

Universidade Federal de Minas Gerais  
Ciência da Computação

Linguagens de Programação - Haniel Barbosa

## Lista de Exercícios 2

### 1 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`)  $\Rightarrow$  multiplica dois valores.
- Divisão. (`Div`)  $\Rightarrow$  divisão inteira de dois valores. Divisão por 0 deve retornar 0.
- Maior valor. (`Max`)  $\Rightarrow$  retorna o maior de dois valores.
- Menor Valor. (`Min`)  $\Rightarrow$  retorna o menor de dois valores.
- Igual. (`Eq`)  $\Rightarrow$  retorna 1 se os valores são iguais, 0 caso contrário.
- Maior que. (`Gt`)  $\Rightarrow$  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 | ARetangulo of area *  
area | ACirculo of area
```

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):

$$0 + e \rightarrow e$$

$$e + 0 \rightarrow e$$

$$e - 0 \rightarrow e$$

$$1 * e \rightarrow e$$

$$e * 1 \rightarrow e$$

$$0 * e \rightarrow 0$$

$$e * 0 \rightarrow 0$$

$$e - e \rightarrow 0$$

$$e \vee e \rightarrow e$$

$$\neg(\neg e) \rightarrow e$$

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.”

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

**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;

## 2 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

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

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  $\neg(x=1)$  do  $(y:=y*x; x:=x-1)$
  - while  $1 \leq x$  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.

- while  $\neg(x=1)$  do  $(y:=y*x; x:=x-1)$
- while  $1 \leq x$  do  $(y:=y*x; x:=x-1)$
- while true do skip

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

```
1  type Num = int;
2  type Var = string;
3
4  datatype Aexpr =
5      N of Num
6      | V of Var
7      | Plus of Aexpr * Aexpr
8      | Mult of Aexpr * Aexpr
9      | Minus of Aexpr * Aexpr;
10
11 datatype Bexpr =
12     True
13     | False
14     | Eq of Aexpr * Aexpr
15     | Leq of Aexpr * Aexpr
16     | Not of Bexpr
17     | And of Bexpr * Bexpr;
18
19 datatype Stm =
20     Assign of Var * Aexpr
21     | Skip
22     | Comp of Stm * Stm
23     | If of Bexpr * Stm * Stm
24     | While of Bexpr * Stm;
25
26 fun evalN n : Num = n
27
28 exception FreeVar;
29 fun lookup [] id = raise FreeVar
30     | lookup ((k:string, v)::l) id = if id = k then v else lookup
31                                     l id;
32
33 fun evalA (N n) _ = evalN n
34     | evalA (V x) s = lookup s x
35     | evalA (Plus(e1, e2)) s = (evalA e1 s) + (evalA e2 s)
36     | evalA (Mult(e1, e2)) s = (evalA e1 s) * (evalA e2 s)
37     | evalA (Minus(e1, e2)) s = (evalA e1 s) - (evalA e2 s);
38
39 fun evalB True _ = true
40     | evalB False _ = false
41     | evalB (Eq(a1, a2)) s = (evalA a1 s) = (evalA a2 s)
42     | evalB (Leq(a1, a2)) s = (evalA a1 s) <= (evalA a2 s)
43     | evalB (Not b) s = not (evalB b s)
44     | evalB (And(b1, b2)) s = (evalB b1 s) andalso (evalB b2 s);
45
46 fun evalStm (stm : Stm) (s : (string * int) list) : (string *
47     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.

### 3 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:

```

1 fun g x =
2   let
3     val inc = 1
4     fun f y = y + inc
5     fun h z =
6       let
7         val inc = 2
8         in
9           f z
10        end
11    in
12      h x
13    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.
3. Escreva uma função principal count\_main: `int -> int list` que receba um número inteiro como entrada e conte de 1 em 1 até chegar neste número. Escreva uma segunda função count: `int -> int list` que receba o primeiro número que deva começar a contagem e faça qualquer regra necessária dentro desta função. A função count deve funcionar apenas dentro do escopo da função count\_main.

//sendo 1 o primeiro número

**input:** count\_main(5);

**output:** val it = [1, 2, 3, 4, 5] : int list

4. Escreva uma função pow: `int -> int` que receba um número e retorne a seu resultado elevado a 2, utilize uma outra função calculePow: `int -> int` para fazer o cálculo, ela deve estar dentro do escopo da função pow.

**input:** pow(3);

**output:** val it = 9 : int

5. A seguinte função `bad_max` encontra o maior número de uma lista e o retorna:

```
1 fun bad_max(xs: int list) =  
2   if null xs  
3   then 0  
4   else if null (tl xs)  
5   then hd xs  
6   else if hd xs > bad_max(tl xs)  
7   then hd xs  
8   else bad_max(tl xs);
```

- (a) Explique porque essa função `bad_max` apesar de funcionar não é a forma mais correta de ser implementada.
- (b) Escreva uma função `good_max: int list -> int` que corrija o problema da função `bad_max` dada anteriormente:
6. Escreva uma função `split: 'a list -> 'a list * 'a list` que receba uma lista de números e divida essa lista em outras duas listas tal que o primeiro elemento esteja na primeira lista, o segundo na segunda, o terceiro na primeira e assim vai. Note que se a lista contiver um número ímpar de elementos o elemento restante deve ficar à esquerda.

**input:** `split([1, 2, 3, 8, 4, 5])`

**output:** `val it = ([1, 3, 4], [2, 8, 5])`

7. Dada a função `expr: unit -> int` substitua os valores e a expressão matemática da linha 5, e informe qual o valor que será impresso. Por substituir, entenda-se passar para a sintaxe de SML, o que corresponde a fazer a declaração de variáveis com um dado valor e usá-las no novo escopo criado. Note que por exemplo `let val n = e1 in e2 end` é uma expressão que tem como resultado a avaliação de `e2`.

```
1 fun expr () =  
2   let  
3     val x = 1  
4   in  
5     (* (x = 2, x + 1) + (y = x + 2, y + 1) *)  
6   end;
```