \vdash_{\varepsilon} x \times x : \text{int}} \text{(op)}$$

- Se $\varepsilon(f_1) = \text{int}$ temos $\vdash_{\varepsilon} f_1 + 1 : \text{int}$

$$\text{(fn)} \frac{\frac{\varepsilon(f_1) = \text{int}}{\vdash_{\varepsilon} f_1 : \text{int}} \quad \frac{}{\vdash_{\varepsilon} 1 : \text{int}} \text{(n)}}{\vdash_{\varepsilon} f_1 + 1 : \text{int}} \text{(op)}$$

- $x : \text{int} \vdash_{\varepsilon} x = 0 : \text{bool}$

$$\frac{\frac{}{x : \text{int} \vdash_{\varepsilon} x : \text{int}} \text{(var)} \quad \frac{}{x : \text{int} \vdash_{\varepsilon} 0 : \text{int}} \text{(n)}}{x : \text{int} \vdash_{\varepsilon} x = 0 : \text{bool}} \text{(bop)}$$

- Se $\varepsilon(\text{fact}) = \text{int} \rightarrow \text{int}$ então $x : \text{int} \vdash_{\varepsilon} \text{fact}(x - 1) : \text{int}$

$$\text{(op)} \frac{\frac{\frac{}{x : \text{int} \vdash_{\varepsilon} x : \text{int}} \text{(var)} \quad \frac{}{x : \text{int} \vdash_{\varepsilon} 1 : \text{int}} \text{(n)}}{x : \text{int} \vdash_{\varepsilon} x - 1 : \text{int}} \text{(fn)} \quad \varepsilon(\text{fact}) = \text{int} \rightarrow \text{int}}{x : \text{int} \vdash_{\varepsilon} \text{fact}(x - 1) : \text{int}}$$

- e $x : \text{int} \vdash_{\varepsilon} \text{if } x = 0 \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) : \text{int}$, pois

$$\frac{x : \text{int} \vdash_{\varepsilon} x = 0 : \text{bool} \quad x : \text{int} \vdash_{\varepsilon} 1 : \text{int} \quad x : \text{int} \vdash_{\varepsilon} x \times \text{fact}(x - 1) : \text{int}}{x : \text{int} \vdash_{\varepsilon} \text{if } x = 0 \text{ then } 1 \text{ else } x \times \text{fact}(x - 1) : \text{int}} \text{(if)}$$

Tipificar Programas SFUN

Um programa P em **SFUN**

$$\begin{aligned} f_1(x_1, \dots, x_{a_1}) &= t_1 \\ &\vdots \\ f_k(x_1, \dots, x_{a_k}) &= t_k \end{aligned}$$

tem tipo (é tipificável) num ambiente de funções ε se para cada equação $f_i(x_1, \dots, x_{a_i}) = t_i$ se existir um tipo τ_i e um contexto Γ_i contendo atribuições de tipo para $x_1, \dots, x_{a_i}$ tal que

$$\Gamma_i \vdash_{\varepsilon} f_i(x_1, \dots, x_{a_i}) : \tau_i \text{ e } \Gamma_i \vdash_{\varepsilon} t_i : \tau_i.$$

Usando as regras anteriores podemos inferir um tipo para um programa em **SFUN**

Exemplos

Exemplo 8.5. Sendo $\varepsilon(f_1) = \text{int}$ e $\varepsilon(f_2) = \varepsilon(f_3) = \text{int} \rightarrow \text{int}$, o programa seguinte admite tipo:

$$\begin{aligned} f_1 &= f_1 + 1 \\ f_2(x) &= 1 \\ f_3(x) &= x \times x \end{aligned}$$

Temos que $\vdash f_1 : \text{int}$ (prova) e usando as deduções anteriores temos

$$\frac{\vdash_{\varepsilon} f_1 : \text{int} \quad \vdash_{\varepsilon} 1 : \text{int}}{\vdash_{\varepsilon} f_1 + 1 : \text{int}} (op)$$

Logo a primeira equação é tipificável.

