INF05516 - Semântica Formal N
Ciência da Computação
Universidade Federal do Rio Grande do Sul

Álvaro Moreira, Marcus Ritt
{afmoreira,mrpritt}@inf.ufrgs.br

31 de Março de 2010

Conteúdo

1 Semântica Operacional e Sistemas de Tipos

Vamos definir a semântica operacional de uma série de linguagens no estilo conhecido por *semântica operacional estrutural* chamado também de *semântica operacional small-step*.
O material dessas notas de aulas foi elaborado com base nas notas de aula de Peter Sewell, Universidade de Cambridge (parte sobre as linguagens L1, L2 e L3) e no livro Types and Programming Languages de Benjamin Pierce (parte sobre exceções, subtipos e orientação a objetos).

1.1 A Linguagem L1

Programas em L1 pertencem ao conjunto definido pela gramática abstrata abaixo:

Sintaxe de L1 (1)

$$
\begin{array}{rcl}
e & ::= & n \mid b \mid e_1 \ op \ e_2 \mid \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \\
& \mid & l\texttt{:=}\ e \mid \texttt{!}\ l \\
& \mid & \texttt{skip} \mid e_1\texttt{;}\ e_2 \\
& \mid & \texttt{while}\ e_1\ \texttt{do}\ e_2
\end{array}
$$

onde

$$
\begin{array}{rcl}
n & \in & \textit{conjunto de numerais inteiros} \\
b & \in & \{\texttt{true}, \texttt{false}\} \\
op & \in & \{+, \geq\} \\
l & \in & \textit{conjunto de endereços}
\end{array}
$$

Observações:

- em L1 não há distinção entre comandos e expressões.
- note que, de acordo com a gramática abstrata acima, fazem parte do conjunto de árvores de sintaxe abstrata de L1 expressões sem sentido tais como 10+`false` e também `while` 2+ 1 `do` 20
- em L1 o programador tem acesso direto a endereços (mas não há aritmética com endereços).

1.1.1 Semântica Operacional de L1

Vamos definir a semântica operacional de L1 no estilo conhecido por *semântica operacional estrutural* chamado também de *semântica operacional small-step*.

Relação de transição entre configurações (2)

- Uma semântica operacional *small-step* é um *sistema de transição de estados*
- A relação de transição entre estados é chamada de $\longrightarrow$
- Escrevemos

$$c \longrightarrow c'$$

  para dizer que há uma transição do estado c para o estado c'
- A relação $\longrightarrow^*$ é o fecho reflexivo e transitivo de $\longrightarrow$
- Escrevemos $c \not\longrightarrow$ quando não existe c' tal que $c \longrightarrow c'$

Semântica Operacional de L1 - Configurações (3)

- Um estado, para L1, é um par $\langle e,\ \sigma \rangle$ onde e é uma expressão e σ é uma *memória*.
- Uma memória é um mapeamento **finito** de endereços para inteiros. Exemplo:

$$\sigma = \{l_1 \mapsto 0, l_2 \mapsto 10\}$$

- Para memória σ acima, temos que $Dom(\sigma) = \{l_1, l_2\}$ e $\sigma(l_1) = 0$ e $\sigma(l_2) = 10$

Valores e Erros de Execução (4)

- Valores são expressões já completamente avaliadas
- Os valores da linguagem L1 são dados pela seguinte gramática:

$$v ::= n \mid b \mid \texttt{skip}$$

- Se $\langle e,\ \sigma \rangle \not\longrightarrow$ e a expressão e *não* é valor temos um *erro de execução*
- Após vermos as regras da semântica operacional de L1 veremos que, por exemplo
  - $\langle 2\texttt{+true},\ \sigma \rangle \not\longrightarrow$
  - $\langle l\texttt{:=}2,\ \sigma \rangle \not\longrightarrow$, caso $l \notin Dom(\sigma)$

A relação de transição $\longrightarrow$ é definida através de um conjunto de regras de inferência da forma

$$\frac{premissa \ldots premissa}{conclusao}$$

Ajuda na compreensão das regras da semântica operacional *small step* lembrar que, tipicamente, para cada construção da gramática abstrata que não é valor, temos:

- uma ou mais regras de *reescrita* (ou redução) e
- uma ou mais regras de *computação*

As regras de reescrita especificam a ordem na qual uma expressão é avaliada e as regras de computação dizem, de fato, como uma determinada expressão efetua uma computação interessante.

