Semânticas de Linguagens de Programação
Exercícios 5
Máquina abstracta AM para a linguagem While

1. Determina o que calcula o código p seguinte executando $\langle p, \varepsilon, s\rangle$, com $s(x) = 10$ e $s(y) = 6$.

   $$\begin{aligned}&\textsc{push}(0) : \textsc{store}(z) : \textsc{fetch}(x) : \textsc{store}(r) :\\ &\textsc{loop}(\textsc{fetch}(r) : \textsc{fetch}(y) : \textsc{le}, \textsc{fetch}(y) : \textsc{fetch}(r) : \textsc{sub} : \textsc{store}(r) :\\ &\textsc{push}(1) : \textsc{fetch}(z) : \textsc{add} : \textsc{store}(z))\end{aligned}$$

2. Verifica que com $s(x) = 3$,

   $$\begin{aligned}& \mathcal{S}_{am}[\![y := 1; \texttt{while } \neg(x = 1) \texttt{ do } (y := y \times x; x := x - 1)]\!]s\\ = \;& \mathcal{M}[\![\textsc{push}(1) : \textsc{store}(y) : \textsc{loop}(\textsc{push}(1) : \textsc{fetch}(x) : \textsc{eq} : \textsc{neg},\\ & \textsc{fetch}(x) : \textsc{fetch}(y) : \textsc{mult} : \textsc{store}(y) : \textsc{push}(1) :\\ & \textsc{fetch}(x) : \textsc{sub} : \textsc{store}(x))]\!]s\\ = \;& s[x \mapsto 1][y \mapsto 6]\end{aligned}$$

   Em geral o que calcula?

3. Gera código **AM** para os seguintes programas:
   (a) $z := x; y := 0; \texttt{while } \neg z = 0 \texttt{ do } (y := y + 1; z := z - 1)$
   (b) $z := 0; \texttt{while } y \leq x \texttt{ do } (z := z + 1; x := x - y)$

   Para cada um deles verifica que a semântica do código coincide com a semântica operacional natural dos programas, considerando o estado inicial s onde $s(x) = 2$ e $s(y) = 1$.

4. Prova que se $\langle p_1, e_1, s_1\rangle \triangleright^k \langle p', e', s'\rangle$ então $\langle p_1 : p_2, e_1 : e_2, s_1\rangle \triangleright^k \langle p' : p_2, e' : e_2, s'\rangle$.

5. Prova que se $\langle p_1 : p_2, e, s\rangle \triangleright^k \langle \varepsilon, e'', s''\rangle$ então existem k_1 e k_2 com $k = k_1 + k_2$ tal que $\langle p_1, e, s\rangle \triangleright^{k_1} \langle \varepsilon, e', s'\rangle$ e $\langle p_2, e', s'\rangle \triangleright^{k_2} \langle \varepsilon, e'', s''\rangle$.

6. Mostra que a semântica operacional de **AM** é deterministica.

7. ((*) $\mathbf{AM}_1$) Modifica a máquina abstracta **AM** de modo a que não se usem nomes de variáveis mas sim posições de memória. As configurações são agora da forma

   $$\langle p, e, m\rangle \in \mathbf{Code} \times \mathbf{Stack} \times \mathbb{Z}^\star$$

   onde $m \in \mathbb{Z}^\star$ representa uma lista de valores (memória) e $len(m)$ representa o seu comprimento. Considera as instruções GET(n) e PUT(n) onde n é um endereço de memória com $1 \leq n \leq len(m)$ e $m[n]$ dá o valor na posição de memória de endereço n.
   (a) Especifica a semântica operacional da máquina abstracta $\mathbf{AM}_1$
   (b) Traduz a linguagem **While** para $\mathbf{AM}_1$. Supõe um ambiente de variáveis $env : \mathbf{Var} \to \mathbb{N}$ que mapea o nome de variáveis aos seus endereços.

8. ((*) $\mathbf{AM}_2$) Modifica a máquina $\mathbf{AM}_1$ do exercício anterior, para usar instruções de salto (mais habituais) em vez de LOOP e BRANCH. Para tal temos de ter instruções que definem etiquetas e saltos para etiquetas.

   - As configurações serão da forma $\langle pc, p, e, m\rangle$ onde pc é o contador de programa (um inteiro que aponta para uma instrução em p. Em geral o valor de pc é incrementado de 1 em cada passo.

- Em vez de LOOP e BRANCH definem-se as instruções:
  - LABEL-ℓ: que actualiza o contador de programa com ℓ (um inteiro que corresponde a endereço duma instrução)
  - JUMP-ℓ: mover o contador de programa para a instrução LABEL-ℓ (se existir tem de ser única).
  - JUMPFALSE-ℓ: como anterior mas só se no topo da pilha está o valor false; se for true o contador de programa é incrementado

Especifica a semântica operacional da máquina abstracta $\mathbf{AM}_2$. Podes referir por $p[pc]$ a instrução na posição pc.

