Semântica de compilação

- Compiladores traduzem o programa fonte para uma linguagem intermediária (IR) de uma máquina abstrata
- IRs de alto nível preservam muitas características da linguagem de programação. IRs de mais baixo nível se aproximam de código de máquinas reais.
- Programas em linguagens intermediárias podem também ser interpretados
- Programas em Java, por exemplo, são compilados para a IR conhecida como bytecode Java, que é então interpretado pela JVM (pode também ser compilado)

Máquina abstrata SSM0

Vamos descrever uma **máquina abstrata** denominada SSM0 (*Simple Stack Machine*).

Definição: O estado da máquina SSM0 (configuração) é um par

$$(\text{code}, \text{stack})$$

onde

- code é uma **lista de instruções SSM0**
- stack é uma **lista de números inteiros**

Instruções SSM0

Abaixo segue uma descrição informal das instruções da máquina SSM0:

- `PUSH z` : empilha inteiro `z`
- `POP` : remove topo da pilha
- `COPY` : empilha uma nova cópia do número no topo da pilha
- `INC` : soma um ao número no topo da pilha
- `DEC` : subtrai um do número no topo da pilha (se for
- `JUMP n` : pula `n` instruções da lista de código
- `JMPIFZERO n` : testa o topo da pilha, removendo-o. Se o topo da pilha for `0`, pula `n` instruções da lista de código. Se não for `0`, não faz nada.

Notação para listas

Notação: para listas

- lista vazia : $[]$
- prefixação: $1 :: []$
- notação simplificada: $[1;2;3] = 1 :: 2 :: 3 :: [], \quad 5 :: [] = [5]$
- concatenação: $[1;2]{+}{+}[3;4] = [1;2;3;4]$
- tamanho: $\texttt{length}([3;4]) = 2$

Nota: tanto code quanto stack serão operados como *pilhas*, onde o elemento mais à esquerda representa o topo.

Exemplo: a pilha $[3;4;2;1;5]$ possui 3 no topo.

Máquina abstrata SSM0: sintaxe

$$
\begin{array}{lcl}
\texttt{n} & \in & \mathbb{N} \\
\texttt{z} & \in & \mathbb{Z} \\
 & & \\
\texttt{i} & \in & \texttt{Inst} \\
\texttt{i} & ::= & \texttt{PUSH z} \mid \texttt{POP} \mid \texttt{COPY} \mid \texttt{INC} \mid \\
 & & \texttt{DEC} \mid \texttt{JUMP n} \mid \texttt{JMPIFZERO n} \\
 & & \\
\texttt{code} & \in & \texttt{SSM0-IR} \\
\texttt{code} & ::= & [] \mid \texttt{i} :: \texttt{code} \\
 & & \\
\texttt{stack} & \in & \texttt{Stack} \\
\texttt{stack} & ::= & [] \mid \texttt{z} :: \texttt{stack}
\end{array}
$$

Máquina abstrata SSM0: semântica operacional

Definição: $\triangleright \subseteq (\text{SSM0–IR}, \text{Stack}) \times (\text{SSM0–IR}, \text{Stack})$ é a menor relação tal que

$$\frac{}{(\texttt{PUSH}\ z :: code,\ stack) \triangleright (code,\ z :: stack)}$$

$$\frac{}{(\texttt{POP} :: code,\ z :: stack) \triangleright (code,\ stack)}$$

$$\frac{}{(\texttt{COPY} :: code,\ z :: stack) \triangleright (code,\ z :: z :: stack)}$$

$$\frac{}{(\texttt{INC} :: code,\ z :: stack) \triangleright (code,\ (z+1) :: stack)}$$

$$\vdots$$

Máquina abstrata SSM0: semântica operacional (cont.)

$$\vdots$$

$$\frac{}{(\texttt{DEC} :: \text{code},\ z :: \text{stack}) \rhd (\text{code},\ (z-1) :: \text{stack})}$$

$$\frac{}{(\texttt{JUMP}\ n :: i_1 :: \ldots :: i_n :: \text{code},\ \text{stack}) \rhd (\text{code},\ \text{stack})}$$

$$\frac{}{(\texttt{JMPIFZERO}\ n :: i_1 :: \ldots :: i_n :: \text{code},\ 0 :: \text{stack}) \rhd (\text{code},\ \text{stack})}$$

$$\frac{z \neq 0}{(\texttt{JMPIFZERO}\ n :: \text{code},\ z :: \text{stack}) \rhd (\text{code},\ \text{stack})}$$

Definição: $\rhd^*$ é o fecho transitivo e reflexivo de $\rhd$.

Máquina abstrata SSM0: comportamentos possíveis

Iniciando em configuração $(\text{code}, \text{stack})$, os seguintes comportamentos são possíveis

- Avaliação termina sem erro: $(\text{code}, \text{stack}) \triangleright^* ([], \text{stack}')$
- Avaliação termina em erro: $(\text{code}, \text{stack}) \triangleright^* (\text{code}', \text{stack}') \not\triangleright$ e $\text{code}' \neq []$

Observar que divergência não é possível na máquina SSM0.

Situações de erro

Com a exceção de `JMP`, todas instruções operam com a pilha. `PUSH` pode operar, sem erro, com pilha vazia.

Erro também ocorre se `JMP n` é executada com `n` maior do que o número de instruções que seguem

O mesmo acontece com a execução de `JUMPIFZERO n` (caso o topo da pilha contém `0`).