Semântica Operacional de Operações Básicas (5)

$$\frac{[\![n]\!] = [\![n_1 + n_2]\!]}{\langle n_1 + n_2,\ \sigma\rangle \longrightarrow \langle n,\ \sigma\rangle} \quad (\text{OP}+)$$

$$\frac{[\![b]\!] = [\![n_1 \geq n_2]\!]}{\langle n_1 \geq n_2,\ \sigma\rangle \longrightarrow \langle b,\ \sigma\rangle} \quad (\text{OP}\geq)$$

$$\frac{\langle e_1,\ \sigma\rangle \longrightarrow \langle e_1',\ \sigma'\rangle}{\langle e_1\ op\ e_2,\ \sigma\rangle \longrightarrow \langle e_1'\ op\ e_2,\ \sigma'\rangle} \quad (\text{OP}1)$$

$$\frac{\langle e_2,\ \sigma\rangle \longrightarrow \langle e_2',\ \sigma'\rangle}{\langle v\ op\ e_2,\ \sigma\rangle \longrightarrow \langle v\ op\ e_2',\ \sigma'\rangle} \quad (\text{OP}2)$$

As regras OP1 e OP2 acima são regras de reescrita, e as regras OP+ e OP$\geq$ são regras de computação. Observe que as regras OP1 e OP2 especificam que a avaliação dos operandos é feita da esquerda para direita. Observe também o uso das meta-variáveis n_1 e n_2 nas regras OP+ e OP$\geq$: as regras se aplicam caso os operandos de + e $\geq$ sejam ambos números inteiros, caso contrário temos um erro de execução.

Condicional (6)

$$\langle \texttt{if true then } e_2 \texttt{ else } e_3,\ \sigma\rangle \longrightarrow \langle e_2,\ \sigma\rangle \quad (\text{IF}1)$$

$$\langle \texttt{if false then } e_2 \texttt{ else } e_3,\ \sigma\rangle \longrightarrow \langle e_3,\ \sigma\rangle \quad (\text{IF}2)$$

$$\frac{\langle e_1,\ \sigma\rangle \longrightarrow \langle e_1',\ \sigma'\rangle}{\langle \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3,\ \sigma\rangle \longrightarrow \langle \texttt{if}\ e_1'\ \texttt{then}\ e_2\ \texttt{else}\ e_3,\ \sigma'\rangle} \quad (\textsc{if}3)$$

As regras IF1 e IF2 acima são regras de computação para o condicional, e a regra IF3 é uma regra de reescrita.

Seqüência (7)

$$\langle \texttt{skip;}e_2,\ \sigma\rangle \longrightarrow \langle e_2,\ \sigma\rangle \quad (\textsc{seq}1)$$

$$\frac{\langle e_1,\ \sigma\rangle \longrightarrow \langle e_1',\ \sigma'\rangle}{\langle e_1\texttt{;}e_2,\ \sigma\rangle \longrightarrow \langle e_1'\texttt{;}e_2,\ \sigma'\rangle} \quad (\textsc{seq}2)$$

Pela regra de reescrita SEQ2 acima, a avaliação seqüencial de duas expressões é feita da esquerda para direita. Pela regra SEQ1, quando o lado esquerdo estiver completamente reduzido para `skip`, a avaliação deve continuar com a expressão no lado direito do ponto e vírgula.
Note que aqui foi feita uma escolha arbritrária no projeto da linguagem: a avaliação continua com a expressão no lado direito somente se a expressão do lado esquerdo do ponto e vírgula avalia para `skip`. Essa opção faz com que expressões sintaticamente bem formadas, como $5 + 4\texttt{;}\ l\texttt{:=}\ 4$ por exemplo, levem a erro de execução, caso sejam avaliadas.

