Programação Ciber-Física

[TPC-2]

Ana Paula Oliveira Henriques
Mestrado em Engenharia Informática
Universidade do Minho, Departamento de Informática
4710-057 Braga, Portugal
e-mail: pg50196@alunos.uminho.pt

1 Registo de mensagens

O problema em questão pretende registar a velocidade de um carro periodicamente. Para tal, é adotada esta linguagem de programação:

$$\texttt{Prog(X)} \; \exists \texttt{x} \; \texttt{:=} \; \texttt{t} \; | \; \texttt{write}_m\texttt{(p)} \; | \; \texttt{p;q} \; | \; \texttt{if b then p else q} \; | \; \texttt{while b do \{p\}}$$

A ideia é que o programa $\texttt{write}_m\texttt{(p)}$ escreva a lista de mensagens `m` e depois corra o programa `p`. No decorrer do presente trabalho, serão utilizadas as regras de semântica apresentadas a seguir:

$$\frac{\langle \texttt{t}, \sigma \rangle \Downarrow \texttt{r}}{\langle \texttt{x} := \texttt{t}, \sigma \rangle \Downarrow [\,], \sigma[\texttt{r}/\texttt{x}]} \text{ (asg)} \qquad \frac{\langle \texttt{p}, \sigma \rangle \Downarrow n, \sigma'}{\langle \texttt{write}_\texttt{m}(\texttt{p}), \sigma \rangle \Downarrow m +\!\!+\, n, \sigma'} \text{ (write)}$$

$$\frac{\langle \texttt{p}, \sigma \rangle \Downarrow m, \sigma' \qquad \langle \texttt{q}, \sigma' \rangle \Downarrow n, \sigma''}{\langle \texttt{p} \,;\, \texttt{q}, \sigma \rangle \Downarrow m +\!\!+\, n, \sigma''} \text{ (seq)}$$

$$\frac{\langle \texttt{b}, \sigma \rangle \Downarrow \texttt{tt} \qquad \langle \texttt{p}, \sigma \rangle \Downarrow m, \sigma'}{\langle \texttt{if b then p else q}, \sigma \rangle \Downarrow m, \sigma'} \text{ (if}_1\text{)} \qquad \frac{\langle \texttt{b}, \sigma \rangle \Downarrow \texttt{ff} \qquad \langle \texttt{q}, \sigma \rangle \Downarrow m, \sigma'}{\langle \texttt{if b then p else q}, \sigma \rangle \Downarrow m, \sigma'} \text{ (if}_2\text{)}$$

$$\frac{\langle \texttt{b}, \sigma \rangle \Downarrow \texttt{tt} \qquad \langle \texttt{p}, \sigma \rangle \Downarrow m, \sigma' \qquad \langle \texttt{while b do \{ p \}}, \sigma' \rangle \Downarrow n, \sigma''}{\langle \texttt{while b do \{ p \}}, \sigma \rangle \Downarrow m +\!\!+\, n, \sigma''} \text{ (wh}_1\text{)}$$

$$\frac{\langle \texttt{b}, \sigma \rangle \Downarrow \texttt{ff}}{\langle \texttt{while b do \{ p \}}, \sigma \rangle \Downarrow [\,], \sigma} \text{ (wh}_2\text{)}$$

Figure 1: Regras de semântica

1.1 Exercício 1

No primeiro exercício 1, é requerido que provemos a seguinte equivalência:

$$\mathtt{write}_m\mathtt{(write}_n\mathtt{(p))} \sim \mathtt{write}_{m+\!\!+n}\mathtt{(p)}$$

Também é dito que, para dois programas p e q serem equivalentes (p $\sim$ q), então, para todos os ambientes σ, $\langle \mathtt{p},\sigma\rangle \Downarrow \mathtt{m},\sigma\mathtt{'}$ iff $\langle \mathtt{q},\sigma\rangle \Downarrow \mathtt{m},\sigma\mathtt{'}$.
Consideremos $\langle \mathtt{p},\sigma\rangle$ como $\langle \mathtt{write}_m\mathtt{(write}_n\mathtt{(p))},\sigma\rangle$ e $\langle \mathtt{q},\sigma\rangle$ como $\langle \mathtt{write}_{m+\!\!+n}\mathtt{(p)},\sigma\rangle$:

- Assumindo que $\langle \mathtt{write}_m\mathtt{(write}_n\mathtt{(p))},\ \sigma\rangle \Downarrow \mathtt{m+\!\!+n+\!\!+n},\ \sigma\mathtt{'}$:

$$\dfrac{\langle p, \sigma\rangle \Downarrow n, \sigma'}{\langle \text{write}_{m+\!\!+n}\,(p), \sigma\rangle \Downarrow m+\!\!+n+\!\!+n, \sigma'}\ (\text{write})$$

- Assumindo que $\langle \mathtt{write}_{m+\!\!+n}\mathtt{(p)},\ \sigma\rangle \Downarrow \mathtt{m+\!\!+n+\!\!+n},\ \sigma\mathtt{'}$:

$$\dfrac{\dfrac{\langle p, \sigma\rangle \Downarrow n, \sigma'}{\langle \text{write}_n\,(p), \sigma\rangle \Downarrow n+\!\!+n, \sigma'}\ (\text{write})}{\langle \text{write}_m\,(\text{write}_n\,(p)), \sigma\rangle \Downarrow m+\!\!+n+\!\!+n, \sigma'}\ (\text{write})$$

Posto isto, conseguimos, então, provar que os dois programas são equivalentes.

Podemos, ainda, pensar noutras equivalências interessantes para que o compilador possa conhecer variadas formas de otimização; por exemplo:

(1) $\mathtt{write}_m\mathtt{(p)} \sim \mathtt{while\ i<1\ do}\ \{\mathtt{write}_m\mathtt{(p);i:=i+1}\}$

- Assumindo que $\langle \mathtt{write}_m\mathtt{(p)},\ \sigma\rangle \Downarrow \mathtt{m+\!\!+n},\ \sigma\mathtt{'}$ e que o *clock* i toma como valor inicial 0:

$$\dfrac{\langle i<1, \sigma\rangle \Downarrow tt \quad \dfrac{\dfrac{\langle p, \sigma\rangle \Downarrow n, \sigma'}{\langle \text{write}_m\,(p), \sigma\rangle \Downarrow m+\!\!+n, \sigma'}\ (\text{write}) \quad \dfrac{\langle i+1, \sigma'\rangle \Downarrow r}{\langle i := i+1, \sigma'\rangle \Downarrow [\,], \sigma'[r/i]}\ (\text{asg})}{\langle \text{write}_m\,(p)\ ;\ i := i+1, \sigma\rangle \Downarrow m+\!\!+n, \sigma'}\ (\text{seq}) \quad \dfrac{\langle i<1, \sigma\rangle \Downarrow ff}{\langle \text{while}\ i<1\ \text{do}\ \{\ \text{write}_m\,(p)\ ;\ i := i+1\ \}, \sigma'\rangle \Downarrow [\,], \sigma'}\ (\text{wh2})}{\langle \text{while}\ i<1\ \text{do}\ \{\ \text{write}_m\,(p)\ ;\ i := i+1\ \}, \sigma\rangle \Downarrow m+\!\!+n, \sigma'}\ (\text{wh1})$$

$$\boxed{m+\!\!+n+\!\!+[\,] = m+\!\!+n}$$

- Assumindo que $\langle \mathtt{while\ i<1\ do}\ \{\mathtt{write}_m\mathtt{(p);i:=i+1}\},\sigma\rangle \Downarrow \mathtt{m+\!\!+n},\ \sigma\mathtt{'}$ e que o *clock* i toma como valor inicial 0:

$$\dfrac{\langle p, \sigma\rangle \Downarrow n, \sigma'}{\langle \text{write}_m\,(p), \sigma\rangle \Downarrow m+\!\!+n, \sigma'}\ (\text{write})$$