Para a segunda $\vdash_{\varepsilon} 1 : \text{int}$ (lado direito da equação) e para o lado esquerdo:

$$\frac{x : \text{int} \vdash_{\varepsilon} x : \text{int}}{x : \text{int} \vdash_{\varepsilon} f_2(x) : \text{int}} (fn)$$

E para a terceira, já vimos que $x : \text{int} \vdash_{\varepsilon} x \times x : \text{int}$ e $x : \text{int} \vdash_{\varepsilon} f_3(x) : \text{int}$ (como o caso anterior). (Verifica!)

Exemplo 8.6. O programa

$$fact(x) = \text{if } x = 0 \text{ then } 1 \text{ else } x \times fact(x - 1)$$

é tipificável se $\varepsilon(fact) = \text{int} \rightarrow \text{int}$ dado que já vimos que

$$x : \text{int} \vdash_{\varepsilon} fact(x) : \text{int}$$

e

$$x : \text{int} \vdash_{\varepsilon} \text{if } x = 0 \text{ then } 1 \text{ else } x \times fact(x - 1) : \text{int}.$$

Unicidade de tipos

Teorema 8.1. *Para qualquer termo t de **SFUN**, contexto Γ e ambiente de símbolos funcionais ε , se $\Gamma \vdash_{\varepsilon} t : \tau$ e $\Gamma \vdash_{\varepsilon} t : \sigma$ então $\tau = \sigma$.*

Demonstração. Por indução na estrutura dos termos t . □

Podemos concluir que este sistema de tipos não é polimórfico. Para tal teríamos de associar variáveis de tipo e ambientes para essas variáveis.

Semântica operacional de SFUN

- Consideramos programas e termos tipificáveis
- Nos sistemas de transição as *configurações*
 - são termos fechados (sem variáveis livres), t
 - os valores (*configurações finais*) são constantes (inteiras ou Booleanas), v
- Para avaliarmos um termo fechado t temos de ter um programa P
- Consideramos duas semânticas (ambas *big-step*) correspondentes a estratégias de avaliação diferentes:
 - Chamada por valor** (*call-by-value*) ordem de redução aplicativa, sistema de transição $\Downarrow_P^v$
 - Chamada por nome** (*call-by-name*) ordem de redução normal, sistema de transições $\Downarrow_P^n$
- Se $t \Downarrow_P^v v$ ($t \Downarrow_P^n v$) dizemos que t avalia para v com a estratégia de avaliação por valor (por nome) usando o programa P .

Substituição de variáveis

Dado um programa P ,

$$\begin{aligned} f_1(x_1, \dots, x_{a_1}) &= d_1 \\ &\vdots \\ f_k(x_1, \dots, x_{a_k}) &= d_k \end{aligned}$$

Sendo $v_1, \dots, v_{a_i}$ termos denotamos por

$$d_i[x_1 \mapsto v_1, \dots, x_{a_i} \mapsto v_{a_i}]$$

o termo que se obtém substituindo (simultaneamente) em d_i cada ocorrência de x_j por v_j , para $1 \leq j \leq a_i$

Exemplo 8.7. Se $d = 2 \times x$ então $d[x \mapsto 5] = 2 \times 5$.

Semântica operacional (avaliação) com passagem de argumentos por valor

$$\frac{}{n \Downarrow_P^v n} \text{ (n)}$$

$$\frac{}{b \Downarrow_P^v b} \text{ (b)}$$

$$\frac{t_1 \Downarrow_P^v n_1 \quad t_2 \Downarrow_P^v n_2}{t_1 \text{ op } t_2 \Downarrow_P^v n_1 \text{ op } n_2} \text{ (op)}$$

$$\frac{t_1 \Downarrow_P^v n_1 \quad t_2 \Downarrow_P^v n_2}{t_1 \text{ bop } t_2 \Downarrow_P^v n_1 \text{ bop } n_2} \text{ (bop)}$$

