Semântica Operacional Estrutural

Semântica Operacional Estrutural

- O ênfase desta semântica é nos *passos individuais* de execução de um programa.
- A relação de transição tem a forma $\langle S, s\rangle \Rightarrow \gamma$ e representa o *primeiro passo* de execução do programa S no estado s.
- Em $\langle S, s\rangle \Rightarrow \gamma$, se γ é da forma

  $\langle S', s'\rangle$ então a execução ainda não está completa (é uma configuração intermédia).

  s' então a execução terminou e o estado final é s'.
- Uma configuração $\langle S, s\rangle$ diz-se ***bloqueada*** se não existir nenhuma configuração γ tal que $\langle S, s\rangle \Rightarrow \gamma$.

Relação de Transição $\Rightarrow$

$$[\text{ass}_{\text{sos}}] \qquad \langle x := a, s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$[\text{skip}_{\text{sos}}] \qquad \langle \texttt{skip}, s\rangle \Rightarrow s$$

Relação de Transição $\Rightarrow$

$$[\text{comp}^1_{\text{sos}}] \qquad \frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1;S_2, s\rangle \Rightarrow \langle S_1';S_2, s'\rangle}$$

$$[\text{comp}^2_{\text{sos}}] \qquad \frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1;S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

Relação de Transição $\Rightarrow$

$[\text{if}^{\text{tt}}_{\text{sos}}]$ $\quad \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_1,\ s\rangle \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{tt}$

$[\text{if}^{\text{ff}}_{\text{sos}}]$ $\quad \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_2,\ s\rangle \text{ if } \mathcal{B}[\![b]\!]s = \mathbf{ff}$

Relação de Transição $\Rightarrow$

$[\text{while}_{\text{sos}}]$ $\quad \langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \Rightarrow$

$\langle \texttt{if}\ b\ \texttt{then}\ (S;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else skip},\ s\rangle$

Sequência de Derivação

- Uma *sequência de derivação* de um programa S num estado s é uma de duas coisas:
  - uma sequência *finita* de configurações $\gamma_0, \gamma_1, \gamma_2, \cdots, \gamma_k$ tais que $\gamma_0 = \langle S, s\rangle$, $\gamma_i \Rightarrow \gamma_{i+1}$ com $0 \le i < k$, $k \ge 0$, e γ_k é uma configuração terminal ou bloqueada;
  - uma sequência *infinita* $\gamma_0, \gamma_1, \gamma_2, \cdots$ tais que $\gamma_0 = \langle S, s\rangle$ e $\gamma_i \Rightarrow \gamma_{i+1}$ para $0 \le i$
- $\gamma_0 \Rightarrow^i \gamma_i$ indica i passos de execução.
- $\gamma_0 \Rightarrow^* \gamma_i$ indica um número finito de passos de execução.

Sequência de Derivação

Exemplo: Seja s_0 o estado que mapeia todas as variáveis em $\mathbf{0}$, excepto $\mathtt{x}$ e $\mathtt{y}$. Nestes casos: $s_0\ \mathtt{x} = \mathbf{5}$ e $s_0\ \mathtt{y} = \mathbf{7}$.

Para o programa $\quad (\mathtt{z} := \mathtt{x};\ \mathtt{x} := \mathtt{y});\ \mathtt{y} := \mathtt{z}$

temos a sequência de derivação

$$\begin{aligned}
&\langle (\mathtt{z} := \mathtt{x};\ \mathtt{x} := \mathtt{y});\ \mathtt{y} := \mathtt{z},\ s_0\rangle \\
&\quad \Rightarrow \langle \mathtt{x} := \mathtt{y};\ \mathtt{y} := \mathtt{z},\ s_0[\mathtt{z} \mapsto \mathbf{5}]\rangle \\
&\quad \Rightarrow \langle \mathtt{y} := \mathtt{z},\ (s_0[\mathtt{z} \mapsto \mathbf{5}])[\mathtt{x} \mapsto \mathbf{7}]\rangle \\
&\quad \Rightarrow ((s_0[\mathtt{z} \mapsto \mathbf{5}])[\mathtt{x} \mapsto \mathbf{7}])[\mathtt{y} \mapsto \mathbf{5}]
\end{aligned}$$

Note que para *cada passo* da sequência temos uma *árvore de derivação* que justifica esse passo.