(2) $\texttt{write}_m\texttt{(write}_{[]}\texttt{(p))} \sim \texttt{if true then write}_m\texttt{(p) else write}_{[]}\texttt{(p)}$

- Assumindo que $\langle\texttt{write}_m\texttt{(write}_{[]}\texttt{(p))}, \sigma\rangle \Downarrow \texttt{m++n}, \sigma\texttt{'}$:

$$\dfrac{\langle \text{true}, \sigma\rangle \Downarrow \text{tt} \qquad \dfrac{\langle \text{p}, \sigma\rangle \Downarrow \text{n}, \sigma'}{\langle \text{write}_\text{m}\,(\text{p}), \sigma\rangle \Downarrow \text{m++n}, \sigma'}\,(\text{write})}{\langle \text{if true then write}_\text{m}\,(\text{p})\text{ else write}_{[]}\,(\text{p}), \sigma\rangle \Downarrow \text{m++n}, \sigma'}\,(\text{if1})$$

- Assumindo que $\langle\texttt{if true then write}_m\texttt{(p) else write}_{[]}\texttt{(p)}, \sigma\rangle \Downarrow \texttt{m++n}, \sigma\texttt{'}$:

$$\dfrac{\dfrac{\langle \text{p}, \sigma\rangle \Downarrow \text{n}, \sigma'}{\langle \text{write}_{[]}\,(\text{p}), \sigma\rangle \Downarrow [\,]\text{++n}, \sigma'}\,(\text{write})}{\langle \text{write}_\text{m}\,(\text{write}_{[]}\,(\text{p})), \sigma\rangle \Downarrow \text{m++}[\,]\text{++n}, \sigma'}\,(\text{write})$$

$$\boxed{\textbf{m++[]++n = m++n}}$$

1.2 Exercício 2

Para o segundo exercício é solicitada a implementação em Haskell da linguagem descrita na Figura 1, bem como as suas regras de semântica. Todos os ficheiros contruídos para a concretização dessa implementação estão disponíveis no mesmo `.zip` que continha este `.pdf`, na pasta `src`. Além disso, esses ficheiros encontram-se devidamente documentados.

1.2.1 Semântica para Termos Lineares

```haskell
module LinearTerm where

data Vars = X | Y | Z deriving (Show, Eq)

data LinearTerm = Leaf (Either Double Vars)
        | Scl Double LinearTerm
        | Add LinearTerm LinearTerm deriving Show

semLinear :: LinearTerm -> (Vars -> Double) -> Double
semLinear (Leaf (Left r)) m = r
semLinear (Leaf (Right x)) m = m x
semLinear (Scl d t) m = let r = semLinear t m
                            in d * r
semLinear (Add t1 t2) m = let r1 = semLinear t1 m
                              r2 = semLinear t2 m
                            in r1 + r2
```

1.2.2 Semântica para Termos Booleanos

```
module BooleanTerm where

import LinearTerm

data BooleanTerm = Comp LinearTerm LinearTerm
        | Neg BooleanTerm
        | And BooleanTerm BooleanTerm deriving Show

boolTerm :: BooleanTerm -> (Vars -> Double) -> Bool
boolTerm (Comp t1 t2) m = let r1 = semLinear t1 m
                              r2 = semLinear t2 m
                            in r1 <= r2
boolTerm (Neg b) m = let v = boolTerm b m in not v
boolTerm (And b1 b2) m = let v1 = boolTerm b1 m
                             v2 = boolTerm b2 m
                            in v1 && v2
```

