Universidade Federal de Minas Gerais
Ciência da Computação

Linguagens de Programação - Haniel Barbosa

Lista de Exercícios 2

Elaborada por José Wesley Magalhães

Linguagem de expressões

1. Utilizando o ADT `expr` visto em aula:

   ```
   datatype expr = IConst of int | Plus of expr * expr | Minus of expr * expr;
   ```

   Estenda essa linguagem com os seguintes operadores:

   - Multiplicação. `(Multi)` ⇒ multiplica dois valores.
   - Divisão. `(Div)` ⇒ divisão inteira de dois valores. Divisão por 0 deve retornar 0.
   - Maior valor. `(Max)` ⇒ retorna o maior de dois valores.
   - Menor Valor. `(Min)` ⇒ retorna o menor de dois valores.
   - Igual. `(Eq)` ⇒ retorna 1 se os valores são iguais, 0 caso contrário.
   - Maior que. `(Gt)` ⇒ retorna 1 se o primeiro valor é estritamente maior que o segundo, e 0 caso contrário.

   Estenda também a função `eval : expr -> int` para conseguir avaliar expressões que utilizem esses operadores.

   input: val e1 = Max(IConst 3, Plus(IConst 2, IConst 3));
   output: val it = 5 : int

   input: val e2 = Div(Multi(IConst 5, IConst 4), Minus(IConst 4, IConst 4));
   output: val it = 0 : int

2. Escreva uma linguagem que calcule a área de objetos quadrados, retangulares, e circulares. Você deve definir o ADT `area`. Os nomes dos construtores de `area` devem seguir o seguinte padrão:

   ```
   datatype area = RConst of real | AQuadrado of area | ACirculo ...
   ```

   Defina também a função `eval : area -> real` para realizar a interpretação dessas expressões. **IMPORTANTE:** as medidas desses objetos deveram ser do tipo `real`.

   input: val e = ACirculo(RConst 2.0);
   output: val it = 12.56: real;

3. Utilizando as mesmas convenções da questão anterior, defina um ADT `perimetro` e sua função `eval : perimetro -> real`. Inclua também suporte para calcular perímetro de triângulos (`PTriangulo`).

   input: val p = PQuadrado(RConst 4.0);
   output: val it = 16.0: real;

4. Compiladores frequentemente aplicam otimizações com intuito de deixar o código gerado mais eficiente. Um exemplo comum é a simplificação de expressões, utilizando propriedades aritméticas e/ou Booleanas. Dados os ADT's abaixo:

   ```
   datatype UnOp = Not;

   datatype BinOp = Add | Sub | Mul | Gt | Eq | Or;

   datatype Sexpr = IConst of int | Op1 of UnOp * Sexpr | Op2 of BinOp * Sexpr
   * Sexpr;
   ```

   Escreva uma função `simplify : Sexpr -> Sexpr` que seja capaz de simplificar expressões `Sexpr` de acordo com as regras de simplificação listadas abaixo ($\vee$ simboliza disjunção lógica e $\neg$ simboliza negação lógica):

$$0 + e \to e$$
$$e + 0 \to e$$
$$e - 0 \to e$$
$$1 * e \to e$$
$$e * 1 \to e$$
$$0 * e \to 0$$
$$e * 0 \to 0$$
$$e - e \to 0$$
$$e \vee e \to e$$
$$\neg(\neg e) \to e$$

   A sua função deve ser capaz de simplificar, por exemplo, as expressões $x + 0$, $1 * x$, e $(1 + 0) * (x + 0)$ para somente x. O retorno deve ser uma expressão que não possa ser mais simplificada.

   DICAS:

   - Não se esqueça dos casos em que não é possível mais simplificar.
   - Passos recursivos podem ser necessários para produzir expressões que não são simplificáveis.
   - "You ain't never had a friend like pattern matching."

//(1+0)*(9+0) → 9
input: Op2(Mul, Op2(Add, IConst 1, IConst 0), Op2(Add, IConst 9, IConst 0));
output: val it = IConst 9: Sexpr;

//(1+0)*((10 ∨ 12) +0) → (10 ∨ 12)
input: Op2 (Mul, Op2 (Add, IConst 1, IConst 0), Op2 (Add, Op2 (Or, IConst 10, IConst 12), IConst 0)): Sexpr;
output: val it = Op2 (Or, IConst 10, IConst 12): Sexpr;

Semântica formal

As questões a seguir são retiradas do livro *Semantics with Applications: An Appetizer*, Capítulos 1 e 2. Portanto a leitura é necessária para o entendimento das questões. Os dois primeiros capítulos encontram-se disponíveis no Moodle.

1. Suponha que o valor inicial da variável x é n e o valor inicial de y é m. Escreva um programa em **WHILE** que atribui a Z o valor de n^m.

