Conceitos em linguagem de programação - Segundo compilador

Projeto da disciplina MAC0316. Este é o projeto de uma linguagem simples implementada em racket. A parte da gramatica é feita pelos arquivos `tradutor.` [l|y], que transformam a gramática explícita da linguagem em uma linguagem interna para ser interpretada por `solver.rkt`

Como buildar

Para executar o projeto precisamos instalar as dependencias e criar o executavel. Basta executar as seguintes linhas:

```
$ sudo apt-get install flex
$ sudo apt-get install bison
$ sudo apt-get install racket
$ raco pkg install plai-typed
$ make all
```

Segundo compilador

As mudanças feitas para essa segunda entrega são:

- Foi adicionado um environment de escopo dinâmico;
- Nomeclatura foi mudada, de *arith* para *expr*;
- Não temos mais uma biblioteca de funções, agora elas são implementadas no core da linguagem;
- Foi criada a closure, o pacote funcao + environment:
  - `fdC` se tornou `lamC` ;
  - nomes para as funcoes se tornaram desnecessarios. Apesar disso, na gramática explicíta exigimos que todas funções tenham nome.
- Foram criadas as Boxes;
- Todas funções sao associadas a um simbolo, recursivas ou não.

A Linguagem

Para chamar nosso compilador, basta dar um *pipe* entre `tradutor` e `solver` e depois digitar a expressão.

Todos os comandos devem ser finalizados com ";" e para terminar o programa utiliza-se o comando FIM

O ";" cria o sequenciamento entre comandos.

0) Operações Básicas

Os operadores básicos são `*, /, +, -,` responsáveis respectivamente por multiplicação, divisão, soma e subtração de expressões.

Para realizar uma operação, utiliza-se *expressao1 operador expressao2*.

Exemplos:

`3*6`

`4*(10/2)`

Além disso, existe o operador IF condicional, sua sintaxe é

`(<condicao>) ? (<caso_verdadeiro>) (<caso_falso>)`

Se condicao é igual a 0, casoFalso é executado. Caso contrário, casoVerdadeiro é executado.

1) Criação de variáveis e mutação

Para definir uma variável, utiliza-se a sintaxe `LET <nome_variavel> := (<valor_inicial>);`

Para mudar o valor da variável, utiliza-se `<nome_variavel> := (<novo_valor>);`

Exemplos:

```
$ ./tradutor | ./solver
> LET x := (4);
> x+1;
> FIM;
5

$ ./tradutor | ./solver
> LET x := (2);
> LET y := (3);
> x := (x+y);
> FIM
5

$ ./tradutor | ./solver
> LET x := (2+(5*3));
> LET y := (x + (5+2));
> x := (x+2);
> x+y;
> FIM
43
```

2) Funções

Para definir uma função, a sintaxe é `FUNC <nome_funcao> (<parametro>) (<corpo>);`

Para chamar uma função, utiliza-se `CALL <nome_funcao> (<parametro>);`

As funções pré-implementadas são

1. `dobro(x)` : retorna o dobro de x
2. `quadrado(x)` : retorna o quadrado de x
3. `fatorial(n)` : retorna o valor do fatorial de n
4. `resposta(x)` : recebe um valor de x e retorna a resposta para a vida, o universo e tudo mais.
5. `fibo(n)` : retorna o n-ésimo número de fibonnaci.

Exemplos:

```
$ ./tradutor | ./solver
> FUNC somaUm(x){
>    x+1;
> };
> CALL somaUm(1);
> FIM
2

$ ./tradutor | ./solver
> FUNC funcaoSofisticada(x){
>    LET y := (x+1);
>    y;
> };
> CALL funcaoSofisticada(1);
> FIM
2

$ ./tradutor | ./solver
FUNC tribonacci(x){
    x ?
      ((x-1) ?
            ((x-2) ?
              ( (CALL tribonacci(x-1)) + (CALL tribonacci(x-2)) + (CALL tribonacci(x-3)))
              (2)
              )
            (1)
              )
          (0);
};
> CALL tribonacci(5);
> FIM
11
```

Grupo

- Carolina Senra Marques - NUSP: 10737101
- Felipe Castro de Noronha - NUSP: 10737032

- Raphael Ribeiro Costa e Silva - NUSP: 10281601