Linguagem Imperativa Simples

Linguagem Imperativa Simples

- Atribuição, if-then-else, while-loop, expressões aritméticas.
- Um único tipo de dados, `Integer`.
- Sem declaração de variáveis.
- Expressões sem efeitos colaterais.
- Sem funções, arrays, records, etc.
- Resultado final é o valor da variável `result`.

Exemplo: cálculo de fatorial

```
x = 10;
r = 1;
while x do
  r = r * x;
  x = x - 1;
result = r
```

Sintaxe Abstrata de Comandos

```
data Cmd =
     CmdAsg Var Exp          -- assignment (var = exp)
   | CmdIf Exp Cmd Cmd       -- if exp then cmd else cmd
   | CmdSeq Cmd Cmd          -- cmd; cmd
   | CmdWhile Exp Cmd        -- while exp do cmd
   | CmdSkip                 -- do nothing
```

Obs: `C1; C2; C3` → `{C1; C2}; C3`

Sintaxe Abstrata de Expressões

```
data Exp = ExpN Integer                 -- constants
         | ExpVar Var                   -- variables
         | ExpAdd Exp Exp               -- e1 + e2
         | ExpSub Exp Exp               -- e1 - e2
         | ExpMul Exp Exp               -- e1 * e2
         | ExpDiv Exp Exp               -- e1 / e2
         | ExpNeg Exp                   -- -e
           ...
```

Exemplo: cálculo de fatorial

```
x = 10;
r = 1;
while x do
  r = r * x;
  x = x - 1;
result = r
```

```
let x = ExpVar "x"
    r = ExpVar "r" in
  CmdSeq
    (CmdSeq
      (CmdSeq (CmdAsg "x" (ExpN 10))
              (CmdAsg "r" (ExpN 1)))
      (CmdWhile (ExpVar "x")
                (CmdSeq (CmdAsg "r" (ExpMul r x))
                        (CmdAsg "x" (ExpSub x (ExpN 1))))))
    (CmdAsg "result" r)
```

Semântica

- Qual o significado de um comando?
  - por exemplo, `x = 10` ?
- Qual o valor de uma variável em uma expressão?
  - por exemplo, `x + 10 - y` ?
- Precisamos de um *estado*, que é modificado por comandos e consultado por expressões
  - efeitos colaterais de linguagens imperativas!!

O Estado

- Para nossa linguagem simples, o estado pode ser apenas uma *memória*, que mapeia variáveis para seus valores correntes:
  - `type Var = String`
  - `type Value = Integer`
  - `type Mem = Var -> Value`

Memória

```
type Mem = Var -> Value

-- An empty memory
emptyMem :: Mem
emptyMem v =
  error ("invalid access to variable " ++ v)

-- update the value of a variable in a memory
update :: Var -> Value -> Mem -> Mem
update var val m = (\v -> if v == var then val
                                      else m v)
```

Uma Implementação Alternativa

```
type Mem = [(Var, Value)]

emptyMem :: Mem
emptyMem = []

queryMem :: Var -> Mem -> Value
queryMem v [] = error (“undefined variable “ ++ v)
queryMem v ((var, val):m) =
     if v == var then val else (queryMem v m)

update :: Var -> Value -> Mem -> Mem
update var val m = (var, val) : m
```

Semântica de Expressões

```
evalExp :: Exp -> Mem -> Value

evalExp (ExpN i) m = i
evalExp (ExpVar v) m = m v
evalExp (ExpAdd e1 e2) m = (evalExp e1 m) + (evalExp e2 m)
evalExp (ExpSub e1 e2) m = (evalExp e1 m) - (evalExp e2 m)
evalExp (ExpMul e1 e2) m = (evalExp e1 m) * (evalExp e2 m)
evalExp (ExpDiv e1 e2) m =
                           (evalExp e1 m) `div` (evalExp e2 m)
evalExp (ExpNeg e) m = -(evalExp e m)
   ...
```

Semântica de Comandos (1)

```
evalCmd :: Cmd -> Mem -> Mem

evalCmd (CmdSkip) m = m

evalCmd (CmdAsg v e) m = update v (evalExp e m) m

evalCmd (CmdIf e ct ce) m = if isTrue (evalExp e m)
                              then (evalCmd ct m)
                              else (evalCmd ce m)
```

Semântica de Comandos (2)

```
evalCmd (CmdSeq c1 c2) m = evalCmd c2 (evalCmd c1 m)

-- Para o while, usamos novamente a idéia de “desenrolar” um nivel:
-- while E do C ≡ if E then {C; while E do C} else skip
evalCmd (CmdWhile e c) m = w m
  where w m = if isTrue (evalExp e m)
                then w (evalCmd c m)
                else m
```

Observações

- Uso de recursão “mal fundada”
- `while 1 do skip`
  - `(CmdWhile (ExpN 1) CmdSkip)`:

```
w m = if isTrue (evalExp (ExpN 1) m)
                then w (evalCmd CmdSkip m)
                else m
    = w (evalCmd CmdSkip m)
    = w m
```