$$\frac{t_1 \Downarrow_P^v b_1 \quad t_2 \Downarrow_P^v b_2}{t_1 \wedge t_2 \Downarrow_P^v b_1 \wedge b_2} \text{ (and)}$$

$$\frac{t \Downarrow_P^v b}{\neg t \Downarrow_P^v \neg b} \text{ (not)}$$

$$\frac{t_0 \Downarrow_P^v \text{ true} \quad t_1 \Downarrow_P^v v_1}{\text{if } t_0 \text{ then } t_1 \text{ else } t_2 \Downarrow_P^v v_1} \text{ (ifT)}$$

$$\frac{t_0 \Downarrow_P^v \text{ false} \quad t_2 \Downarrow_P^v v_2}{\text{if } t_0 \text{ then } t_1 \text{ else } t_2 \Downarrow_P^v v_2} \text{ (ifF)}$$

$$\frac{t_1 \Downarrow_P^v v_1 \quad \dots \quad t_{a_i} \Downarrow_P^v v_{a_i} \quad d_i[x_1 \mapsto v_1, \dots, x_{a_i} \mapsto v_{a_i}] \Downarrow_P^v v}{f_i(t_1, \dots, t_{a_i}) \Downarrow_P^v v} \text{ (fn)}$$

Notar que em *(op)*, *(bop)*, *(and)* e *(not)* os valores do lado direito são calculados usando a definição habitual de cada operação.

Exemplo 8.8. *Seja P o programa*

$$\begin{aligned} f_1 &= f_1 + 1 \\ f_2(x) &= 1 \\ f_3(x) &= x \times x \end{aligned}$$

Avalia na semântica $\Downarrow_P^v$

1. $f_2(0)$

2. $f_2(f_1)$

3. $f_3(2 + 1)$

1. Para $f_2(0)$ temos

$$\frac{\frac{}{0 \Downarrow_P^v 0} (n)}{f_2(0) \Downarrow_P^v 1} \quad \frac{\frac{}{1[x \mapsto 0] \Downarrow_P^v 1} (n)}{(fn)}$$

2. Para $f_2(f_1)$ não há avaliação, i.e não existe um valor v tal que $f_2(f_1) \Downarrow_P^v v$. Uma avaliação teria de usar a regra (fn) que requeria a avaliação de f_1 . Mas para isso usaria-se a regra (fn) que teria de requer a avaliação de f_1 para calcular $f_1 + 1$, isto leva a um processo infinito de avaliação.

3. Para $f_3(2 + 1)$ temos

$$\frac{\frac{\frac{}{2 \Downarrow_P^v 2} (n)}{2 + 1 \Downarrow_P^v 3} (op) \quad \frac{\frac{\frac{}{1 \Downarrow_P^v 1} (n)}{3 \Downarrow_P^v 3} (n) \quad \frac{\frac{}{3 \Downarrow_P^v 3} (n)}{3 \Downarrow_P^v 3} (op)}{x \times x[x \mapsto 3] \Downarrow_P^v 9} (op)}{f_3(2 + 1) \Downarrow_P^v 9} (fn)$$

onde $x \times x[x \mapsto 3] = 3 * 3$.

Exercício 8.1. Para o factorial definido como acima

$$fact(x) = \text{if } x = 0 \text{ then } 1 \text{ else } x \times fact(x - 1)$$

avalie $fact(0)$ para a semântica $\Downarrow_P^v$. $\diamond$

Propriedades de SFUN

Teorema 8.2 (Preservação do Tipo). *Se P é um programa tipificável no ambiente ε e t é um termo fechado então se t tem tipo σ , i.e. $\Gamma \vdash_\varepsilon t : \sigma$, e $t \Downarrow_P^v v$ então v tem tipo σ , i.e., $\Gamma \vdash_\varepsilon v : \sigma$.*

Demonstração. Por indução usando o lema seguinte. □

Casos base.