Operações com a Memória (8)

$$\frac{l \in Dom(\sigma)}{\langle l\texttt{:=}\ n,\ \sigma\rangle \longrightarrow \langle \texttt{skip},\ \sigma[l \mapsto n]\rangle} \quad (\textsc{atr}1)$$

$$\frac{\langle e,\ \sigma\rangle \longrightarrow \langle e',\ \sigma'\rangle}{\langle l\texttt{:=}\ e,\ \sigma\rangle \longrightarrow \langle l\texttt{:=}\ e',\ \sigma'\rangle} \quad (\textsc{atr}2)$$

$$\frac{l \in Dom(\sigma)\ \ \sigma(l) = n}{\langle \texttt{!}\ l,\ \sigma\rangle \longrightarrow \langle n,\ \sigma\rangle} \quad (\textsc{deref})$$

Observe que as operações de atribuição e derreferência (regras ATR1 e DEREF) somente são executadas se o endereço l estiver mapeado na memória (formalizado pela premissa $l \in Dom(\sigma)$).

Na regra ATR1 podemos observar outra escolha feita em relaçao a semântica da linguagem: uma atribuição é reduzida para o valor `skip`. Em algumas linguagens de programação o valor final de uma expressão de atribuição é mesmo do valor atribuido.

While (9)

$$\langle \texttt{while}\ e_1\ \texttt{do}\ e_2,\ \sigma\rangle \longrightarrow \langle \texttt{if}\ e_1\ \texttt{then}\ (e_2\texttt{;while}\ e_1\ \texttt{do}\ e_2)\ \texttt{else skip},\ \sigma\rangle \ (\text{WHILE})$$

Note que a regra para o comando while não se encaixa na classificação das regras dada anteriormente (regras de reescrita e computação).

1.1.2 Sistema de Tipos para L1

Observe que a gramática para L1 admite expressões cuja execução leva erro (um erro aqui é representado pela impossibilidade de aplicar uma regra da semântica operacional para uma expressão que não é valor). Vamos ver um sistema de tipos que especifica uma análise estática a ser feita sobre árevores de sintaxe abstrata de expressões. Somente expressões consideradas *bem tipadas* por essa análise serão avaliadas.
Um sistema de tipos deve ser definido em acordo com a semântica operacional, em outras palavras, uma expressão só deve ser considerada bem tipada pelas regras de um sistema de tipos se a sua avaliação, pelas regras da semântica operacional, não levar a erro de execução. Essa propriedade fundamental é conhecida como *segurança* do sistema de tipos em relação a semântica operacional.

Tipos para L1 (10)

- o sistema de tipos de L1 consiste de um conjunto de regras de inferência

$$\frac{premissa \ldots premissa}{conclusao}$$

- Premissas e conclusão são da forma $\Delta \vdash e : T$ onde:
  - Δ é um mapeamento de endereços para seus tipos (int ref)
  - e é uma expressão da linguagem, e
  - T é um tipo pertencente ao conjunto definido pela seguinte gramática

$$T \ ::= \ \mathsf{int} \mid \mathsf{bool} \mid \mathsf{unit}$$

Note que em L1 a memória só pode receber valores inteiros, logo todos os endereços em L1 são do tipo int ref. Segue abaixo as regras do sistema de tipos. Há uma regra para cada cláusula da gramática de expressões.

Valores e Operações Básicas (11)

$$\Delta \vdash n : \mathsf{int} \qquad (\textsc{Tint})$$

$$\Delta \vdash b : \mathsf{bool} \qquad (\textsc{Tbool})$$

$$\frac{\Delta \vdash e_1 : \mathsf{int} \qquad \Delta \vdash e_2 : \mathsf{int}}{\Delta \vdash e_1 + e_2 : \mathsf{int}} \qquad (\mathrm{T}+)$$

$$\frac{\Delta \vdash e_1 : \mathsf{int} \qquad \Delta \vdash e_2 : \mathsf{int}}{\Delta \vdash e_1 \geq e_2 : \mathsf{bool}} \qquad (\mathrm{T}\geq)$$

Observe pelas regras OP+ e OP≥ que os operandos de + e de ≥ devem ser do tipo inteiro. Estas regras excluem expressões tais como $4 + \texttt{true}$ e $\texttt{true} \geq \texttt{skip}$.

