Uma Máquina Abstracta

Máquinas Abstractas

Máquinas abstractas

- são *máquinas* porque permitem a execução passo a passo dos programas;
- são *abstractas* porque omitem os detalhes das máquinas reais.

- Fornecem uma *linguagem intermédia* para a compilação, fazendo a ponte entre as linguagens de programação e as linguagem máquina reais.
- As instruções de uma máquina abstracta devem ser talhadas de acordo com as características da linguagem de programação.
- É comum lidarem com conceitos como: *state, store, stack, registers*.
- *Emulador, interpretador,* e *máquina virtual* são termos alternativos para este conceito.
- Tornam a implementação das linguagens de programação mais portáveis e mais fáceis de manter.

Uma implementação correcta

- A especificação formal da semântica de uma linguagem de programação permite-nos argumentar acerca da correcção da sua implementação.
- Vamos ilustrar isso definindo uma tradução da linguagem **While** para o assembly de uma máquina abstracta e provar que a tradução é correcta. Como?
  - Damos significado às instruções da máquina abstracta através de uma semântica operacional.
  - Definimos uma função de tradução que mapeia expressões e comandos da linguagem **While** em sequências de instruções da máquina abstracta.
  - Demonstramos que o significado do programa traduzido e do programa original coincidem.

A máquina abstracta (**AM**)

A máquina abstracta **AM** tem configurações da forma $\langle c,\ e,\ s \rangle$ sendo

- c a sequência de *instruções* (ou *código*) a ser executada
- e a *stack* de avaliação das expressões aritméticas e booleanas (formalmente é uma lista de valores)
- s o *storage (*ou *state)* (o estado das variáveis)

A maquina abstracta **AM**

As *instruções* de **AM** são dadas pela sintaxe abstracta:

$$\begin{array}{lll} inst & ::= & \text{PUSH-}n \mid \text{ADD} \mid \text{MULT} \mid \text{SUB} \\ & \mid & \text{TRUE} \mid \text{FALSE} \mid \text{EQ} \mid \text{LE} \mid \text{AND} \mid \text{NEG} \\ & \mid & \text{FETCH-}x \mid \text{STORE-}x \\ & \mid & \text{NOOP} \mid \text{BRANCH}(c,\ c) \mid \text{LOOP}(c,\ c) \\ c & ::= & \varepsilon \mid inst{:}c \end{array}$$

$\mathbf{Code}$ é a categoria sintática das sequências de instruções.

$\mathbf{Stack} = (\mathbf{Z} \cup \mathbf{T})^\star$

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

Configurações de **AM**

$\langle c,\ e,\ s\rangle \in \mathbf{Code} \times \mathbf{Stack} \times \mathbf{State}$

$\langle \varepsilon,\ e,\ s\rangle$ é uma configuração terminal (ou final).

A semântica das instruções de **AM** é dada pela relação de transição entre configurações

$$\langle c,\ e,\ s\rangle \rhd \langle c',\ e',\ s'\rangle$$

Cada $\langle c,\ e,\ s\rangle \rhd \langle c',\ e',\ s'\rangle$ representa *um passo de execução* da máquina.

Semântica operacional de **AM**

$$\begin{array}{lll} \langle \text{PUSH-}n{:}c,\ e,\ s\rangle & \rhd & \langle c,\ \mathcal{N}[\![n]\!]{:}e,\ s\rangle \\ \langle \text{ADD}{:}c,\ z_1{:}z_2{:}e,\ s\rangle & \rhd & \langle c,\ (z_1{+}z_2){:}e,\ s\rangle \quad \text{if } z_1,\ z_2{\in}\mathbf{Z} \\ \langle \text{MULT}{:}c,\ z_1{:}z_2{:}e,\ s\rangle & \rhd & \langle c,\ (z_1{\star}z_2){:}e,\ s\rangle \quad \text{if } z_1,\ z_2{\in}\mathbf{Z} \\ \langle \text{SUB}{:}c,\ z_1{:}z_2{:}e,\ s\rangle & \rhd & \langle c,\ (z_1{-}z_2){:}e,\ s\rangle \quad \text{if } z_1,\ z_2{\in}\mathbf{Z} \\ \langle \text{TRUE}{:}c,\ e,\ s\rangle & \rhd & \langle c,\ \mathbf{tt}{:}e,\ s\rangle \\ \langle \text{FALSE}{:}c,\ e,\ s\rangle & \rhd & \langle c,\ \mathbf{ff}{:}e,\ s\rangle \\ \langle \text{EQ}{:}c,\ z_1{:}z_2{:}e,\ s\rangle & \rhd & \langle c,\ (z_1{=}z_2){:}e,\ s\rangle \quad \text{if } z_1,\ z_2{\in}\mathbf{Z} \\ \langle \text{LE}{:}c,\ z_1{:}z_2{:}e,\ s\rangle & \rhd & \langle c,\ (z_1{\leq}z_2){:}e,\ s\rangle \quad \text{if } z_1,\ z_2{\in}\mathbf{Z} \end{array}$$