Árvore de Derivação

Por exemplo, o primeiro passo

$$\langle (\mathtt{z} := \mathtt{x};\ \mathtt{x} := \mathtt{y});\ \mathtt{y} := \mathtt{z},\ s_0 \rangle \Rightarrow \langle \mathtt{x} := \mathtt{y};\ \mathtt{y} := \mathtt{z},\ s_0[\mathtt{z} \mapsto \mathbf{5}] \rangle$$

a árvore de derivação é

$$\dfrac{\dfrac{\langle \mathtt{z} := \mathtt{x},\ s_0 \rangle \Rightarrow s_0[\mathtt{z} \mapsto \mathbf{5}]}{\langle \mathtt{z} := \mathtt{x};\ \mathtt{x} := \mathtt{y},\ s_0 \rangle \Rightarrow \langle \mathtt{x} := \mathtt{y},\ s_0[\mathtt{z} \mapsto \mathbf{5}] \rangle}}{\langle (\mathtt{z} := \mathtt{x};\ \mathtt{x} := \mathtt{y});\ \mathtt{y} := \mathtt{z},\ s_0 \rangle \Rightarrow \langle \mathtt{x} := \mathtt{y};\ \mathtt{y} := \mathtt{z},\ s_0[\mathtt{z} \mapsto \mathbf{5}] \rangle}$$

Construa as árvores de derivação para os restantes passos da sequência.

Sequência de Derivação

Exercício: Considere o seguinte programa

$$\mathtt{z}{:=}\mathbf{0};\ \mathtt{while}\ \mathtt{y} \leq \mathtt{x}\ \mathtt{do}\ (\mathtt{z}{:=}\mathtt{z}{+}\mathbf{1};\ \mathtt{x}{:=}\mathtt{x}{-}\mathtt{y})$$

Determine um estado para o qual a sequência de derivação deste programa é finita e outro para o qual é infinita.

- A execução de um programa S num estado s
  - *termina* se e só se existir uma sequência de derivação finita começada em $\langle S,\ s \rangle$
  - *diverge* (ou *entra em ciclo*) se e só se existir uma sequência de derivação infinita começada em $\langle S,\ s \rangle$
  - *termina com sucesso* se e só se $\langle S,\ s \rangle \Rightarrow^* s'$

Nota: na linguagem **While** não temos configurações que bloqueiem, mas veremos extensões que sim.

Indução no comprimento das sequências

Provar uma propriedade para todas as sequências de derivação.

Caso de base

- Provar que a propriedade se verifica para sequências de derivação de comprimento 0.

Passo indutivo

- Assumir que a propriedade se verifica para todas as sequências de derivação de comprimento até k (é a *hipótese de indução*) e provar que a propriedade também se verifica para as sequências derivação de comprimento k+1.

Propriedades

Lema: Se $\langle S_1,\ s \rangle \Rightarrow^{\mathrm{k}} s'$ então $\langle S_1;S_2,\ s \rangle \Rightarrow^{\mathrm{k}} \langle S_2,\ s' \rangle$

Prova: Por indução em k.

Lema: Se $\langle S_1;S_2,\ s \rangle \Rightarrow^{\mathrm{k}} s''$ então existe um estado s' e números naturais k_1 e k_2 tais que $\langle S_1,\ s \rangle \Rightarrow^{\mathrm{k}_1} s'$, $\langle S_2,\ s' \rangle \Rightarrow^{\mathrm{k}_2} s''$ e $\mathrm{k} = \mathrm{k}_1 + \mathrm{k}_2$.

Prova: Por indução no comprimento da sequência $\langle S_1;S_2,\ s \rangle \Rightarrow^{\mathrm{k}} s''$.

