UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Ronistone Junior - 11521BCC018

Relatório Parcial 6 - Construção de Compiladores

March 2019

Sumário

	Sumário	**1**
0.1	**Instalação**	**4**
0.1.1	Instalando o OCaml	4
0.1.2	Instalando JVM e Assembler	4
0.2	**Gerando e Compilando o Assembly Dos Arquivos**	**4**
0.2.1	Compilando	5
0.2.2	Desassemblando	5
0.2.3	Compilando o Assembly	5
0.3	**Assembly Gerado**	**5**
0.3.1	Nano01	6
0.3.1.1	Comentários	6
0.3.1.2	Diretivas	6
0.3.1.3	Métodos	7
0.3.2	Nano02	7
0.3.3	Nano03	8
0.3.4	Nano04	9
0.3.5	Nano05	9
0.3.6	Nano06	10
0.3.7	Nano07	11
0.3.8	Nano08	13
0.3.9	Nano09	14
0.3.10	Nano10	15
0.3.11	Nano11	17
0.3.12	Nano12	19
0.4	**Principais Instruções do Jasmin**	**21**
0.4.1	.bytecode	22
0.4.2	.source	22
0.4.3	.class	22
0.4.4	.super	22
0.4.5	.method	22
0.4.6	.limit	22
0.4.7	.end	22
0.4.8	.line	22
0.4.9	lload	22
0.4.10	lstore	22

0.4.11 iaload . . . 23
0.4.12 iload . . . 23
0.4.13 istore . . . 23
0.4.14 fload . . . 23
0.4.15 fstore . . . 23
0.4.16 ldc . . . 23
0.4.17 dup . . . 23
0.4.18 pop . . . 23
0.4.19 swap . . . 23
0.4.20 iadd . . . 23
0.4.21 idiv . . . 24
0.4.22 imul . . . 24
0.4.23 isub . . . 24
0.4.24 fadd . . . 24
0.4.25 fdiv . . . 24
0.4.26 fmul . . . 24
0.4.27 fsub . . . 24
0.4.28 goto . . . 24
0.4.29 ifeq . . . 24
0.4.30 ifge . . . 24
0.4.31 ifgt . . . 24
0.4.32 ifle . . . 25
0.4.33 iflt . . . 25
0.4.34 ifne . . . 25
0.4.35 if_icmple . . . 25
0.4.36 if_icmpne . . . 25
0.4.37 if_icmpe . . . 25
0.4.38 iand . . . 25
0.4.39 ior . . . 25
0.4.40 i2f . . . 25
0.4.41 f2i . . . 25
0.4.42 jsr . . . 25
0.4.43 ret . . . 26
0.4.44 invokevirtual . . . 26
0.4.45 invokeinterfacel . . . 26
0.4.46 invokespecial . . . 26
0.4.47 invostatic . . . 26
0.4.48 getstatic . . . 26
0.5 Analisador Léxico . . . 26

0.5.1 Implementação 26
0.5.2 Execução 29
0.6 Analisador Sintático 31
0.6.1 Preparando o Ambiente 31
0.6.2 Árvore Sintática Abstrata 32
0.6.3 Gramática 34
0.6.4 Execução 38
0.6.5 Compilando 40
0.7 Analisador Semântico 40
0.8 Interprete 54
0.8.1 ambinterp.ml 54
0.8.2 ambInterp.mli 55
0.8.3 interprete.ml 56
0.8.4 interprete.mli 65
0.8.5 interpreteTest.ml 65
0.9 Geração de Código Intermediário 67
0.9.1 Cod3End.ml 67
0.9.2 cod3endTest.ml 75
0.9.3 Codigo.ml 78