Semântica operacional de **AM**

$$\langle \text{AND}{:}c,\ t_1{:}t_2{:}e,\ s\rangle \ \rhd \begin{cases} \langle c, \mathbf{tt} : e, s\rangle & \text{if } t_1{=}\mathbf{tt} \text{ and } t_2{=}\mathbf{tt} \\ \langle c, \mathbf{ff} : e, s\rangle & \text{if } t_1{=}\mathbf{ff} \text{ or } t_2{=}\mathbf{ff},\ t_1,\ t_2{\in}\mathbf{T} \end{cases}$$

$$\langle \text{NEG}{:}c,\ t{:}e,\ s\rangle \ \rhd \begin{cases} \langle c, \mathbf{ff} : e, s\rangle & \text{if } t{=}\mathbf{tt} \\ \langle c, \mathbf{tt} : e, s\rangle & \text{if } t{=}\mathbf{ff} \end{cases}$$

Semântica operacional de **AM**

$\langle \text{FETCH-}x{:}c,\ e,\ s\rangle \ \rhd\ \langle c,\ (s\ x){:}e,\ s\rangle$

$\langle \text{STORE-}x{:}c,\ z{:}e,\ s\rangle \ \rhd\ \langle c,\ e,\ s[x{\mapsto}z]\rangle \quad \text{if } z{\in}\mathbf{Z}$

$\langle \text{NOOP}{:}c,\ e,\ s\rangle \ \rhd\ \langle c,\ e,\ s\rangle$

$$\langle \text{BRANCH}(c_1,\ c_2){:}c,\ t{:}e,\ s\rangle \ \rhd\ \begin{cases} \langle c_1 : c, e, s\rangle & \text{if } t{=}\mathbf{tt} \\ \langle c_2 : c, e, s\rangle & \text{if } t{=}\mathbf{ff} \end{cases}$$

$\langle \text{LOOP}(c_1,\ c_2){:}c,\ e,\ s\rangle \ \rhd$

$$\langle c_1{:}\text{BRANCH}(c_2{:}\text{LOOP}(c_1,\ c_2),\ \text{NOOP}){:}c,\ e,\ s\rangle$$

Sequência de computação

- Uma *sequência de computação* de uma sequência de instruções **c** num estado **s** é uma de duas coisas:
  - uma sequência *finita* de configurações $\gamma_0, \gamma_1, \gamma_2, \cdots, \gamma_k$ tais que $\gamma_0 = \langle c,\ \varepsilon,\ s\rangle$ com $\gamma_i \rhd \gamma_{i+1}$ e $0{\le}i{<}k, k{\ge}0$, e não há nenhum γ tal que $\gamma_k \rhd \gamma$
  - uma sequência *infinita* $\gamma_0, \gamma_1, \gamma_2, \cdots$ tais que $\gamma_0 = \langle c,\ \varepsilon,\ s\rangle$ e $\gamma_i \rhd \gamma_{i+1}$ para $0{\le}i$
- As configurações *iniciais* têm sempre a stack vazia.
- Uma sequência de computação finita pode terminar numa configuração *final* ou *bloqueada*.

Sequência de computação

Exemplo: Assumindo que $s\ \mathbf{x} = \mathbf{3}$

$$\begin{aligned} &\langle \text{PUSH-1:FETCH-x:ADD:STORE-x},\ \varepsilon,\ s\rangle \\ &\rhd \langle \text{FETCH-x:ADD:STORE-x},\ \mathbf{1},\ s\rangle \\ &\rhd \langle \text{ADD:STORE-x},\ \mathbf{3}{:}\mathbf{1},\ s\rangle \\ &\rhd \langle \text{STORE-x},\ \mathbf{4},\ s\rangle \\ &\rhd \langle \varepsilon,\ \varepsilon,\ s[\mathrm{x}{\mapsto}\mathbf{4}]\rangle \end{aligned}$$

Exercício: $\langle \text{ADD},\ \varepsilon,\ s\rangle$ é uma configuração bloqueada.

Dê exemplos de outras configurações bloqueadas e de sequências de computação finitas que terminam em configurações bloqueadas.

