Semânticas de Linguagens de Programação - Exercícios 3
Semântica operacional *big-step* para a linguagem While

1. (a) Sendo $s_0 = [x = 3, y = 5]$, determina o estado a após a a execução do comando

$$\texttt{if } x > y \texttt{ then } z := x \texttt{ else } z := y$$

(b) Sendo $s_0 = [x = 3]$, determinar o estado após a execução de:

$$y := 1; \texttt{while } \neg(x = 1) \texttt{ do } (y := y \times x; x := x - 1)$$

(c) Sendo $s_0 = [x = 10, y = 5]$, determinar o estado após a execução de:

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

2. Mostra que são equivalentes os comandos seguintes para quaisquer comandos c_i, $i = 1, 2, 3$.
   (a) $(c_1; c_2); c_3$ e $c_1; (c_2; c_3)$.
   (b) c_1 e $c_1; (\texttt{while } \mathsf{false} \texttt{ do } c_2)$
   (c) $\texttt{while } b \texttt{ do } c_1$ e $\texttt{if } b \texttt{ then } (c_1 \,; \texttt{while } b \texttt{ do } c_1) \texttt{ else skip}$

3. Considera o comando $\texttt{for } x := a_1 \texttt{ to } a_2 \texttt{ do } c$.
   (a) Define uma semântica operacional natural para esse comado (sem usar a do `while`).
   (b) Sendo $s_0 = [x = 5]$, determinar o estado após a execução de:

$$y := 1; \texttt{for } z := 1 \texttt{ to } x \texttt{ do } (y := y \times x; x := x - 1)$$

4. Considera o comando $\texttt{repeat } c \texttt{ until } b$.
   (a) Define uma semântica operacional natural para esse comado (sem usar a do `while`).
   (b) Mostra que $\texttt{repeat } c \texttt{ until } b$ é semânticamente equivalente a

$$c; \texttt{if } b \texttt{ then skip else } (\texttt{repeat } c \texttt{ until } b)$$

   (c) Mostra que $\texttt{repeat } c \texttt{ until } b$ é semânticamente equivalente a

$$c \,; \texttt{while } \neg b \texttt{ do } c$$

5. A semântica das expressões aritméticas pode ser definida usando a relação $\rightarrow$ considerando os seguintes tipos de configurações: $\langle a, s \rangle$ e z (significando um valor de $\mathbb{Z}$). A relação de transição é definida por $\langle a, s \rangle \rightarrow z$.

   Considerando a semântica dada por $\mathcal{A}$ especifica o sistema de transições para **Aexp** e mostra que a sua semântica coincide com a de $\mathcal{A}$.

6. Repete o exercício anterior mas considerando as expressões booleanas **Bexp**.

7. Determina se os seguintes comandos são equivalentes na semântica natural, para quaisquer comandos c_0, c_1 e c_2 e condição b:

$$c_0 \;;\; \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2$$

   e

$$\mathsf{if}\ b\ \mathsf{then}\ (c_0; c_1)\ \mathsf{else}\ (c_0; c_2).$$

8. Implementação em Haskell da semântica operacional natural dada para a linguagem **While**. Podes usar os seguintes tipos para a memória, as expressões e comandos.

```
module SNWhile
where

type Var = String
type Num = Int

type Mem = [ (Var, Int) ]

data Com = Atrib Var Aexp
             | Skip
             | Comp Com Com
             | If Bexp Com Com
             | While Bexp Com

data Aexp = Var Var
          | Num Int
          | Sum Aexp Aexp
          | Mult Aexp Aexp
          | Sub  Aexp Aexp

data Bexp = Btrue
          | Bfalse
          | Eq Aexp Aexp
          | Le Aexp Aexp
          | Not Bexp
          | And Bexp Bexp
```

Nota: os tipos acima simplificam a sintaxe da linguagem **While** e poderás alterar.

(a) Sendo $s \in \mathbf{Mem}$, i.e. `s::Mem`, define funções para:
   i. Encontrar o valor duma variável x, $s(x)$: `value :: Var -> Mem -> Int`
   ii. Dada uma variável x e um inteiro v, modificar a memória s para $s[x \mapsto v]$: `upd :: Var -> Mem -> Int -> Mem`

(b) Para $\mathcal{A}[\![\cdot]\!]\, s$ define uma função `semA :: Aexp -> Mem -> Int`

(c) Para $\mathcal{B}[\![\cdot]\!]\, s$ define uma função `semB :: Bexp -> Mem -> Bool`

(d) Dado $c \in \mathbf{Com}$, para calcular $\langle c, s\rangle \longrightarrow s'$ define uma função `snCom :: Com -> Mem -> Mem`

(e) Testa a tua implementação codificando os comandos do exercício 1 e avaliando-os para diversos valores de memória (estados).