Condicional (12)

$$\frac{\Delta \vdash e_1 : \mathsf{bool} \qquad \Delta \vdash e_2 : T \qquad \Delta \vdash e_3 : T}{\Delta \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : T} \qquad (\textsc{Tif})$$

- Note que para o condicional ser bem tipado a expressão da parte `then` e a expressão da parte `else` devem ser do mesmo tipo
- Note que, com essa regra, expressões tais como $\texttt{if}\ 5 + 3 \geq 2\ \texttt{then true else}\ 5$ não são consideradas bem tipadas mesmo que não levem a erro de execução, de acordo com a semântica operacional. Podemos dizer que o sistema de tipos está sendo muito conservador e recusando mais expressões do que deveria
- Isso acontece pois o sistema de tipos especifica uma análise estática

Operações com a memória (13)

$$\frac{\Delta \vdash e : \mathsf{int} \qquad \Delta(l) = \mathsf{int\ ref}}{\Delta \vdash l\texttt{:=}e : \mathsf{unit}} \qquad (\textsc{Tatr})$$

$$\frac{\Delta(l) = \mathsf{int\ ref}}{\Delta \vdash \texttt{!}\ l : \mathsf{int}} \quad (\textsc{Tderef})$$

Seqüência e While (14)

$$\Delta \vdash \texttt{skip} : \mathsf{unit} \quad (\textsc{Tskip})$$

$$\frac{\Delta \vdash e_1 : \mathsf{unit} \qquad \Delta \vdash e_2 : T}{\Delta \vdash e_1 \texttt{;} e_2 : T} \quad (\textsc{Tseq})$$

$$\frac{\Delta \vdash e_1 : \mathsf{bool} \qquad \Delta \vdash e_2 : \mathsf{unit}}{\Delta \vdash \texttt{while}\ e_1\ \texttt{do}\ e_2 : \mathsf{unit}} \quad (\textsc{Twhile})$$

Observe que o tipo unit é reservado para expressões que são avaliadas mais pelo seu efeito. O tipo unit possui somente um valor que é o `skip`.

1.1.3 Propriedades de L1

Propriedades (15)

O teorema abaixo expressa que a avaliação, **em um passo**, é determinística

Teorema 1 (Determinismo) Se $\langle e,\ \sigma\rangle \longrightarrow \langle e',\ \sigma'\rangle$ e se $\langle e,\ \sigma\rangle \longrightarrow \langle e'',\ \sigma''\rangle$ então $\langle e',\ \sigma'\rangle = \langle e'',\ \sigma''\rangle$.

Prova. Por indução na estrutura de e. ■

A partir do teorema acima concluimos que a avaliação de programas L1 é determinística.

Na seção anterior vimos que é fundamental que um sistema de tipos seja seguro em relação a semântica operacional da linguagem. A noção de segurança foi então explicada de maneira informal:

Segurança (16)

- Um sistema de tipos é seguro se expressões consideradas bem tipadas pelas suas regras não levam a **erro de execução** quando avaliadas de acordo com as regras da semântica operacional

- Erro de execução o ocorre quando temos uma configuração não final a qual nenhuma regra da semântica operacional se aplica
- De maneira **informal** podemos resumir essa noção de segurança através do seguinte *slogan*:

$$Se\ \Delta \vdash e : T\ então\ e \not\longrightarrow^*\quad erro$$

A técnica de prova mais utilizada para provar que um sistema de tipos é seguro é conhecida como *segurança sintática*. Ela consiste em realizar basicamente duas provas:

Segurança Sintática (17)

- prova do ***progresso de expressões bem tipadas***, ou seja, provar que, se uma expressão for bem tipada, e se ela não for um valor, sua avaliação pode progredir um passo pelas regras da semântica operacional. *Slogan*:

$$Se\ \Delta \vdash e : T\ então\ e\ é\ valor,\ ou\ existe\ e'\ tal\ que\ e \longrightarrow e'$$

- prova da ***preservação, após um passo da avaliação, do tipo de uma expressão***, ou seja, se uma expressão bem tipada progride em um passo, a expressão resultante possui o mesmo tipo da expressão original. *Slogan*:

$$Se\ \Delta \vdash e : T\ e\ e \longrightarrow e'\ então\ \Delta \vdash e' : T.$$

Note que ambas as provas são necessárias para provar segurança, ou seja:

$$Segurança = Progresso + Preservação$$

Provar somente *progresso* não é suficiente para provar segurança. É preciso provar que a expressão que resulta da progressão em um passo de uma expressão bem tipada também é bem tipada (ou seja, que a propriedade de ser bem tipado é preservada pela avaliação em um passo). Da mesma forma, provar somente *preservação* não é suficiente para provar segurança. É preciso provar que a expressão bem tipada que resulta da progressão em um passo da expressão original pode progredir (ou seja, é preciso provar progresso em um passo de expressões bem tipadas).
Observe que os *slogans* acima capturam a *essência* de progresso e preservação válida para qualquer linguagem de programação. Seguem abaixo as formulações precisas de progresso e preservação *especificas* para a linguagem L1.

