

MAC 316 – Conceitos Fund. de Linguagens de Programação

Prof: Ana C. V. de Melo

Monitor: Eduardo Sândalo Porto

Trabalho – Interpretador

Número de integrantes da equipe: 5 (máximo)

Prazo de Entrega: até dia 10/12/2022

Interpretador - programação funcional

Considere uma linguagem funcional simplificada (LFSimp) definida informalmente como segue (sintaxe concreta):

Simbolos/palavras da linguagem : (,), +, -, *, ~, lambda, call,
if, let, letrec, cons, head, tail

```
-- Elementos básicos
<character> ::= <letter> | <digit> | <symbol>
<letter>   ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
               "J" | "K" | "L" | "M" | "N" | "O" | "P" |
               "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" |
               "Z" | "a" | "b" | "c" | "d" | "e" | "f" |
               "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" |
               "p" | "q" | "r" | "s" | "t" | "u" | "v" |
               "w" | "x" | "y" | "z"
<digit>    ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<intnum>   ::= <digit> | <digit><intnum>
<number>   ::= <intnum> | <intnum>."<intnum>
<symbol>::=  "_" | "!" | "?" | "-" | "+" | "*" | "?" | "%" | "<" | ">" | "#" | "~"
<id>::= <character> | <character><id>

-- Código do programa
<code> ::= "(" <expr> ")" | <number>

<expr> ::= <arith_expr> | <list_expr> | <lamb_expr> | <let_expr> | <if> | <quote>

-- expressões aritméticas
<arith_expr> ::= <plus> | <mult> | <bminus> | <uminus>
<plus>      ::= "+" <code> <code>
<mult>      ::= "*" <code> <code>
<bminus>    ::= "-" <code> <code>
<uminus>    ::= "~" <code>

-- expressões sobre listas
<list_expr> ::= <cons> | <head> | <tail>
<cons>::= "cons" <code> <code>
<head>::= "head" <code>
<tail>  ::= "tail" <code>

-- expressões lambda
<lamb_expr> ::= <lambda> | <call>
```

```

<lambda> ::= "lambda" <param> <code>
<param>  ::= <id>
<call>   ::= "call" <lambda> <code>

-- expressões let
<let_expr> ::= <let> | <letrec>
<let>      ::= "let" <id> <def> <body>
<letrec>   ::= "letrec" <id> <lambda> <body>
<def>      ::= <code>
<body>     ::= <code>

-- expressões if
<if>       ::= "if" <cond> <pos> <neg>
<cond>     ::= <code>
<pos>      ::= <code>
<neg>      ::= <code>

-- expressões quote
<quote>    ::= <code> | <id>

```

Essa linguagem não possui declaração de tipos e cada um dos operadores aritméticos é aplicado a expressões. A verificação da compatibilidade dos tipos é realizada mediante a aplicação dos operadores, se inteiros, listas ou outras expressões.

As listas são construídas com o construtor **cons**, e as operações **head** e **tail** só podem ser aplicadas a listas construídas na linguagem. Cada lista é construída com pelo menos 2 elementos (veja os vários testes sugeridos, junto ao código do interpretador).

Além das listas, a LFSimp contém expressões aritméticas, expressões lambda, expressões **if** e expressões **let** e **letrec**. Estes últimos definem nomes que podem ser associados a quaisquer valores de expressões disponíveis na linguagem de forma simplificada ou recursiva, respectivamente. Para ver como funcionam as expressões da linguagem, leia os comentários no [Readme.md](#) e no código disponível, e faça pelo menos os testes sugeridos nos arquivos: `testes_basicos_LFSimpl.txt` e `Tests.hs` (veja como executar os testes nos próprios arquivos).

A execução de um programa é dada pela avaliação da expressão. No interpretador, essa avaliação é precedida por outras tarefas, como descrito no código correspondente (veja os comentários no código do interpretador):

```
interp = eval . desugar . analyze . parse . tokenize
```

Para a sintaxe aqui definida, alguns elementos são permitidos na escrita das expressões mas podem ser problemáticos na execução (erro na avaliação da expressão). Por exemplo, sob o ponto-de-vista sintático, uma expressão aritmética admite que cada um dos operandos seja uma expressão qualquer da linguagem, mas na avaliação (execução) a operação só pode ser realizada sobre números. Nesses casos, o erro só será apontado na avaliação da expressão. Isso também acontece com outras operações sobre listas... (veja nos testes sugeridos). Isso significa que parte da verificação de tipos é realizada em tempo de execução das expressões.

As Suas Tarefas Antes de iniciar as tarefas aqui definidas, V. deve ler o código fornecido e executar os testes. Os itens abaixo não vão fazer sentido para vocês antes disso.

1. Formação dos identificadores: Um dos problemas na linguagem é quando definimos identificadores nas expressões **lambda**, **let** e **letrec** que são números, ou palavras/símbolos reservados à linguagem. Nesses casos, as expressões podem ficar confusas, ou ainda gerar resultados errados ao esperado (vide os testes sugeridos). Para eliminar esses tipos de problemas, os identificadores usados nas expressões **não podem ser** símbolos ou palavras reservadas da linguagem (ex.: `let`, `+`, `)`), nem devem ser um número (ex.: `2`, `34`), ou começar por um número (ex.: `5as`, `2f`, `23c`). A sua tarefa será modificar o código fornecido para que os identificadores respeitem essas novas regras de formação.

2. Implementação da expressão case: Esse comando tem 3 elementos principais: `<num>`, `<intlist>` e `<codelist>`. `<num>` é um número inteiro que vai ser calculado e comparado com os números inteiros da lista `<intlist>`. Se o número é encontrado na i -ésima posição da lista de inteiros, então o código na i -ésima posição da lista de códigos (`<codelist>`) será executado. A formação da expressão é dada pela seguinte sintaxe:

```
<case> ::= "case" <num> "of" <intlist> <codelist>
<num> ::= <arith_expr> | <intnum>
<intlist> ::= "cons" <intnum> <intnum>
<codelist> ::= "cons" <code> <code>
```

Exemplo corrigido:

```
(case (+ 6 4) of (cons (* 5 2) 8) (cons (* (+ 1 3) (- 10 20)) (if 0 (+ 1 2) (+ 10 20))))
```

Neste caso, calculamos $(+ 6 4) = 10$ e comparamos com os valores na lista $(cons (* 5 2) 8)$. Como esse valor é igual ao primeiro elemento da lista de inteiros $(* 5 2) = 10$, então o primeiro elemento da lista de códigos $(* (+ 1 3) (- 10 20))$ será executado como resultado da expressão. Como resultado da avaliação desta expressão, teríamos "numV = -40.0".

Para simplificar a implementação da expressão, vamos assumir que o tamanho da lista `<intlist>` é igual ao tamanho da lista `<codelist>`, temos um código a ser executado para cada valor. Além disso, vamos assumir que não existem valores repetidos em `<intlist>`. Essas restrições evitam que algumas verificações sejam realizadas.

O que está sendo fornecido O código do interpretador que funciona para a linguagem atual, e dois conjuntos de testes.

O que V. deve entregar O código do interpretador modificado com os itens das tarefas a serem realizadas. **Não mude os nomes dos arquivos e nem das funções já implementadas** pq vamos fazer os testes com esses nomes. Se achar pertinente, acrescente novas funções, tipos...

Como o seu EP será avaliado Vamos submeter o seu código a um conjunto de testes, tanto para o que já funciona quanto para os novos itens solicitados na ep atual.