0.1 Instalação

0.1.1 Instalando o OCaml

Utilizando os comandos:

```
1 sudo apt install wget
2 sudo apt install git aspcud m4 mercurial darcs
3 wget https://raw.github.com/ocaml/opam/master/shell/opam_installer.sh -O -
    | sh -s /usr/local/bin
4 opam init
5 opam update
6 eval `opam config env`
7 sudo apt-get install rlwrap
```

foi instalado o OPAM, rlwrap e o OCaml, com todas as suas dependências

0.1.2 Instalando JVM e Assembler

- Como já possuía java instalado não foi necessário instalação da JVM, estou utilizando o Java 8.
- Já para instalar o jasmin, que é o assembler da JVM, baixei o source no seguinte link Após o download temos um arquivo zip que contém o jar do jasmin.
- Foi utilizado também o disassembler ClassFileAnalyzer (Can) que pode ser baixado no seguinte link
- Para instalar o Assembler e o Disassembler extrai os dois .zip para o meu /opt utilizando unzip nome-do-arquivo.zip e criei um script no /usr/bin para cara um para executa-los, abaixo estão os dois scripts:

```
1 # /usr/bin/run-jasmin
2 java -jar /opt/jasmin-2.4/jasmin.jar $@
```

```
1 # /usr/bin/generate-dot-j
2 java -jar /opt/classfileanalyzer/classfileanalyzer.jar $@
```

0.2 Gerando e Compilando o Assembly Dos Arquivos

Todos os códigos utilizados foram disponibilizados pelo professor no forum do google groups da matéria, utilizaremos os arquivos "Nanos", que podem ser baixados neste link.

Aqui vamos gerar os arquivos .j que é utilizado pelo Jasmin para gerar como estamos utilizando um disassembler para gerar o assembly de forma mais rápida, seguiremos o seguinte processo:

1. Compilaremos um arquivo .java utilizando o javac
2. Depois de compilado teremos o .class, usaremos ele no disassembler para gerar o .j
3. Agora podemos manipular o assembly da forma que for necessária e ao fim utilizar o Jasmin para compilar o nosso assembly

0.2.1 Compilando

Para compilar entrar na pasta que se encontra os arquivos .java e executar:

```
1       javac *.java
```

0.2.2 Desassemblando

Para desassemblar e gerar os arquivos .j para cada código java, utilizaremos o disassembler ClassFileAnalyzer para isto criei um pequeno script para realizar este processo:

```
1       # ./generate.sh
2       files=`ls *.class`
3
4       for i in $files
5       do
6               name=$( echo "$i" | cut -f 1 -d '.' )
7               echo 'Generating ' ${name} ' ...'
8               generate-dot-j $i > $name.j
9       done;
```

0.2.3 Compilando o Assembly

Após geramos os arquivos .j temos o nosso assembly gerado, caso queiramos compilar este assembly basta utilizarmos o nosso Assembler Jasmin, para realizar este processo criei o script abaixo que irá compilar todos os nossos assembly:

```
1 # ./compile.sh
2 files=`ls *.j`
3
4 for i in $files
5 do
6  name=$( echo "$i" | cut -f 1 -d '.' )
7  echo 'Compiling ' ${name} '...'
8  run-jasmin $i
9 done;
```

0.3 Assembly Gerado

Vamos agora entender um pouco do nosso assembly gerado.

0.3.1 Nano01

No Nano01 tem uma função main que não executa nenhum operação

```
1  ; Nano01.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano01.java
12 .class public Nano01
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 0
26   .limit locals 1
27   .line 3
28   0: return
29 .end method
```

0.3.1.1 Comentários

Aqui temos nosso primeiro assembly, a primeira coisa que podemos notar é que para escrever comentários basta utilizar o ";"(ponto e virgula) e tudo que estiver na frente do ";"faz parte do comentário.

0.3.1.2 Diretivas

Logo depois notamos algumas palavras iniciadas com ".", essas são as chamadas Diretivas. Elas nada mais são que palavras utilizadas para controle pelo Jasmin, ou seja, elas não são instruções da JVM. Podemos ver as seguintes diretivas: .bytecode, .source, .class, .super, .method, .limit, .line, .end. As diretivas estão melhor explicadas na seção [0.4]

0.3.1.3 Métodos

Podemos notar dois métodos, sendo o primeiro o init que é o construtor da classe e o método main.