Progresso e Preservação para L1 (18)

Teorema 2 (Progresso) Se $\Delta \vdash e : T$ e $Dom(\Delta) \subseteq Dom(\sigma)$ então (i) e é valor, ou (ii) existe $\langle e',\ \sigma' \rangle$ tal que $\langle e,\ \sigma \rangle \longrightarrow \langle e',\ \sigma' \rangle$

Prova. Por indução na estrutura de e. ■

Teorema 3 (Preservação) Se $\Delta \vdash e : T$ e $Dom(\Delta) \subseteq Dom(\sigma)$ e $\langle e,\ \sigma \rangle \longrightarrow \langle e',\ \sigma' \rangle$ então $\Delta \vdash e' : T$ e $Dom(\Delta) \subseteq Dom(\sigma')$

Prova. Por indução na estrutura de e. ■

Estamos interessados em saber se os dois problemas abaixo são decidíveis, ou seja, se existem algoritmos que os resolvem.

Problemas algoritmicos (19)

- Problema da **Verificação de Tipos**: dados ambiente Δ, expressão e e tipo T, o julgamento de tipo $\Delta \vdash e : T$ é derivável usando as regras do sistema de tipos?
- Problema da **Tipabilidade**: dados ambiente Δ e expressão e, encontrar tipo T tal que $\Delta \vdash e : T$ é derivável de acordo com as regras do sistema de tipos

O problema da tipabilidade é mais difícil do que o problema da verificação de tipos para sistemas de tipos de linguagens de programação. Dependendo do sistema de tipos, resolver o problema da tipabilidade requer algoritmos de *inferência de tipos* muitas vezes complicados. No caso da linguagem L1 há algoritmos simples para ambos os problemas
Observe que, do ponto de vista prático, o problema da *verificação de tipos* para L1 não é interessante[1]. Já o problema da *tipabilidade* é relevante na prática para L1: dado um programa L1 queremos saber se ele é ou não bem tipado e, ser for, queremos saber qual é o seu tipo .

Algoritmo de Inferência de Tipos (20)

Teorema 4 Dados ambiente Δ e expressão e, existe algoritmo que resolve o problema da tipabilidade para L1.

Prova. A prova consiste em exibir um algoritmo (ver exercício abaixo) que tem como entrada um ambiente de tipo Γ e uma expressão e e que: (i) termina sempre sua execução e que (ii) retorne um tipo T se e somente se $\Delta \vdash e : T$ é derivável de acordo com o sistema de tipos para L1 ■

Observação: Embora a diferença entre os dois problemas acima (da *verificação* de tipos e *tipabilidade*) seja clara, é bem comum se referir ao programa que os resolve como sendo o *verificador* de tipos para a linguagem.

[1] fora alguns exercícios de verificação de tipos a serem feitos

1.2 A Linguagem L2

A linguagem L2 é uma extensão de L1 com funções, aplicação, variáveis, funções recursivas e declarações.

Sintaxe de L2 (21)

Primeiro, devemos estender a sintaxe de L1:

$$
\begin{array}{lcl}
e & ::= & n \mid b \mid e_1\ op\ e_2 \mid \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \\
 & \mid & l\texttt{:=}\ e \mid \texttt{!}\ l \\
 & \mid & \texttt{skip} \mid e_1\texttt{;}\ e_2 \\
 & \mid & \texttt{while}\ e_1\ \texttt{do}\ e_2 \\
(*) & \mid & fn\ x\texttt{:}T \Rightarrow e \mid e_1\ e_2 \mid x \\
(*) & \mid & \texttt{let}\ x\texttt{:}T = e_1\ \texttt{in}\ e_2\ \texttt{end} \\
(*) & \mid & \texttt{let rec}\ f\texttt{:}T_1 \rightarrow T_2 = (fn\ y\texttt{:}T_1 \Rightarrow e_1)\ \texttt{in}\ e_2\ \texttt{end} \\
 & & \\
v & ::= & n \mid b \mid \texttt{skip} \\
(*) & \mid & fn\ x\texttt{:}T \Rightarrow e
\end{array}
$$