Sequência de computação

Exemplo: $\langle \text{LOOP(TRUE, NOOP)},\ \varepsilon,\ s\rangle$

$$\begin{aligned} &\rhd \langle \text{TRUE:BRANCH(NOOP:LOOP(TRUE, NOOP), NOOP)},\ \varepsilon,\ s\rangle \\ &\rhd \langle \text{BRANCH(NOOP:LOOP(TRUE, NOOP), NOOP)},\ \mathbf{tt},\ s\rangle \\ &\rhd \langle \text{NOOP:LOOP(TRUE, NOOP)},\ \varepsilon,\ s\rangle \\ &\rhd \langle \text{LOOP(TRUE, NOOP)},\ \varepsilon,\ s\rangle \\ &\rhd \cdots \end{aligned}$$

Exercício: Indique que função é implementada pelo seguinte código

```
PUSH-0:STORE-z:FETCH-x:STORE-r:
LOOP(FETCH-r:FETCH-y:LE,
     FETCH-y:FETCH-r:SUB:STORE-r:
     PUSH-1:FETCH-z:ADD:STORE-z)
```

Propriedades de **AM**

Lema:
Se $\langle c_1, e_1, s\rangle \rhd^{\mathrm{k}} \langle c', e', s'\rangle$ então $\langle c_1{:}c_2, e_1{:}e_2, s\rangle \rhd^{\mathrm{k}} \langle c'{:}c_2, e'{:}e_2, s'\rangle$

Lema: Se $\langle c_1{:}c_2, e, s\rangle \rhd^{\mathrm{k}} \langle \varepsilon, e'', s''\rangle$
então $\langle c_1, e, s\rangle \rhd^{\mathrm{k}_1} \langle \varepsilon, e', s'\rangle$ e $\langle c_2, e', s'\rangle \rhd^{\mathrm{k}_2} \langle \varepsilon, e'', s''\rangle$
para alguma configuração $\langle \varepsilon, e', s'\rangle$ e números naturais k_1 e k_2
tais que $\mathrm{k} = \mathrm{k}_1 + \mathrm{k}_2$

Propriedades de **AM**

A semântica da máquina abstracta aqui apresentada é *determinista*.

Teorema: Para quaisquer γ, γ' e γ'',

se $\gamma \rhd \gamma''$ e $\gamma \rhd \gamma'$ então $\gamma' = \gamma''$.

A função de execução $\mathcal{M}$

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

Definição: $\mathcal{M}$: $\mathbf{Code} \to (\mathbf{State} \hookrightarrow \mathbf{State})$

$$\mathcal{M}[\![c]\!]\ s = \begin{cases} s' & \text{if } \langle c, \varepsilon, s\rangle \rhd^* \langle \varepsilon, e, s'\rangle \\ \underline{\text{undef}} & \text{otherwise} \end{cases}$$

A boa definição de $\mathcal{M}$ é uma consequência do determinismo da relação de transição.

Especificação da tradução

*Como gerar código **AM** para um dado programa?*

- A *tradução* de um programa **While** gera código **AM**.
- Precisamos de traduzir expressões e comandos.
- Cada tradução é especificada por uma função total.

Tradução de expressões aritméticas

Definição: $\mathcal{CA}\colon\ \mathbf{Aexp} \to \mathbf{Code}$

$$
\begin{aligned}
\mathcal{CA}[\![n]\!] &= \text{PUSH-}n\\
\mathcal{CA}[\![x]\!] &= \text{FETCH-}x\\
\mathcal{CA}[\![a_1{+}a_2]\!] &= \mathcal{CA}[\![a_2]\!]{:}\mathcal{CA}[\![a_1]\!]{:}\text{ADD}\\
\mathcal{CA}[\![a_1 \star a_2]\!] &= \mathcal{CA}[\![a_2]\!]{:}\mathcal{CA}[\![a_1]\!]{:}\text{MULT}\\
\mathcal{CA}[\![a_1{-}a_2]\!] &= \mathcal{CA}[\![a_2]\!]{:}\mathcal{CA}[\![a_1]\!]{:}\text{SUB}
\end{aligned}
$$

Tradução de expressões booleanas

Definição: $\mathcal{CB}\colon\ \mathbf{Bexp} \to \mathbf{Code}$

$$
\begin{aligned}
\mathcal{CB}[\![\texttt{true}]\!] &= \text{TRUE}\\
\mathcal{CB}[\![\texttt{false}]\!] &= \text{FALSE}\\
\mathcal{CB}[\![a_1 = a_2]\!] &= \mathcal{CA}[\![a_2]\!]{:}\mathcal{CA}[\![a_1]\!]{:}\text{EQ}\\
\mathcal{CB}[\![a_1{\leq}a_2]\!] &= \mathcal{CA}[\![a_2]\!]{:}\mathcal{CA}[\![a_1]\!]{:}\text{LE}\\
\mathcal{CB}[\![\neg b]\!] &= \mathcal{CB}[\![b]\!]{:}\text{NEG}\\
\mathcal{CB}[\![b_1{\wedge}b_2]\!] &= \mathcal{CB}[\![b_2]\!]{:}\mathcal{CB}[\![b_1]\!]{:}\text{AND}
\end{aligned}
$$