Dentro do construtor podemos ver a chamada o construtor da classe Object que é uma super classe desta nossa Classe Nano

Dentro do main podemos ver que não é feito quase nada, apenas é definido o tamanho inicial da pilha, o tamanho das variáveis locais e logo em seguida é realizado um return finalizando o método.

0.3.2 Nano02

Declaramos apenas uma variável dentro da nossa classe main.

```
1  ; Nano02.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano02.java
12 .class public Nano02
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 0
26   .limit locals 2
27   .line 4
28   0: return
29 .end method
```

Podemos notar que o este assembly é praticamente identico ao assembly do Nano01 [0.3.1]. Diferindo apenas no .limit locals da função main, podemos ver que agora temos

.limit locals 2 e antes tinhamos .limit locals 1, isto nos mostra que temos mais uma variável local.

0.3.3 Nano03

Declaramos e definimos um valor para uma variável.

```
1  ; Nano03.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano03.java
12 .class public Nano03
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 1
26   .limit locals 2
27   .line 4
28   0: iconst_1
29   1: istore_1
30   .line 5
31   2: return
32 .end method
```

Aqui podemos ver um limite maior para nossa pilha, e duas novas instruções, o limite maior se dá devido a definição de um valor para a variável. Já as duas novas instruções são iconst_1 carrega a constante 1 no topo da pilha. Enquanto a instrução istore_1 carrega o valor do topo da pilha para a variável 1.

0.3.4 Nano04

Declarando uma variável e atribuindo a soma de duas constantes a esta variável.

```
1  ; Nano04.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano04.java
12 .class public Nano04
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 1
26   .limit locals 2
27   .line 4
28   0: iconst_3
29   1: istore_1
30   .line 5
31   2: return
32 .end method
```

Neste exemplo podemos ver que não há apenas uma diferença com o Nano03 0.3.3 que é o valor da constante agora é 3, podemos concluir disto que ouve uma otimização, pois como estávamos somando duas constantes o compilador carregou o resultado da soma direto para não precisar refazer o calculo em tempo de execução.

0.3.5 Nano05

Declarando, atribuindo valor a uma variável e imprimindo o seu valor.

```
1  ; Nano05.j
2
```

```
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano05.java
12 .class public Nano05
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 2
27   .line 4
28   0: iconst_2
29   1: istore_1
30   .line 5
31   2: getstatic java/lang/System/out Ljava/io/PrintStream;
32   5: iload_1
33   6: invokevirtual java/io/PrintStream/println(I)V
34   .line 6
35   9: return
36 .end method
```

Aqui notamos 3 novas instruções, o getstatic que carrega um campo estático para o topo da pilha, o iload_1 que carrega o valor da variável 1 para o topo da pilha e o invokevirtual que realiza a chamada de um método.

0.3.6 Nano06

Declarando variável, atribuindo uma subtração de duas constantes para a variável e por fim imprimindo a variável.

```
1  ; Nano06.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
```

```
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano06.java
12 .class public Nano06
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 2
27   .line 4
28   0: iconst_m1
29   1: istore_1
30   .line 5
31   2: getstatic java/lang/System/out Ljava/io/PrintStream;
32   5: iload_1
33   6: invokevirtual java/io/PrintStream/println(I)V
34   .line 6
35   9: return
36 .end method
```

Podemos notar aqui que ouve a mesma otimização citada no Nano04 0.3.4, sendo utilizado o iconst_m1 para carregar a constante -1 para a pilha. No mais o assembly é identico ao assembly do 0.3.5.