Semântica operacional de AM e tradução da linguagem While

$$
\begin{array}{rcl}
\langle \textsc{push}(n) : p, e, s\rangle & \triangleright & \langle p, \mathcal{N}[\![n]\!] : e, s\rangle \\
\langle \textsc{add} : p, z_1 : z_2 : e, s\rangle & \triangleright & \langle p, (z_1 + z_2) : e, s\rangle \quad \text{se } z_1, z_2 \in \mathbb{Z} \\
\langle \textsc{mult} : p, z_1 : z_2 : e, s\rangle & \triangleright & \langle p, (z_1 * z_2) : e, s\rangle \quad \text{se } z_1, z_2 \in \mathbb{Z} \\
\langle \textsc{sub} : p, z_1 : z_2 : e, s\rangle & \triangleright & \langle p, (z_1 - z_2) : e, s\rangle \quad \text{se } z_1, z_2 \in \mathbb{Z} \\
\langle \textsc{true} : p, e, s\rangle & \triangleright & \langle p, \mathsf{true} : e, s\rangle \\
\langle \textsc{false} : p, e, s\rangle & \triangleright & \langle p, \mathsf{false} : e, s\rangle \\
\langle \textsc{eq} : p, z_1 : z_2 : e, s\rangle & \triangleright & \begin{cases} \langle p, \mathsf{true} : e, s\rangle \text{ se } z_1 = z_2 \text{ e } z_i \in \mathbb{Z} \\ \langle p, \mathsf{false} : e, s\rangle \text{ caso contrário e } z_i \in \mathbb{Z} \end{cases} \\
\langle \textsc{le} : p, z_1 : z_2 : e, s\rangle & \triangleright & \begin{cases} \langle p, \mathsf{true} : e, s\rangle \text{ se } z_1 \leq z_2 \text{ e } z_i \in \mathbb{Z} \\ \langle p, \mathsf{false} : e, s\rangle \text{ caso contrário e } z_i \in \mathbb{Z} \end{cases} \\
\langle \textsc{and} : p, t_1 : t_2 : e, s\rangle & \triangleright & \begin{cases} \langle p, \mathsf{true} : e, s\rangle \text{ se } t_1 = t_2 = \mathsf{true} \\ \langle p, \mathsf{false} : e, s\rangle \text{ caso contrário} \end{cases} \\
\langle \textsc{neg} : p, t_1 : e, s\rangle & \triangleright & \begin{cases} \langle p, \mathsf{true} : e, s\rangle \text{ se } t_1 = \mathsf{false} \\ \langle p, \mathsf{false} : e, s\rangle \text{ se } t_1 = \mathsf{true} \end{cases} \\
\langle \textsc{fetch}(x) : p, e, s\rangle & \triangleright & \langle p, s(x) : e, s\rangle \\
\langle \textsc{store}(x) : p, z : e, s\rangle & \triangleright & \langle p, e, s[x \mapsto z]\rangle \quad \text{se } z \in \mathbb{Z} \\
\langle \textsc{noop} : p, e, s\rangle & \triangleright & \langle p, e, s\rangle \\
\langle \textsc{branch}(p_1, p_2) : p, t : e, s\rangle & \triangleright & \begin{cases} \langle p_1 : p, e, s\rangle \quad \text{se } t = \mathsf{true} \\ \langle p_2 : p, e, s\rangle \quad \text{se } t = \mathsf{false} \end{cases} \\
\langle \textsc{loop}(p_1, p_2) : p, e, s\rangle & \triangleright & \langle p_1 : \textsc{branch}(p_2 : \textsc{loop}(p_1, p_2), \textsc{noop}) : p, e, s\rangle
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{CA}[\![n]\!] & = & \textsc{push}(n) \\
\mathcal{CA}[\![x]\!] & = & \textsc{fetch}(x) \\
\mathcal{CA}[\![a_1 + a_2]\!] & = & \mathcal{CA}[\![a_2]\!] : \mathcal{CA}[\![a_1]\!] : \textsc{add} \\
\mathcal{CA}[\![a_1 * a_2]\!] & = & \mathcal{CA}[\![a_2]\!] : \mathcal{CA}[\![a_1]\!] : \textsc{mult} \\
\mathcal{CA}[\![a_1 - a_2]\!] & = & \mathcal{CA}[\![a_2]\!] : \mathcal{CA}[\![a_1]\!] : \textsc{sub} \\
\mathcal{CB}[\![\texttt{true}]\!] & = & \textsc{true} \\
\mathcal{CB}[\![\texttt{false}]\!] & = & \textsc{false} \\
\mathcal{CB}[\![a_1 = a_2]\!] & = & \mathcal{CA}[\![a_2]\!] : \mathcal{CA}[\![a_1]\!] : \textsc{eq} \\
\mathcal{CB}[\![a_1 \leq a_2]\!] & = & \mathcal{CA}[\![a_2]\!] : \mathcal{CA}[\![a_1]\!] : \textsc{le} \\
\mathcal{CB}[\![\neg b]\!] & = & \mathcal{CB}[\![b]\!] : \textsc{neg} \\
\mathcal{CB}[\![b_1 \wedge b_2]\!] & = & \mathcal{CB}[\![b_2]\!] : \mathcal{CB}[\![b_1]\!] : \textsc{and} \\
\mathcal{CC}[\![x := a]\!] & = & \mathcal{CA}[\![a]\!] : \textsc{store}(x) \\
\mathcal{CC}[\![\texttt{skip}]\!] & = & \textsc{noop} \\
\mathcal{CC}[\![c_1; c_2]\!] & = & \mathcal{CC}[\![c_1]\!] : \mathcal{CC}[\![c_2]\!] \\
\mathcal{CC}[\![\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2]\!] & = & \mathcal{CB}[\![b]\!] : \textsc{branch}(\mathcal{CC}[\![c_1]\!], \mathcal{CC}[\![c_2]\!]) \\
\mathcal{CC}[\![\texttt{while } b \texttt{ do } c]\!] & = & \textsc{loop}(\mathcal{CB}[\![b]\!], \mathcal{CC}[\![c]\!])
\end{array}
$$