Na gramática acima:

- x representa um elemento pertencente ao conjunto $Ident$ de identificadores
- $fn\ x\texttt{:}T \Rightarrow e$ é uma função (sem nome)
- $e_1\ e_2$ é a aplicação da expressão e_1 a expressão e_2
- $\texttt{let}\ x\texttt{:}T = e_1\ \texttt{in}\ e_2\ \texttt{end}$ é uma construção que permite declarar identificadores e $\texttt{let rec}\ f\texttt{:}T_1 \rightarrow T_2 = (fn\ y\texttt{:}T_1 \Rightarrow e_1)\ \texttt{in}\ e_2\ \texttt{end}$ permite a declaração de funções recursivas
- note que em L2 o programador deve escrever informação de tipo em programas

Linguagem de Tipos (22)

- os tipos da linguagem L2 são dados pela seguinte gramática:

$$T ::= \mathsf{int} \mid \mathsf{bool} \mid \mathsf{unit} \mid T_1 \rightarrow T_2$$

- $T_1 \rightarrow T_2$ é o tipo de função cujo argumento é do tipo T_1 e cujo resultado é do tipo T_1

Sintaxe de L2 (23)

- Aplicação é associativa a esquerda, logo $e_1\ e_2\ e_3$ é o mesmo que $(e_1\ e_2)\ e_3$

- As setas em tipos função são associativas a direita, logo o tipo $T_1 \to T_2 \to T_3$ é o mesmo que $T_1 \to (T_2 \to T_3)$
- fn se estende o mais a direita possível, logo $fn\ x{:}\mathsf{unit} \Rightarrow x; x$ é o mesmo que $fn\ x{:}\mathsf{unit} \Rightarrow (x;\ x)$

1.2.1 Funções

Informalmente a semântica *call by value* de L2 pode se expressa da seguinte maneira: reduzimos o lado esquerdo de uma aplicação para uma função; reduzimos o lado direito para um valor; computamos a aplicação da função ao seu argumento.

Semântica Formal (24)

$$\langle (fn\ x{:}T \Rightarrow e)\ v,\ \sigma\rangle \longrightarrow \langle \{v/x\}e,\ \sigma\rangle \qquad (\beta)$$

$$\frac{\langle e_2,\ \sigma\rangle \longrightarrow \langle e_2',\ \sigma'\rangle}{\langle v\ e_2,\ \sigma\rangle \longrightarrow \langle v\ e_2',\ \sigma'\rangle} \qquad (\text{APP1})$$

$$\frac{\langle e_1,\ \sigma\rangle \longrightarrow \langle e_1',\ \sigma'\rangle}{\langle e_1\ e_2,\ \sigma\rangle \longrightarrow \langle e_1'\ e_2,\ \sigma'\rangle} \qquad (\text{APP2})$$

Substituição - Exemplos (25)

- A semântica da aplicação de função a argumento envolve substituir variável por valor no corpo da função
- a notação $\{v/x\}e$ representa a expressão que resulta da substituição de todas as **ocorrências livres** de x em e por v.
- Exemplos:

$\{3/x\}(x = x) \equiv (3 = 3)$
$\{3/x\}((fn\ x : \mathsf{int} \Rightarrow x + y)\ x) \equiv (fn\ x : \mathsf{int} \Rightarrow x + y)3$
$\{2/x\}(fn\ y : \mathsf{int} \Rightarrow x + y) \equiv fn\ y : \mathsf{int} \Rightarrow 2 + y$

Segue abaixo a definição da operação de substituição. Note que a condição associada a aplicação da substituição a funções garante que somente variáveis livres vão ser substituidas e que nenhuma variável livre ficará indevidamente ligada após a substituição.