Determinismo

A semântica operacional estrutural aqui apresentada é *determinista*.

Teorema: Para quaisquer S, s, γ e γ',

se $\langle S, s\rangle \Rightarrow \gamma$ e $\langle S, s\rangle \Rightarrow \gamma'$ então $\gamma = \gamma'$.

Prova: Por indução na estrutura da derivação.

Equivalência Semântica

Definição: Dois programas S_1 e S_2 dizem-se *semanticamente equivalentes* se, para todo o estado **s,**

- $\langle S_1, s\rangle \Rightarrow^* \gamma$ sse $\langle S_2, s\rangle \Rightarrow^* \gamma$, caso γ seja uma configuração terminal ou bloqueada
- houver uma sequência de derivação começada em $\langle S_1, s\rangle$ divergente sse houver uma começada em $\langle S_2, s\rangle$. também divergente.

Exercício: Mostre que os programas seguintes são semanticamente equivalentes:

- S;`skip` and S
- `while` b `do` S and `if` b `then` (S; `while` b `do` S) `else skip`
- S_1;(S_2;S_3) and (S_1;S_2);S_3

A função semântica $\mathcal{S}_{\text{sos}}$

O significado de um programa pode ser visto como uma função parcial de $\mathbf{State}$ para $\mathbf{State}$.

Definição:

$$\mathcal{S}_{\text{sos}}\colon \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

$$\mathcal{S}_{\text{sos}}[\![S]\!]s = \begin{cases} s' & \text{if } \langle S, s\rangle \Rightarrow^* s' \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

A boa definição de $\mathcal{S}_{\text{sos}}$ é uma consequência do determinismo da relação de transição.

Semântica de transições para expressões

Podemos adoptar uma semântica operacional *small-step* para dar semântica às expressões aritméticas e booleanas. Para isso temos que considerar

$$\mathbf{State} = \mathbf{Var} \to \mathbf{Num}$$

Exercício: Defina uma semântica de transições para expressões aritméticas.

Exercício: Defina uma semântica de transições para expressões booleanas.

Com poderia fazer uma avaliação *"curto-circuito"* das expressões booleanas (ao estilo do C) ?

Tratamento de erros

Considere que queremos estender a linguagem de expressões aritméticas com a operação de divisão.

Como avaliar a divisão por zero ?

Temos diferentes opções:

- Associar à divisão por zero um valor no domínio de interpretação (considerando assim a operação válida).
- Usando semântica operacional, definir o sistema de inferência das transições de forma a que não seja possível derivar transições de expressões que envolvam divisões por zero. Teremos assim algumas configurações bloqueadas.
- Introduzir um valor especial de *erro*, $\perp$, e estender os domínios de interpretação das expressões com este valor.

$$\mathbf{Z}_\perp = \mathbf{Z} \cup \{\perp\} \qquad \mathbf{T}_\perp = \mathbf{T} \cup \{\perp\}$$

Tratamento de erros

Exercício:

- Defina uma semântica operacional para expressões aritméticas que assinale erro no caso de ocorrências de divisões por zero.
- Faça o mesmo para avaliação de expressões booleanas.
- Experimente as várias opções sugeridas.

Execução de programas com ocorrência de erros

Usando semântica operacional, podemos lidar com esta situação de duas formas:

- Definir o sistema de inferência das transições de forma a que comandos avaliados em estados que deem origem a expressões erróneas venham a dar origem a configurações bloqueadas.
- Considerar a existência de um *estado de erro*, $\perp$, e considerar como estados finais possíveis os elementos do conjunto

$$\mathbf{State}_\perp = \mathbf{State} \cup \{\perp\}$$

  Os estados onde os comandos podem ser executados são os do conjunto $\mathbf{State}$.

Execução de programas com ocorrência de erros

Exercício:

- Escreva as regras de uma versão da semântica de transições small-step dos comandos que lide com situações de erro.
- Experimente as várias opções sugeridas.