Caso n : Se t é um inteiro n então $\vdash_\varepsilon n : \text{int}$ e como $n \Downarrow_P^v n$ (axioma (n)) obviamente $\vdash_\varepsilon n : \text{int}$.

Caso b : Se t é um booleano b então $\vdash_\varepsilon b : \text{bool}$ e como $b \Downarrow_P^v b$ (axioma (b)) obviamente $\vdash_\varepsilon b : \text{bool}$.

Passo de Indução Supor que é válido para as premissas provar para a conclusão da regra.

- Se t tem um operador como raiz as regras que se podem aplicar são (op) , (bop) , (and) ou (not) . Iremos apenas considerar o caso em que t é $t_1 + t_2$. Nesta caso t , t_1 e t_2 têm tipo int , i.e. $\Gamma \vdash_\varepsilon t : \text{int}$, $\Gamma \vdash_\varepsilon t_i : \text{int}$ para $i = 1, 2$. Pela hipótese de indução se $t_i \Downarrow_P^v v_i$, então $\Gamma \vdash_\varepsilon v_i : \text{int}$ e pela regra (op) , $\Gamma \vdash_\varepsilon v_1 + v_2 : \text{int}$.
- Se t é um condicional **if** t_0 **then** t_1 **else** t_2 com tipo σ , então t_0 tem tipo booleano ($\Gamma \vdash_\varepsilon t_0 : \text{bool}$) e t_1 e t_2 têm o mesmo tipo σ , i.e. $\Gamma \vdash_\varepsilon t_i : \sigma$. Pela indução, se $t_0 \Downarrow_P^v v_0$, $t_1 \Downarrow_P^v v_1$ e $t_2 \Downarrow_P^v v_2$ os tipos dos v_j são os mesmos, i.e. $\Gamma \vdash_\varepsilon v_0 : \text{bool}$ e $\Gamma \vdash_\varepsilon v_i : \sigma$. Se v_0 é **true** podemos aplicar a regra (ifT) e o valor de t é v_1 , logo tem o mesmo tipo de t . Senão o valor de v_0 é **false** a regra a aplica é (ifF) , sendo o valor de t , v_2 e concluímos do mesmo modo.
- Se t é uma aplicação duma função $f_i(t_1, \dots, t_{a_i})$, suponhamos que $\Gamma \vdash_\varepsilon f_i(t_1, \dots, t_{a_i}) : \sigma$. Pela regra (fn) sabemos que $\Gamma \vdash_\varepsilon t_j : \sigma_j$ e $\varepsilon(f_i) = (\sigma_1, \dots, \sigma_{a_i}) \rightarrow \sigma$. Como P é bem tipificado também temos que $x_j : \sigma_j \vdash_\varepsilon f_i(x_1, \dots, x_{a_i}) : \sigma$, e $x_j : \sigma_j \vdash_\varepsilon d_i : \sigma$. Por outro lado, $f_i(t_1, \dots, t_{a_i}) \Downarrow_P^v v$ sse $t_j \Downarrow_P^v v_j$ para $1 \leq j \leq a_i$ e $d_i[x_1 \mapsto v_1, \dots, x_{a_i} \mapsto v_{a_i}] \Downarrow_P^v v$. Pela hipótese de indução, $\Gamma \vdash_\varepsilon v_j : \sigma_j$ e pelo Lema da substituição, concluímos que $\vdash_\varepsilon d_i[x_j \mapsto v_j] : \sigma$. Mais uma vez por indução concluímos que $\Gamma \vdash_\varepsilon v : \sigma$, como queríamos.