$$
\begin{array}{lcl}
\{e/x\}\ x & = & v \\
\{e/x\}\ y & = & y \ \ (se\ x \neq y) \\
\{e/x\}\ fn\ y : T \Rightarrow e' & = & fn\ y : T \Rightarrow (\{e/x\}e')\ se\ x \neq y\ e\ y \notin\ fv(e) \\
\{e/x\}\ (e_1 e_2) & = & (\{e/x\}e_1)(\{e/x\}e_2) \\
\{e/x\}\ n & = & n \\
\{e/x\}\ (e_1\ \text{op}\ e_2) & = & \{e/x\}e1\ \text{op}\ \{e/x\}e2 \\
\{e/x\}\ (\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3) & = & \texttt{if}\ \{e/x\}e1\ \texttt{then}\ \{e/x\}e2\ \texttt{else}\ \{e/x\}e3 \\
\{e/x\}\ b & = & b \\
\{e/x\}\ \texttt{skip} & = & \texttt{skip} \\
\{e/x\}\ l := e' & = & l := \{e/x\}e' \\
\{e/x\}\ !l & = & !l \\
\{e/x\}\ (e1; e2) & = & \{e/x\}e1; \{e/x\}e2 \\
\{e/x\}(\texttt{while}\ e_1\ \texttt{do}\ e_2) & = & \texttt{while}\ \{e/x\}e_1\ \texttt{do}\ \{e/x\}e_2 \\
\{e/x\}(\texttt{let}\ y{:}T = e_1\ \texttt{in}\ e_2\ \texttt{end}) & = & \texttt{let}\ y{:}T = \{e/x\}e1\ \texttt{in}\ \{e/x\}e_2\ \texttt{end}\ se\ x \neq y\ e\ y \notin\ fv(e)
\end{array}
$$

Na definição acima $fv(e)$ é o conjunto de variáveis livres da expressão e (variáveis que ocorrem mas não são decalaradas em e).

$$
\begin{array}{lcl}
fv(x) & = & \{x\} \\
fv(fn\ y : T \Rightarrow e) & = & fv(e) - \{y\} \\
fv(e_1 e_2) & = & fv(e_1) \cup fv(e_2) \\
fv(n) & = & \{\ \} \\
fv(e1\ \text{op}\ e2) & = & fv(e1) \cup fv(e2) \\
\ldots & \ldots & \ldots
\end{array}
$$

Como exercíocio, termine a definição da de fv iniciada acima.

Tipando Funções (26)

- O ambiente Δ mapeia somente os tipos de endereços; em L2, Γ é um mapeamento de identificadores para seus tipos
- Notação: escrevemos $\Gamma, x : T$ para a função parcial que mapeia identificador x para T, mas para outros identificadores diferentes de x o mapemanto é de acordo com Γ
- Γ é uma representaçao simplificada de uma tabela de símbolos (ver disciplina de Compiladores)

$$\frac{\Gamma(x) = T}{\Gamma; \Delta \vdash x : T} \quad (\textsc{Tvar})$$

$$\frac{\Gamma, x : T; \Delta \vdash e : T'}{\Gamma; \Delta \vdash fn\ x : T \Rightarrow e : T \rightarrow T'} \quad (\textsc{Tfn})$$

$$\frac{\Gamma; \Delta \vdash e_1 : T \rightarrow T' \qquad \Gamma; \Delta \vdash e_2 : T}{\Gamma; \Delta \vdash e_1\ e_2 : T'} \quad (\textsc{Tapp})$$

1.2.2 Declarações

Definições Locais (27)

Para facilitar a leitura, nomeando expressões e restringindo o escopo, é adicionada a seguinte construção:

$$e ::= \cdots \mid \texttt{let}\ x{:}T = e_1\ \texttt{in}\ e_2\ \texttt{end}$$

Pode ser considerada como simples açúcar sintático:

$$\texttt{let}\ x{:}T = e_1\ \texttt{in}\ e_2\ \texttt{end} \quad \equiv \quad (fn\ x{:}T \Rightarrow e_2)\ e_1$$

Exemplo: a expressão abaixo declara o identificador de nome f associado a uma função que soma um ao seu argumento. Esta função é aplicada a 10 na parte `in` da expressão:

$$\texttt{let}\ f{:}\mathsf{int} \to \mathsf{int} = fn\ x{:}\mathsf{int} \Rightarrow x + 1\ \texttt{in} \quad f\ 10\ \texttt{end}$$