0.3.7 Nano07

Declarando uma variável, atribuindo valor e verificando se o valor é igual a 1, se for imprime a variável.

```
1  ; Nano07.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
```

```
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano07.java
12 .class public Nano07
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 2
27   .line 4
28   0: iconst_1
29   1: istore_1
30   .line 5
31   2: iload_1
32   3: iconst_1
33   4: if_icmpne Label14
34   .line 6
35   7: getstatic java/lang/System/out Ljava/io/PrintStream;
36   10: iload_1
37   11: invokevirtual java/io/PrintStream/println(I)V
38 Label14:
39   .line 8
40   14: return
41   ; append_frame (frameNumber = 0)
42   ; frame_type = 252, offset_delta = 14
43   ; frame bytes: 252 0 14 1
44   .stack
45     offset 14
46     locals Integer
47     .end stack
48 .end method
```

Agora temos uma nova instrução, o if_icmpne verifica se os dois valores do topo da pilha não são iguais se não forem, pula para o Label14. Para a execução desta instrução podemos ver um preparo para tal, onde é carregado a variável 1 e a constante 1 para o topo da pilha, fazendo então o if_icmpne que corresponde a 'if(n==1)'.

0.3.8 Nano08

Verifica se valor na variável é igual a 1 se for imprime a variável, se não imprime 0.

```
1  ; Nano08.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano08.java
12 .class public Nano08
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 2
27   .line 4
28   0: iconst_1
29   1: istore_1
30   .line 5
31   2: iload_1
32   3: iconst_1
33   4: if_icmpne Label17
34   .line 6
35   7: getstatic java/lang/System/out Ljava/io/PrintStream;
36   10: iload_1
37   11: invokevirtual java/io/PrintStream/println(I)V
38   14: goto Label24
39 Label17:
40   .line 8
41   17: getstatic java/lang/System/out Ljava/io/PrintStream;
42   20: iconst_0
43   21: invokevirtual java/io/PrintStream/println(I)V
44 Label24:
```

```
45    .line 10
46    24: return
47    ; append_frame (frameNumber = 0)
48    ; frame_type = 252, offset_delta = 17
49    ; frame bytes: 252 0 17 1
50    .stack
51      offset 17
52      locals Integer
53      .end stack
54    ; same_frame (frameNumber = 1)
55    ; frame_type = 6, offset_delta = 6
56    ; frame bytes: 6
57    .stack
58      offset 24
59      locals Integer
60      .end stack
61  .end method
```

Aqui temos apenas mais uma instrução nova, o goto, ele simplesmente salta para o um label. Assim podemos ver o if feito no Nano07 0.3.7 com a diferença de mais um bloco de instruções e no primeiro bloco que será executado se o if_icmpne falhar temos o goto para pular para o fim do if, enquanto o if_icmpne salta direto para o bloco de código do else.

0.3.9 Nano09

Atribui o valor de uma expressão a variável e depois verifica o valor com um if e else.

```
1  ; Nano09.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano09.java
12 .class public Nano09
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
18   .line 1
19   0: aload_0
20   1: invokespecial java/lang/Object/<init>()V
21   4: return
```

```
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 2
27   .line 5
28   0: iconst_1
29   1: istore_1
30   .line 6
31   2: iload_1
32   3: iconst_1
33   4: if_icmpne Label17
34   .line 7
35   7: getstatic java/lang/System/out Ljava/io/PrintStream;
36   10: iload_1
37   11: invokevirtual java/io/PrintStream/println(I)V
38   14: goto Label24
39 Label17:
40   .line 9
41   17: getstatic java/lang/System/out Ljava/io/PrintStream;
42   20: iconst_0
43   21: invokevirtual java/io/PrintStream/println(I)V
44 Label24:
45   .line 11
46   24: return
47   ; append_frame (frameNumber = 0)
48   ; frame_type = 252, offset_delta = 17
49   ; frame bytes: 252 0 17 1
50   .stack
51     offset 17
52     locals Integer
53     .end stack
54   ; same_frame (frameNumber = 1)
55   ; frame_type = 6, offset_delta = 6
56   ; frame bytes: 6
57   .stack
58     offset 24
59     locals Integer
60     .end stack
61 .end method
```

Vemos um código idêntico ao Nano08 [0.3.8] devido a otimização do compilador resolvendo a expressão em tempo de compilação.