1.2.3 Semântica para Programas *While*

```
module WhileTerm where

import LinearTerm
import BooleanTerm

data WhileTerm = Asg Vars LinearTerm
        | Write [Int] WhileTerm
        | Seq WhileTerm WhileTerm
        | Ife BooleanTerm WhileTerm WhileTerm
        | Whi BooleanTerm WhileTerm deriving Show

asgMem :: Vars -> Double -> (Vars -> Double) -> (Vars -> Double)
asgMem x d mem = \v -> if v == x then d else mem v

semWhile :: WhileTerm -> (Vars -> Double) -> ([Int], (Vars -> Double))
semWhile (Asg v t) mem = let r = semLinear t mem
                            in ([], asgMem v r mem)
semWhile (Write msg p) mem = let (msgs, mem') = semWhile p mem
                                in (msg ++ msgs, mem')
semWhile (Seq p q) mem = let (msgs1, mem') = semWhile p mem
                             (msgs2, mem'') = semWhile q mem'
                            in (msgs1 ++ msgs2, mem'')
```

```
semWhile (Ife b p q) mem = if (boolTerm b mem)
                           then semWhile p mem
                           else semWhile q mem
semWhile (Whi b p) mem = if boolTerm b mem
                          then let (msgs1, mem') = semWhile p mem
                                   (msgs2, mem'') = semWhile (Whi b p) mem'
                                  in (msgs1 ++ msgs2, mem'')
                         else ([], mem)
```

1.2.4 Testes

```
module Tests where

import LinearTerm
import BooleanTerm
import WhileTerm

x,y,z :: Vars
x = X
y = Y
z = Z

mem :: Vars -> Double
mem _ = 0.0

test1 = Asg x (Leaf (Left 5.0))

test2 = Seq (
            Asg x (Leaf (Left 5.0)))
            (Write [13232,0,12] (Write [123,98] (Asg y (Leaf (Left 2.0))))
        )

test3 = Ife (
            Comp (Leaf (Right x)) (Leaf (Right y)))
            (Asg z (Leaf (Left 1.0)))
            (Asg z (Leaf (Left 2.0))
        )

test4 = Whi (
            Comp (Leaf (Right x)) (Leaf (Right y)))
            (Asg x (Add (Leaf (Right x)) (Leaf (Right y)))
        )

test5 = Write [1,2,3,4,5] test1
```

```haskell
main :: IO ()
main = do
    let (msgs1, mem1) = semWhile test1 mem
    putStrLn "Resultado do teste 1:"
    putStrLn ("Mensagens: " ++ show msgs1)
    putStrLn ("Memória X: " ++ show (mem1 x))
    putStrLn ("Memória Y: " ++ show (mem1 y))
    putStrLn ("Memória Z: " ++ show (mem1 z))
    putStrLn ""

    let (msgs2, mem2) = semWhile test2 mem1
    putStrLn "Resultado do teste 2:"
    putStrLn ("Mensagens: " ++ show msgs2)
    putStrLn ("Memória X: " ++ show (mem2 x))
    putStrLn ("Memória Y: " ++ show (mem2 y))
    putStrLn ("Memória Z: " ++ show (mem2 z))
    putStrLn ""

    let (msgs3, mem3) = semWhile test3 mem2
    putStrLn "Resultado do teste 3:"
    putStrLn ("Mensagens: " ++ show msgs3)
    putStrLn ("Memória X: " ++ show (mem3 x))
    putStrLn ("Memória Y: " ++ show (mem3 y))
    putStrLn ("Memória Z: " ++ show (mem3 z))
    putStrLn ""

    let (msgs4, mem4) = semWhile test4 mem3
    putStrLn "Resultado teste 4:"
    putStrLn ("Mensagens: " ++ show msgs4)
    putStrLn ("Memória X: " ++ show (mem4 x))
    putStrLn ("Memória Y: " ++ show (mem4 y))
    putStrLn ("Memória Z: " ++ show (mem4 z))
    putStrLn ""

    let (msgs5, mem5) = semWhile test5 mem4
    putStrLn "Resultado teste 5:"
    putStrLn ("Mensagens: " ++ show msgs5)
    putStrLn ("Memória X: " ++ show (mem5 x))
    putStrLn ("Memória Y: " ++ show (mem5 y))
    putStrLn ("Memória Z: " ++ show (mem5 z))
    putStrLn ""
```