Regras de tipagem e regras de redução (28)

$$\frac{\Gamma; \Delta \vdash e_1 : T \qquad \Gamma, x : T; \Delta, \vdash e_2 : T'}{\Gamma; \Delta \vdash \texttt{let}\ x{:}T = e_1\ \texttt{in}\ e_2\ \texttt{end} : T'} \quad (\textsc{Tlet})$$

$$\langle \texttt{let}\ x{:}T = v\ \texttt{in}\ e_2\ \texttt{end},\ \sigma\rangle \longrightarrow \langle \{v/x\}e_2,\ \sigma\rangle \quad (\textsc{let1})$$

$$\frac{\langle e_1,\ \sigma\rangle \longrightarrow \langle e_1',\ \sigma'\rangle}{\langle \texttt{let}\ x{:}T = e_1\ \texttt{in}\ e_2\ \texttt{end},\ \sigma\rangle \longrightarrow \langle \texttt{let}\ x{:}T = e_1'\ \texttt{in}\ e_2\ \texttt{end},\ \sigma'\rangle} \quad (\textsc{let1})$$

Funções recursivas (29)

$$e ::= \ldots \mid \texttt{let rec}\ f{:}T_1 \to T_2 = (fn\ y{:}T_1 \Rightarrow e_1)\ \texttt{in}\ e_2\ \texttt{end}$$

$$\frac{\Gamma, f : T_1 \to T_2, y : T_1;; \Delta \vdash e_1 : T_2 \qquad \Gamma, f : T_1 \to T_2; \Delta, \vdash e_2 : T}{\Gamma; \Delta \vdash \texttt{let rec}\ f{:}T_1 \to T_2 = (fn\ y{:}T_1 \Rightarrow e_1)\ \texttt{in}\ e_2\ \texttt{end} : T} \quad (\textsc{Tletrec})$$

Segue abaixo a definição da função fatorial e a sua chamada para calcular o fatorial de 5 (neste exemplo supomos que operadores para igualdade, multiplicação e para subtração foram adicionados a linguagem):

```
let rec fat : int -> int =
        (fn y:int => if y=0 then 1 else y * fat (y-1))
in fat 5
end
```

Semântica de funções recursivas (30)

$$\langle \texttt{let rec } f{:}T_1 \to T_2 = (fn\ y{:}T_1 \Rightarrow e_1) \texttt{ in } e_2 \texttt{ end},\ \sigma\rangle \longrightarrow \langle \{(fn\ y{:}T_1 \Rightarrow \texttt{let rec } f{:}T_1 \to T_2 = (fn\ y{:}T_1 \Rightarrow e_1) \texttt{ in } e_1 \texttt{ end})/f\}e_2,\ \sigma\rangle \quad \text{(LETREC)}$$

Mais açucar sintático (31)

- $e1; e2$ pode ser codificado como $(fn\ y{:}\mathsf{unit} \Rightarrow e2)\ e1$ onde $y \notin fv(e_2)$
- `while` $e1$ `do` $e2$ poderia ser codificado como:

```
let rec w:unit -> unit =
    fn y:unit => if e1 then (e2; (w skip)) else skip
in
    w skip
end
```

para um novo w e $y \notin fv(e1) \cup fv(e2)$.

1.2.3 Propriedades de L2

Assim como L1, a linguagem L2 é determinística e o enunciado do teorema que afirma essa propriedade é o mesmo. O enunciado das propriedades de preservação e progresso na essência são os mesmos mas neles assumimos que as expressões em consideração são expressões fechadas, ou seja, expressões onde não há variáveis não declaradas.

Progresso e Preservação (32)

Teorema 5 (Progresso) Se e é fechado e $\Gamma;\Delta \vdash e : T$ e $Dom(\Gamma;\Delta) \subseteq Dom(\sigma)$ então e é um valor ou existe $e';\sigma'$ tal que $\langle e,\ \sigma\rangle \longrightarrow \langle e',\ \sigma'\rangle$ e e' é fechado.

Teorema 6 (Preservação) Se e é fechado e $\Gamma;\Delta \vdash e : T$, $Dom(\Gamma;\Delta) \subseteq Dom(\sigma)$ e $\langle e,\ \sigma\rangle \longrightarrow \langle e',\ \sigma'\rangle$ então $\Gamma;\Delta \vdash e' : T$ e $Dom(\Gamma;\Delta) \subseteq dom(\sigma')$.

Todos os resultados sobre os problemas da tipabilidade e da verificação de tipos de L1 permancem os mesmos para L2.