0.3.10 Nano10

Declara duas variáveis, atribui valores para as variáveis e verifica se são iguais

```
; Nano10.j

; Generated by ClassFileAnalyzer (Can)
; Analyzer and Disassembler for Java class files
; (Jasmin syntax 2, http://jasmin.sourceforge.net)
;
; ClassFileAnalyzer, version 0.7.0


.bytecode 52.0
.source Nano10.java
.class public Nano10
.super java/lang/Object

.method public <init>()V
  .limit stack 1
  .limit locals 1
  .line 1
  0: aload_0
  1: invokespecial java/lang/Object/<init>()V
  4: return
.end method

.method public static main([Ljava/lang/String;)V
  .limit stack 2
  .limit locals 3
  .line 4
  0: iconst_1
  1: istore_1
  .line 5
  2: iconst_2
  3: istore_2
  .line 6
  4: iload_1
  5: iload_2
  6: if_icmpne Label19
  .line 7
  9: getstatic java/lang/System/out Ljava/io/PrintStream;
  12: iload_1
  13: invokevirtual java/io/PrintStream/println(I)V
  16: goto Label26
Label19:
  .line 9
  19: getstatic java/lang/System/out Ljava/io/PrintStream;
  22: iconst_0
  23: invokevirtual java/io/PrintStream/println(I)V
Label26:
```

```
48    .line 11
49    26: return
50    ; append_frame (frameNumber = 0)
51    ; frame_type = 253, offset_delta = 19
52    ; frame bytes: 253 0 19 1 1
53    .stack
54      offset 19
55      locals Integer
56      locals Integer
57      .end stack
58    ; same_frame (frameNumber = 1)
59    ; frame_type = 6, offset_delta = 6
60    ; frame bytes: 6
61    .stack
62      offset 26
63      locals Integer
64      locals Integer
65      .end stack
66  .end method
```

Aqui não temos nada novo, a única diferença para o que já foi visto é que agora a comparação é feita utilizando apenas variáveis, ou seja, antes de executar o if_icmpne carregamos as variáveis 1 e 2 para a pilha ao invés de carregar uma variável e uma constante.

0.3.11 Nano11

Declaração de três variáveis e um while verificando se x ¿, n, realizando n = n + m e imprimindo o valor de n a cada interação.

```
1  ; Nano11.j
2
3  ; Generated by ClassFileAnalyzer (Can)
4  ; Analyzer and Disassembler for Java class files
5  ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6  ;
7  ; ClassFileAnalyzer, version 0.7.0
8
9
10 .bytecode 52.0
11 .source Nano11.java
12 .class public Nano11
13 .super java/lang/Object
14
15 .method public <init>()V
16   .limit stack 1
17   .limit locals 1
```

```
18    .line 1
19    0: aload_0
20    1: invokespecial java/lang/Object/<init>()V
21    4: return
22  .end method
23
24  .method public static main([Ljava/lang/String;)V
25    .limit stack 2
26    .limit locals 4
27    .line 5
28    0: iconst_1
29    1: istore_1
30    .line 6
31    2: iconst_2
32    3: istore_2
33    .line 7
34    4: iconst_5
35    5: istore_3
36  Label6:
37    .line 8
38    6: iload_3
39    7: iload_1
40    8: if_icmple Label25
41    .line 9
42    11: iload_1
43    12: iload_2
44    13: iadd
45    14: istore_1
46    .line 10
47    15: getstatic java/lang/System/out Ljava/io/PrintStream;
48    18: iload_1
49    19: invokevirtual java/io/PrintStream/println(I)V
50    22: goto Label6
51  Label25:
52    .line 12
53    25: return
54    ; append_frame (frameNumber = 0)
55    ; frame_type = 254, offset_delta = 6
56    ; frame bytes: 254 0 6 1 1 1
57    .stack
58      offset 6
59      locals Integer
60      locals Integer
61      locals Integer
62      .end stack
63    ; same_frame (frameNumber = 1)
64    ; frame_type = 18, offset_delta = 18
```

```
65    ; frame bytes: 18
66    .stack
67      offset 25
68      locals Integer
69      locals Integer
70      locals Integer
71      .end stack
72  .end method
```