9. Implementar o exercício anterior em Python.

Semântica operacional para Expressões (aritméticas e booleanas)

Seja $\mathbf{Mem} = [\mathbf{Var} \to \mathbb{Z}]$ e $s \in \mathbf{Mem}$. Para cada $a \in \mathbf{Aexp}$, $\mathcal{A}[\![a]\!] : \mathbf{Mem} \hookrightarrow \mathbb{Z}$:

$$
\begin{array}{rcl}
\mathcal{A}[\![n]\!]\, s & = & \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]\, s & = & \begin{cases} s(x) & \text{se } x \in dom(s) \\ indefinido & \text{caso contrário} \end{cases} \\
\mathcal{A}[\![a_1 + a_2]\!]\, s & = & \begin{cases} z_1 + z_2 & \text{se } z_1 = \mathcal{A}[\![a_1]\!]\, s \text{ e } z_2 = \mathcal{A}[\![a_2]\!]\, s \\ indefinido & \text{caso contrário} \end{cases} \\
\mathcal{A}[\![a_1 - a_2]\!]\, s & = & \begin{cases} z_1 - z_2 & \text{se } z_1 = \mathcal{A}[\![a_1]\!]\, s \text{ e } z_2 = \mathcal{A}[\![a_2]\!]\, s \\ indefinido & \text{caso contrário} \end{cases} \\
\mathcal{A}[\![a_1 * a_2]\!]\, s & = & \begin{cases} z_1 \cdot z_2 & \text{se } z_1 = \mathcal{A}[\![a_1]\!]\, s \text{ e } z_2 = \mathcal{A}[\![a_2]\!]\, s \\ indefinido & \text{caso contrário} \end{cases}
\end{array}
$$

$\mathbb{T} = \{\mathsf{true}, \mathsf{false}\}$ e $\mathcal{B} : \mathbf{Bexp} \to (\mathbf{Mem} \hookrightarrow \mathbb{T})$

$$
\begin{array}{rcl}
\mathcal{B}[\![\mathtt{true}]\!]\, s & = & \mathsf{true} \\
\mathcal{B}[\![a_1 = a_2]\!]\, s & = & \begin{cases} \mathsf{true} & \text{se } z_1 = \mathcal{A}[\![a_1]\!]\, s, z_2 = \mathcal{A}[\![a_2]\!]\, s \text{ e } z_1 = z_2 \\ \mathsf{false} & \text{se } z_1 = \mathcal{A}[\![a_1]\!]\, s, z_2 = \mathcal{A}[\![a_2]\!]\, s \text{ e } z_1 \neq z_2 \\ indefinido & \text{caso contrário} \end{cases} \\
\mathcal{B}[\![a_1 \leq a_2]\!]\, s & = & \begin{cases} \mathsf{true} & \text{se } z_1 = \mathcal{A}[\![a_1]\!]\, s, z_2 = \mathcal{A}[\![a_2]\!]\, s \text{ e } z_1 \leq z_2 \\ \mathsf{false} & \text{se } z_1 = \mathcal{A}[\![a_1]\!]\, s, z_2 = \mathcal{A}[\![a_2]\!]\, s \text{ e } z_1 > z_2 \\ indefinido & \text{caso contrário} \end{cases} \\
\mathcal{B}[\![\neg b]\!]\, s & = & \begin{cases} \mathsf{true} & \text{se } \mathcal{B}[\![b]\!]\, s = \mathsf{false} \\ \mathsf{false} & \text{se } \mathcal{B}[\![b]\!]\, s = \mathsf{true} \\ indefinido & \text{caso contrário} \end{cases} \\
\mathcal{B}[\![b_1 \wedge b_2]\!]\, s & = & \begin{cases} \mathsf{true} & \text{se } \mathcal{B}[\![b_1]\!]\, s = \mathsf{true} \text{ e } \mathcal{B}[\![b_2]\!]\, s = \mathsf{true} \\ indefinido & \text{se } \mathcal{B}[\![b_1]\!]\, s \text{ ou } \mathcal{B}[\![b_2]\!]\, s \text{ são indefinidos} \\ \mathsf{false} & \text{caso contrário .} \end{cases}
\end{array}
$$

Semântica operacionais para While

Semântica operacional natural (*big-step*)

$$
\begin{array}{rl}
\texttt{att}_{sn} & \langle x := a, s\rangle \ \longrightarrow\ s[x \mapsto \mathcal{A}[\![a]\!]\, s] \\
\texttt{skip}_{sn} & \langle \texttt{skip}, s\rangle \ \longrightarrow\ s \\
\texttt{comp}_{sn} & \dfrac{\langle c_1, s\rangle \ \longrightarrow\ s' \ ,\ \langle c_2, s'\rangle \ \longrightarrow\ s''}{\langle c_1; c_2, s\rangle \ \longrightarrow\ s''} \\
\texttt{if}^{v}{}_{sn} & \dfrac{\langle c_1, s\rangle \ \longrightarrow\ s'}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2, s\rangle \ \longrightarrow\ s'} \ \text{se } \mathcal{B}[\![b]\!]\, s = \mathsf{true} \\
\texttt{if}^{f}{}_{sn} & \dfrac{\langle c_2, s\rangle \ \longrightarrow\ s'}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2, s\rangle \ \longrightarrow\ s'} \ \text{se } \mathcal{B}[\![b]\!]\, s = \mathsf{false} \\
\texttt{while}^{v}{}_{sn} & \dfrac{\langle c, s\rangle \ \longrightarrow\ s',\ \langle \texttt{while}\ b\ \texttt{do}\ c, s'\rangle \ \longrightarrow\ s''}{\langle \texttt{while}\ b\ \texttt{do}\ c, s\rangle \ \longrightarrow\ s''} \\
 & \text{se } \mathcal{B}[\![b]\!]\, s = \mathsf{true} \\
\texttt{while}^{f}{}_{sn} & \langle \texttt{while}\ b\ \texttt{do}\ c, s\rangle \ \longrightarrow\ s \ \text{se } \mathcal{B}[\![b]\!]\, s = \mathsf{false}
\end{array}
$$