2. Considere a função de interpretação de expressões Booleanas de **WHILE**, $\mathcal{B}$ (Tabela 1.2), e suponha que $s\ x = 3$. Determine $\mathcal{B}[\![\neg(x = 1)]\!]$.

3. Defina uma substituição para expressões Booleanas de **WHILE**: $b[y \mapsto a_0]$ deve ser a expressão Booleana correspondente a b exceto que todas as ocorrências da variável y são substituídas pela expressão aritmética a_0.

4. Dado o programa

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

   construa uma árvore de derivação para este programa quando executado em um estado em que x tem valor **17** e y tem valor **5**.

5. Considere os seguintes programas:

   - while ¬(x=1) do (y:=y*x; x:=x-1)
   - while 1 ≤ x do (y:=y*x; x:=x-1)
   - while true do skip

   Para cada um deles determine se eles *sempre* terminam ou se sempre entram em um laço infinito. Tente embasar suas repostas usando os axiomas e regras da Tabela 2.1.

6. Considere a seguinte AST e interpretador parcial para linguagem **WHILE**:

```
type Num = int;
type Var = string;

datatype Aexpr =
         N of Num
         | V of Var
         | Plus of Aexpr * Aexpr
         | Mult of Aexpr * Aexpr
         | Minus of Aexpr * Aexpr;

datatype Bexpr =
         True
       | False
       | Eq of Aexpr * Aexpr
       | Leq of Aexpr * Aexpr
       | Not of Bexpr
       | And of Bexpr * Bexpr;

datatype Stm =
         Assign of Var * Aexpr
       | Skip
       | Comp of Stm * Stm
       | If of Bexpr * Stm * Stm
       | While of Bexpr * Stm;

fun evalN n : Num = n

exception FreeVar;
fun lookup [] id = raise FreeVar
  | lookup ((k:string, v)::l) id = if id = k then v else lookup
      l id;

fun evalA (N n) _ = evalN n
  | evalA (V x) s = lookup s x
  | evalA (Plus(e1, e2)) s = (evalA e1 s) + (evalA e2 s)
  | evalA (Mult(e1, e2)) s = (evalA e1 s) * (evalA e2 s)
  | evalA (Minus(e1, e2)) s = (evalA e1 s) - (evalA e2 s);

fun evalB True _ = true
  | evalB False _ = false
  | evalB (Eq(a1, a2)) s = (evalA a1 s) = (evalA a2 s)
  | evalB (Leq(a1, a2)) s = (evalA a1 s) <= (evalA a2 s)
  | evalB (Not b) s = not (evalB b s)
  | evalB (And(b1, b2)) s = (evalB b1 s) andalso (evalB b2 s);

fun evalStm (stm : Stm) (s : (string * int) list) : (string *
   int) list =
```

```
46      case stm of
47          (Assign(x, a)) => (x, evalA a s)::s
48        |  Skip => s
49        | (Comp(stm1, stm2)) => evalStm stm2 (evalStm stm1 s)
50        | (If(b, stm1, stm2)) =>
51          if (evalB b s) then evalStm stm1 s else evalStm stm2 s
52        (* | While(b, stm) =>  ... *)
53        | _ => raise Match;
```

(a) Estenda o interpretador com o tratamento do comando while, seguindo a semântica da Tabela 2.1.

(b) Estenda a linguagem com o comando

repeat S until b

e defina a relação $\rightarrow$ para ele. (A semântica de repeat não deve utilizar o operador while da linguagem).

(c) Estenda o interpretador acima para o comando repeat.

(d) Demonstre que repeat S until b e S ; if b then skip else (repeat S until b) são semanticamente equivalentes.

Binding, escopo

1. Considere o seguinte programa em uma linguagem de programação genérica:

```
1  func p {
2      x: integer;
3
4      func q{
5          x := x + 1;
6      }
7
8      func r{
9          x: integer;
10         x := 1;
11         q;
12         write(x);
13     }
14
15     x := 2;
16     r;
17 }
```

Qual é a saída desse programa (linha 12) ao executar p:

(a) Caso essa linguagem possua escopo estático?

(b) Caso esta linguagem possua escopo dinâmico?

2. Considere o programa abaixo, escrito em SML:

```
fun g x =
  let
    val inc = 1
    fun f y = y + inc
    fun h z =
      let
        val inc = 2
      in
        f z
      end
  in
    h x
  end
```

(a) Enumere cada bloco que esse programa contém. Exemplo: Escopo de g = bloco 1, etc.

(b) Quais são os nomes definidos nesse programa?

(c) Para cada definição, descreva seu escopo em termos dos números de blocos que você definiu no item a.

(d) Tente responder sem executar o programa. Qual é o valor de `g 5`? Qual seria esse valor se SML possuísse escopo dinâmico? Explique o motivo de esses valores serem diferentes.