Agora temos uma estrutura muito similar ao que é realizado nos códigos anteriores onde tinhamos um if, com o while(x¿n) foi gerado um if_icmple que verifica se o primeiro valor é menor ou equal ao segundo, se for pula para um label. Isto é utilizado para marcar o fim do while, caso seja verdadeiro ele pula para o fim do bloco do while, se não ele apenas executa o bloco.

Dentro do bloco vemos uma n = n + m, que é feito da seguinte forma, carrega-se n e m para a pilha depois executamos iadd para somar os dois valores do topo da pilha e salvamos o resultado no topo da pilha, por fim armazenamos o resultado que esta no topo da pilha na variável 1, realizamos a impressão do valor e pulamos de volta para o label que marca a inicialização para o if_icmple.

0.3.12 Nano12

Realiza dentro do while do Nano11 [0.3.11] agora uma verificação se n==m se for imprime n senão imprime 0, e por fim decrementa x.

```
1   ; Nano12.j
2
3   ; Generated by ClassFileAnalyzer (Can)
4   ; Analyzer and Disassembler for Java class files
5   ; (Jasmin syntax 2, http://jasmin.sourceforge.net)
6   ;
7   ; ClassFileAnalyzer, version 0.7.0
8
9
10  .bytecode 52.0
11  .source Nano12.java
12  .class public Nano12
13  .super java/lang/Object
14
15  .method public <init>()V
16    .limit stack 1
17    .limit locals 1
18    .line 1
19    0: aload_0
20    1: invokespecial java/lang/Object/<init>()V
21    4: return
```

```
22 .end method
23
24 .method public static main([Ljava/lang/String;)V
25   .limit stack 2
26   .limit locals 4
27   .line 4
28   0: iconst_1
29   1: istore_1
30   .line 5
31   2: iconst_2
32   3: istore_2
33   .line 6
34   4: iconst_5
35   5: istore_3
36 Label6:
37   .line 7
38   6: iload_3
39   7: iload_1
40   8: if_icmple Label40
41   .line 8
42   11: iload_1
43   12: iload_2
44   13: if_icmpne Label26
45   .line 9
46   16: getstatic java/lang/System/out Ljava/io/PrintStream;
47   19: iload_1
48   20: invokevirtual java/io/PrintStream/println(I)V
49   23: goto Label33
50 Label26:
51   .line 11
52   26: getstatic java/lang/System/out Ljava/io/PrintStream;
53   29: iconst_0
54   30: invokevirtual java/io/PrintStream/println(I)V
55 Label33:
56   .line 12
57   33: iload_3
58   34: iconst_1
59   35: isub
60   36: istore_3
61   37: goto Label6
62 Label40:
63   .line 14
64   40: return
65   ; append_frame (frameNumber = 0)
66   ; frame_type = 254, offset_delta = 6
67   ; frame bytes: 254 0 6 1 1 1
68   .stack
```

```
69      offset 6
70      locals Integer
71      locals Integer
72      locals Integer
73      .end stack
74    ; same_frame (frameNumber = 1)
75    ; frame_type = 19, offset_delta = 19
76    ; frame bytes: 19
77    .stack
78      offset 26
79      locals Integer
80      locals Integer
81      locals Integer
82      .end stack
83    ; same_frame (frameNumber = 2)
84    ; frame_type = 6, offset_delta = 6
85    ; frame bytes: 6
86    .stack
87      offset 33
88      locals Integer
89      locals Integer
90      locals Integer
91      .end stack
92    ; same_frame (frameNumber = 3)
93    ; frame_type = 6, offset_delta = 6
94    ; frame bytes: 6
95    .stack
96      offset 40
97      locals Integer
98      locals Integer
99      locals Integer
100     .end stack
101 .end method
```

Aqui temos um código muito parecido com o Nano11 [0.3.11], podemos ver uma if ser realizado dentro do while e ao fim desse if é realizado uma subtração, onde carregamos o valor da variável 3 e a constante 1 para o topo da pilha, depois executamos isub para realizar a subtração dos dois valores do topo da pilha armazenando o resultado no topo da pilha e por fim usamos istore para armazenar o valor do topo da pilha na variável 3.

