Semânticas de Linguagens de Programação - Exercícios 2
Semânticas operacionais *big-step* e *small-step* para a linguagem While

Semântica das expressões

1. Sendo $s(x) = 2$ calcula
   (a) $\mathcal{A}[\![x + 2]\!]s$
   (b) $\mathcal{A}[\![x * y)]\!]s$
   (c) $\mathcal{B}[\![\neg(x < 1)]\!]s$
2. Sejam s e s' tal que $s(x) = s'(x)$ para todo $x \in FV(a)$. Mostra que $\mathcal{A}[\![a]\!]s = \mathcal{A}[\![a]\!]s'$. Análogo para $b \in$ **Bexp**.
3. Mostra que $\mathcal{A}[\![a[y \mapsto a_0]]\!]s = \mathcal{A}[\![a]\!]s[y \mapsto \mathcal{A}[\![a_0]\!]s]$.
4. Define a substituição para expressões booleanas $b[x \mapsto a_0]$ e mostra a proposição equivalente ao do exercício anterior.

Semântica operacional natural (*big-step*) da linguagem While

5. (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:

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

6. Mostra que são equivalentes os comandos
   (a) $(c_1; c_2); c_3$ e $c_1; (c_2; c_3)$.
   (b) `while` b `do` c e `if` b `then` $(c$; `while` b `do` $c)$ `else skip`
7. Considera o comando `for` $x := a_1$ `to` a_2 `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:

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

8. Considera o comando `repeat` c `until` b.
   (a) Define uma semântica operacional natural para esse comado (sem usar a do `while`).

(b) Mostra que `repeat` c `until` b é semânticamente equivalente a

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

(c) Mostra que `repeat` c `until` b é semânticamente equivalente a

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

9. 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 $\mathcal{A}$.

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

11. Determina se os seguintes comandos são equivalentes na semântica natural, para quaisquer comandos c_0, c_1 e c_2e condição b:

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

e

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

12. 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 $s \in$ **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 $c \in$ **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 5 e avaliando-os para diversos valores de memória (estados).

Semântica operacional estrutural (*small-step*) da linguagem While

13. Repete o exercício 5 considerando a semântica operacional estrutural dada no curso.

14. Define uma semântica operacional estrutural para o comando `repeat` c `until` b (sem usar a do `while`).

15. Define uma semântica operacional estrutural para o comando `for` $x := a_1$ `to` a_2 `do` c (sem usar a do `while`).

16. Mostra que para todo o $k \geq 1$, se $\langle c_1, s\rangle \Rightarrow^k s'$ então $\langle c_1; c_2, s\rangle \Rightarrow^k \langle c_2, s'\rangle$.

17. Mostra que a semântica operacional estrutural dada para a linguagem **While** é determinística.

18. Mostra a equivalência semântica dos seguintes comandos:
   (a) `skip`; c e c
   (b) `while` b `do` c e `if` b `then` (c ; `while` b `do` c) `else skip`

19. Implementa em Haskell a semântica operacional estrutural para a linguagem **While**, usando as estruturas de dados definidas no exercício 12

Semântica operacional para Expressões (aritméticas e booleanas)

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

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

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

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

Semântica operacionais para While

Semântica operacional natural (*big-step*)

$$\texttt{att}_{sn} \quad \langle x := a, s\rangle \longrightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\texttt{skip}_{sn} \quad \langle \texttt{skip}, s\rangle \longrightarrow s$$

$$\texttt{comp}_{sn} \quad \frac{\langle c_1, s\rangle \longrightarrow s' \ , \ \langle c_2, s'\rangle \longrightarrow s''}{\langle c_1; c_2, s\rangle \longrightarrow s''}$$

$$\texttt{if}^{v}{}_{sn} \quad \frac{\langle c_1, s\rangle \longrightarrow s'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, s\rangle \longrightarrow s'} \text{ se } \mathcal{B}[\![b]\!]s = \mathsf{true}$$

$$\texttt{if}^{f}{}_{sn} \quad \frac{\langle c_2, s\rangle \longrightarrow s'}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, s\rangle \longrightarrow s'} \text{ se } \mathcal{B}[\![b]\!]s = \mathsf{false}$$

$$\texttt{while}^{v}{}_{sn} \quad \frac{\langle c, s\rangle \longrightarrow s', \ \langle \texttt{while } b \texttt{ do } c, s'\rangle \longrightarrow s''}{\langle \texttt{while } b \texttt{ do } c, s\rangle \longrightarrow s''} \text{ se } \mathcal{B}[\![b]\!]s = \mathsf{true}$$

$$\texttt{while}^{f}{}_{sn} \quad \langle \texttt{while } b \texttt{ do } c, s\rangle \longrightarrow s \text{ se } \mathcal{B}[\![b]\!]s = \mathsf{false}$$

Semântica operacional estrutural (*small-step*)

$$\texttt{att}_{sos} \quad \langle x := a, s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$\texttt{skip}_{sos} \quad \langle \texttt{skip}, s\rangle \Rightarrow s$$

$$\texttt{comp}^{1}{}_{sos} \quad \frac{\langle c_1, s\rangle \Rightarrow \langle c_1', s'\rangle}{\langle c_1; c_2, s\rangle \Rightarrow \langle c_1'; c_2, s'\rangle}$$

$$\texttt{comp}^{2}{}_{sos} \quad \frac{\langle c_1, s\rangle \Rightarrow s'}{\langle c_1; c_2, s\rangle \Rightarrow \langle c_2, s'\rangle}$$

$$\texttt{if}^{v}{}_{sos} \quad \langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, s\rangle \Rightarrow \langle c_1, s\rangle \text{ se } \mathcal{B}[\![b]\!]s = \mathsf{true}$$

$$\texttt{if}^{f}{}_{sos} \quad \langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, s\rangle \Rightarrow \langle c_2, s\rangle \text{ se } \mathcal{B}[\![b]\!]s = \mathsf{false}$$

$$\texttt{while}_{sos} \quad \langle \texttt{while } b \texttt{ do } c, s\rangle \Rightarrow \langle \texttt{if } b \texttt{ then}(c; \texttt{while } b \texttt{ do } c) \texttt{ else skip}, s\rangle$$