Lema 8.1 (Substituição). *Se $x_1 : \sigma_1, \dots, x_n : \sigma_n \vdash_\varepsilon t : \tau$ e $t_1, \dots, t_n$ são termos fechados tal que $\vdash_\varepsilon t_i : \sigma_i$ para $1 \leq i \leq n$, então $\vdash_\varepsilon t[x_i \mapsto t_i] : \tau$.*

Determinismo e Tipificação Forte

A um termo fechado apenas um valor pode ser associado e se o termo é bem-tipificado a avaliação por passagem de argumentos por valor não pode produzir erros (tipificação forte)

Teorema 8.3. *A avaliação de um termo t fechado e tipificável para um programa P não pode produzir erros. E se $t \Downarrow_P^v v_1$ e $t \Downarrow_P^v v_2$ então $v_1 = v_2$.*

Demonstração

Por indução no conjunto de regras de $\Downarrow_P^v$.

Casos base.

Caso n : Se t é um inteiro n então $n \Downarrow_P^v n$ (axioma (n)) e esta é a única regra que se pode aplicar

Caso b : Se t é um booleano b então $b \Downarrow_P^v b$ (axioma (b)) e esta é a única regra que se pode aplicar

Nestes casos a avaliação termina e só pode haver uma solução. Como t é fechado não pode ser uma variável. *Passo de Indução* Supor que é válido para as premissas provar para a conclusão da regra.

- Se t tem um operador como raiz as regras que se podem aplicar são (op) , (bop) , (and) ou (not) . Iremos apenas considerar o caso em que t é $t_1 + t_2$. Nesta caso t , t_1 e t_2 têm tipo int . Pela hipótese de indução a $t_i \Downarrow_P^v v_i$, não pode trazer erros de tipos, v_i têm tipo int e são únicos. Assim só pode haver um valor $v_1 + v_2$ e tem tipo int .
- Se t é um condicional $\text{if } t_0 \text{ then } t_1 \text{ else } t_2$ então t_0 tem tipo booleano e t_1 e t_2 têm o mesmo tipo σ . Pela indução, a avaliação de t_0 , t_1 e t_2 não produz erro de tipos e cada um tem apenas um valor v_0 , v_1 e v_2 (respectivamente). Como a avaliação preserva o tipo de v_0 é uma constante Booleana. Se for true podemos aplicar a regra (ifT) e o valor de t é v_1 . Senão o valor de v_0 é false a regra a aplica é (ifF) , sendo o valor de t , v_2 . Em ambos os casos o valor está univocamente determinado.
- Se t é uma aplicação duma função $f_i(t_1, \dots, t_{a_i})$ então usando a regra (fn) temos

$$f_i(t_1, \dots, t_{a_i}) \Downarrow_P^v v$$

sse $t_j \Downarrow_P^v v_j$ para $1 \leq j \leq a_i$ e $d_i[x_1 \mapsto v_1, \dots, x_{a_i} \mapsto v_{a_i}] \Downarrow_P^v v$. Pela hipótese de indução, existe no máximo um valor v_j para cada t_j e um único valor v . Logo o valor de $f_i(t_1, \dots, t_{a_i})$ é unicamente determinado. Falta ver que não se produzem erros de tipo. A avaliação dos t_j não produz erros de tipos. Temos que mostrar que $d_i[x_1 \mapsto v_1, \dots, x_{a_i} \mapsto v_{a_i}]$ admite tipo. Como $f_i(t_1, \dots, t_{a_i})$ é tipificável, os tipos de t_j têm de ser compatíveis com os da declaração $\varepsilon(f_i) = (\sigma_1, \dots, \sigma_{a_i}) \rightarrow \sigma$. Logo t_j tem tipo σ_j , assim como v_j para $1 \leq j \leq a_i$. Como o programa P é bem-tipificado d_i tem tipo σ , i.e.

$$x_1 : \sigma_1, \dots, x_{a_i} : \sigma_{a_i} \vdash_\varepsilon d_i : \sigma$$

Pelo Lema da Substituição 8.1 deduzimos que $\vdash_\varepsilon d_i[x_1 \mapsto v_1, \dots, x_{a_i} \mapsto v_{a_i}] : \sigma$. Pela hipótese de indução, concluímos que temos um termo fechado que admite tipo e a avaliação não pode produzir erro de tipos.