0.4 Principais Instruções do Jasmin

Abaixo estão listadas as principais instruções da sintaxe do Jasmin

0.4.1 .bytecode

Indica a versão de bytecode, não é necessário utiliza-la

0.4.2 .source

Indica o nome do arquivo source original.

0.4.3 .class

Utilizado para declarar uma classe

0.4.4 .super

Utilizado para declarar uma herança.

0.4.5 .method

Utilizado para declarar um método.

0.4.6 .limit

Utilizado para definir o tamanho da pilha ou o tamanho da variavel local

0.4.7 .end

Indica o final (p.e.: uma Classe ou um método)

0.4.8 .line

Utilizado para marcar a linha do código fonte, não é necessário, mas ajuda no entendimento do assembly.

0.4.9 lload

Carrega o valor de uma variável local para o topo da pilha.

0.4.10 lstore

Carrega o valor do topo da pilha para a variável determinada pelo índice passado como parâmetro.

0.4.11 iaload

Carrega um valor int de um vetor. O valor da referência do vetor e o índice são retirados do topo da pilha. O valor recuperado do vetor é colocado topo da pilha.

0.4.12 iload

Carrega o valor de uma variável local que é um inteiro para o topo da pilha.

0.4.13 istore

Carrega o valor do topo da pilha para a variável.

0.4.14 fload

Carrega o valor de uma variável local que é um float para o topo da pilha.

0.4.15 fstore

Carrega o valor do topo da pilha que é um float para a variável.

0.4.16 ldc

Desempilha a constante da pilha.

0.4.17 dup

A palavra no topo da pilha é duplicado.

0.4.18 pop

Retira o valor que esta no topo da pilha.

0.4.19 swap

Troca dois operandos da pilha (uma troca, o primeiro vira o segundo e o segundo vira o primeiro).

0.4.20 iadd

Adiciona dois inteiros.

0.4.21 idiv

Divide dois inteiros.

0.4.22 imul

Multiplica dois inteiros.

0.4.23 isub

Subtrai dois inteiros.

0.4.24 fadd

Adiciona dois float.

0.4.25 fdiv

Divide dois float.

0.4.26 fmul

Multiplica dois float.

0.4.27 fsub

Subtrai dois float.

0.4.28 goto

Ir para o marcador.

0.4.29 ifeq

Ir para o rótulo (marcador), se o valor no topo da pilha é 0.

0.4.30 ifge

Ir para o rótulo, se o valor no topo da pilha é igual ou superior (maior) a 0.

0.4.31 ifgt

Ir para o rótulo, se o valor no topo da pilha é superior (maior) a 0.

0.4.32 ifle

Ir para o rótulo, se o valor no topo da pilha é inferior (menor) ou igual a 0.

0.4.33 iflt

Ir para o rótulo, se o valor no topo da pilha é inferior (menor) a 0.

0.4.34 ifne

Ir para o rótulo, se o valor no topo da pilha não é igual a 0.

0.4.35 if_icmple

Ir para o rótulo, se o primeiro valor da pilha for menor ou igual ao segundo valor da pilha.

0.4.36 if_icmpne

Ir para o rótulo, se o primeiro valor da pilha for diferente do segundo valor da pilha.

0.4.37 if_icmpe

Ir para o rótulo, se o primeiro valor da pilha for diferente do segundo valor da pilha.

0.4.38 iand

AND (inteiros).

0.4.39 ior

OR (inteiros).

0.4.40 i2f

Converte inteiro para float.

0.4.41 f2i

Converte float para inteiro.