Tradução de comandos

Definição: $\mathcal{CS}\colon\ \mathbf{Stm} \to \mathbf{Code}$

$$
\begin{aligned}
\mathcal{CS}[\![x := a]\!] &= \mathcal{CA}[\![a]\!]{:}\text{STORE-}x\\
\mathcal{CS}[\![\texttt{skip}]\!] &= \text{NOOP}\\
\mathcal{CS}[\![S_1;S_2]\!] &= \mathcal{CS}[\![S_1]\!]{:}\mathcal{CS}[\![S_2]\!]\\
\mathcal{CS}[\![\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2]\!] &= \mathcal{CB}[\![b]\!]{:}\text{BRANCH}(\mathcal{CS}[\![S_1]\!],\mathcal{CS}[\![S_2]\!])\\
\mathcal{CS}[\![\texttt{while}\ b\ \texttt{do}\ S]\!] &= \text{LOOP}(\mathcal{CB}[\![b]\!],\mathcal{CS}[\![S]\!])
\end{aligned}
$$

A função de semântica $\mathcal{S}_{\mathrm{am}}$

O significado de um programa **While** pode ser obtido, primeiro traduzindo-o para código **AM**, e depois executando o código gerado na máquina abstracta.

Definição: $\mathcal{S}_{\mathrm{am}}\colon\ \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$

$$\mathcal{S}_{\mathrm{am}}[\![S]\!] = (\mathcal{M} \circ \mathcal{CS})[\![S]\!]$$

Correcção da tradução

Lema: Para toda a expressão aritmética **a**, verifica-se que

$$\langle \mathcal{CA}[\![a]\!],\, \varepsilon,\, s\rangle \rhd^* \langle \varepsilon,\, \mathcal{A}[\![a]\!]s,\, s\rangle$$

Além disso, todas as configurações intermédias desta sequência de computação têm uma stack não vazia.

Prova: Por indução estrutural em **a**.

Lema: Para toda a expressão boleada b, verifica-se que

$$\langle \mathcal{CB}[\![b]\!],\, \varepsilon,\, s\rangle \rhd^* \langle \varepsilon,\, \mathcal{B}[\![b]\!]s,\, s\rangle$$

Além disso, todas as configurações intermédias desta sequência de computação têm uma stack não vazia.

Prova: Por indução estrutural em b.

Correcção da tradução

- Para formular a correcção da tradução de comandos podemos usar semântica natural ou semântica operacional estrutural.
- Vamos aqui ilustrar a prova seguindo a abordagem da semântica natural.
- Queremos provar que para qualquer programa S de **While**,

$$\mathcal{S}_{\mathrm{ns}}[\![S]\!] = \mathcal{S}_{\mathrm{am}}[\![S]\!]$$

Correcção da tradução

Lema: Para qualquer programa S e estados s e s',

se $\langle S,\, s\rangle \to s'$ então $\langle \mathcal{CS}[\![S]\!],\, \varepsilon,\, s\rangle \rhd^* \langle \varepsilon,\, \varepsilon,\, s'\rangle$

Prova: Por indução na estrutura da derivação de $\langle S,\, s\rangle \to s'$.

Lema: Para qualquer programa S, estados s e s',

se $\langle \mathcal{CS}[\![S]\!],\, \varepsilon,\, s\rangle \rhd^{\mathrm{k}} \langle \varepsilon,\, e,\, s'\rangle$ então $\langle S,\, s\rangle \to s'$ e $e = \varepsilon$

Prova: Por indução no comprimento k da sequência de computação.

Correcção da implementação

Teorema: Para qualquer programa S da linguagem **While**,

$$\mathcal{S}_{\mathrm{ns}}[\![S]\!] = \mathcal{S}_{\mathrm{am}}[\![S]\!]$$

Prova: Consequência directa dos lemas anteriores.

É evidente que a implementação também é correcta em relação à semântica operacional estrutural, i.e., para qualquer programa S da linguagem **While**,

$$\mathcal{S}_{\mathrm{sos}}[\![S]\!] = \mathcal{S}_{\mathrm{am}}[\![S]\!]$$

Porquê?