Semântica operacional (avaliação) com passagem de argumentos por nome

Apenas temos de alterar a regra (fn) considerando que todas as outras regras do sistema de transição $\Downarrow_P^n$ coincidem com as do sistema $\Downarrow_P^v$. Para (fn_N) temos

$$\frac{d_i[x_1 \mapsto t_1, \dots, x_{a_i} \mapsto t_{a_i}] \Downarrow_P^n v}{f_i(t_1, \dots, t_{a_i}) \Downarrow_P^n v} (fn_N)$$

Neste caso substituímos as variáveis x_j pelos termos t_j em vez dos seus valores após avaliação.

Determinismo e Tipificação Forte

A um termo fechado apenas um valor pode ser associado e se o termo é bem-tipificado a avaliação por passagem de argumentos por nome não pode produzir erros (tipificação forte)

Teorema 8.4. *A avaliação de um termo t fechado e admite tipo para um programa P não pode produzir erros. E se $t \Downarrow_P^n v_1$ e $t \Downarrow_P^n v_2$ então $v_1 = v_2$.*

A demonstração é idêntica à da avaliação por passagem de argumentos por valor.

Exemplo 8.9. *Seja P o programa*

$$\begin{aligned} f_1 &= f_1 + 1 \\ f_2(x) &= 1 \\ f_3(x) &= x \times x \end{aligned}$$

Avalia na semântica $\Downarrow_P^n$

1. $f_2(0)$
2. $f_2(f_1)$
3. $f_3(2 + 1)$

1. Para $f_2(0)$ temos

$$\frac{\frac{1[x \mapsto 0] \Downarrow_P^n 1}{f_2(0) \Downarrow_P^n 1} (n)}{(fn_N)}$$

2. Para $f_2(f_1)$, neste caso há avaliação dado que o argumento de f_2 não é avaliado. Esta é uma propriedade importante desta estratégia: se o termo tiver um valor ele é descoberto com esta estratégia.

$$\frac{\frac{1[x \mapsto f_1] \Downarrow_P^n 1}{f_2(f_1) \Downarrow_P^n 1} (n)}{(fn_N)}$$

3. Para $f_3(2 + 1)$ temos

$$\frac{\frac{\frac{2 \Downarrow_P^n 2}{2 + 1 \Downarrow_P^n 3} (n) \quad \frac{1 \Downarrow_P^n 1}{(op)} (n)}{\frac{x * x[x \mapsto 2 + 1] \Downarrow_P^n 9}{f_3(2 + 1) \Downarrow_P^n 9} (fn)} \quad \frac{\frac{2 \Downarrow_P^n 2}{2 + 1 \Downarrow_P^n 3} (n) \quad \frac{1 \Downarrow_P^n 1}{(op)} (n)}{(op)}$$

Avaliação *lazy*

No exemplo anterior vimos duas características da avaliação por passagem de argumentos por nome:

- Se houver avaliação de um termo ela é obtida por esta estratégia
- Como $x * x[x \mapsto 2 + 1] = (2 + 1) * (2 + 1)$, subtermos iguais poderão ter de ser avaliados diversas vezes.
- Para evitar isto pode-se tentar fazer partilha de subtermos iguais.
- A esta estratégia dá-se o nome de avaliação *lazy*
- Os termos são representados por grafos onde uma mesma variável só ocorre uma vez.

Referências

- [1] Maribel Fernández. *Programming Languages and Operational Semantics: a concise overview*. Springer, 2014.
- [2] Thérèse Hardin, Mathieu Jaume, François Pessaux, and Véronique Viguié Donzeau-Gouge. *Concepts and Semantics of Programming Languages 1: A Semantical Approach with OCaml and Python*. Wiley, 2021.
- [3] Robert W. Sebesta. *Concepts of Programming Languages*. Pearson, 11th edition, 2016.