Situações de erro:

- $(\texttt{i} :: \texttt{code}, []) \not\triangleright$, se $\texttt{i} \in \{$ `POP`, `COPY`, `INC`, `DEC`, `JMPIFZERO n` $\}$
- $(\texttt{JMP n} :: \texttt{code}, \texttt{stack}) \not\triangleright$, se $\texttt{n} > \texttt{length(code)}$
- $(\texttt{JMPIFZERO n} :: \texttt{code}, 0 :: \texttt{stack}) \not\triangleright$, se $\texttt{n} > \texttt{length(code)}$

OBS.: Instrução `JMP 0` tem o mesmo efeito de uma instrução `NOP` (*no operation*). Instrução `JMPIFZERO 0` tem o mesmo efeito de uma instrução `POP`.

Compilação de L0 para SSM0

- Observar que todas as instruções, com exceção de `JMP`, operam com o topo da pila.
- A compilação $\mathcal{C}(\texttt{succ}\ \texttt{e}_1)$ por exemplo, gera a seguinte lista de instruções

$$\mathcal{C}(\texttt{e}_1)\ \texttt{++}\ \texttt{[INC]}$$

- As intruções geradas por $\mathcal{C}(\texttt{e}_1)$, quando executadas, deixam um valor no topo da pilha.
- Esse valor será incrementado pela execução da instrução `INC`.

Compilação de L0 para SSM0-IR

$\mathcal{C} : \text{L0} \rightarrow \text{SSM0–IR}$

$$
\begin{array}{lcl}
\mathcal{C}(\texttt{true}) & = & \texttt{[PUSH 1]} \\
\mathcal{C}(\texttt{false}) & = & \texttt{[PUSH 0]} \\
\mathcal{C}(\texttt{0}) & = & \texttt{[PUSH 0]} \\
 & & \\
\mathcal{C}(\texttt{succ } e_1) & = & \mathcal{C}(e_1) \texttt{ ++ [INC]} \\
 & & \\
\mathcal{C}(\texttt{pred } e_1) & = & \mathcal{C}(e_1) \texttt{ ++ [COPY; JUMPIFZERO 1; DEC]} \\
\mathcal{C}(\texttt{iszero } e_1) & = & \mathcal{C}(e_1) \texttt{ ++ [JMPIFZERO 2; PUSH 0; JUMP 1; PUSH 1]} \\
 & & \\
\mathcal{C}(\texttt{if } e_1 & = & \textit{let} \\
\quad \texttt{then } e_2 & & \quad n_2 = \texttt{length}(\mathcal{C}(e_2)) \\
\quad \texttt{else } e_3) & & \quad n_3 = \texttt{length}(\mathcal{C}(e_3)) \\
 & & \textit{in} \\
 & & \quad \mathcal{C}(e_1) \texttt{ ++ [JMPIFZERO (}n_2\texttt{+1)] ++} \\
 & & \quad \mathcal{C}(e_2) \texttt{ ++ [JUMP } n_3\texttt{] ++ } \mathcal{C}(e_3)
\end{array}
$$

Preservação da semântica

Execução de programa SSM0 que **resulta da função de compilação**, se inicia com pilha vazia, termina com pilha contendo apenas um valor.

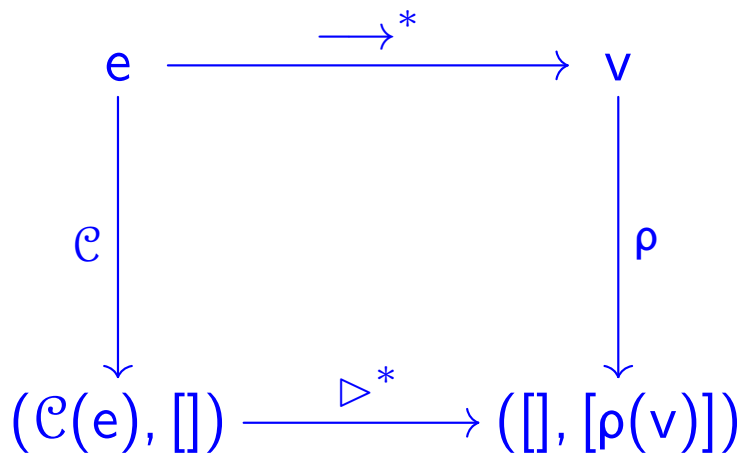
Teorema: Preservação de comportamentos que levam a valor.

Se $\mathsf{e} \longrightarrow^* \mathsf{v}$ então

$$(\mathcal{C}(\mathsf{e}), []) \rhd^* ([], [\rho(\mathsf{v})])$$

onde

$$\begin{array}{lcl} \rho(\texttt{true}) & = & 1 \\ \rho(\texttt{false}) & = & 0 \\ \rho(\texttt{0}) & = & 0 \\ \rho(\texttt{succ nv}) & = & 1 + \rho(\mathsf{nv}) \end{array}$$



Preservação de avaliações: demonstração

Teorema: Se $e \longrightarrow^* v$ então $(\mathcal{C}(e), []) \rhd^* ([], [\rho(v)])$

Demonstração: da hipótese $e \longrightarrow^* v$ obtém-se $e \Downarrow v$.

Precisamos fortalecer a hipótese indutiva, considerando a propriedade abaixo:

$$\forall c \in \texttt{SSM0-IR}, \forall s \in \mathsf{Stack}, (\mathcal{C}(e)\texttt{++}c, s) \rhd^* (c, \rho(v) :: s)$$

Prova-se a propriedade acima por indução em $e \Downarrow v$.

O teorema desejado é um corolário da propriedade acima. □