0.4.42 jsr

Retorna o endereço da pilha e ”pula”para subrotina indicada.

0.4.43 ret

Retorna o endereço da subrotina que esta armazenado a variável local.

0.4.44 invokevirtual

É a forma padrão para a chamada de um método.

0.4.45 invokeinterfacel

Realiza a chamada de um método de uma interface procurando o método implementado por um objeto particular de um ambiente de execução.

0.4.46 invokespecial

Realiza a chamada de métodos especiais como métodos privados, ou métodos da super classe.

0.4.47 invostatic

Faz a chamada de métodos de classe (static).

0.4.48 getstatic

Carrega um campo estático para o topo da pilha.

0.5 Analisador Léxico

O analisador léxico é quem realiza o primeiro processamento sobre o código, ele remove todos os espaços em branco e os comentários, além de produzir uma sequência de palavras chaves através do código, que é denominado de token.

0.5.1 Implementação

Para a implementação do analisador léxico para o portugol, foi utilizado uma adaptação dos códigos gerados em aula.

O código abaixou é o código final do analisador léxico

```
1 {
2   open Lexing
3   open Printf
4   open Sintatico
5
6   exception Erro of string
```

```
7
8   let incr_num_linha lexbuf =
9     let pos = lexbuf.lex_curr_p in
10     lexbuf.lex_curr_p <- { pos with
11        pos_lnum = pos.pos_lnum + 1;
12        pos_bol = pos.pos_cnum;
13     }
14
15  let pos_atual lexbuf = lexbuf.lex_start_p
16
17  let msg_erro lexbuf c =
18    let pos = lexbuf.lex_curr_p in
19    let lin = pos.pos_lnum
20    and col = pos.pos_cnum - pos.pos_bol - 1 in
21    sprintf "%d-%d: caracter desconhecido %c" lin col c
22
23  let erro lin col msg =
24    let mensagem = sprintf "%d-%d: %s" lin col msg in
25       failwith mensagem
26 }
27
28 let digito = ['0' - '9']
29 let inteiro = digito+
30 let real = digito+ '.' digito+
31
32 let letra = ['a' - 'z' 'A' - 'Z']
33 let identificador = ('_' | letra) ( letra | digito | '_')*
34
35 let brancos = [' ' '\t']+
36 let novalinha = '\r' | '\n' | "\r\n"
37
38 let comentario = "//" [^ '\r' '\n' ]*
39
40 rule read = parse
41   brancos    { read lexbuf }
42 | novalinha  { incr_num_linha lexbuf; read lexbuf }
43 | comentario { read lexbuf }
44 | "/*"       { comentario_bloco 0 lexbuf }
45 | '('        { APAR (pos_atual lexbuf) }
46 | '+'        { MAIS (pos_atual lexbuf) }
47 | '-'        { MENOS (pos_atual lexbuf) }
48 | '/'        { DIV (pos_atual lexbuf) }
49 | '*'        { MULTI (pos_atual lexbuf) }
50 | '%'        { RESTO (pos_atual lexbuf) }
51 | '^'        { POTEN (pos_atual lexbuf) }
52 | ':'        { DEF (pos_atual lexbuf) }
53 | ')'        { FPAR (pos_atual lexbuf) }
```

```
54  | "<-"        { ATRIB (pos_atual lexbuf) }
55  | "var"       { VAR (pos_atual lexbuf) }
56  | "="         { EQUALS (pos_atual lexbuf) }
57  | "<>"        { DIFER (pos_atual lexbuf) }
58  | ">="        { GTE (pos_atual lexbuf) }
59  | "<="        { LTE (pos_atual lexbuf) }
60  | '>'         { GT (pos_atual lexbuf) }
61  | '<'         { LT (pos_atual lexbuf) }
62  | ','         { VIRG (pos_atual lexbuf) }
63  | real as num    { let numero = float_of_string num in
64                        LITREAL (numero, pos_atual lexbuf) }
65  | inteiro as num { let numero = int_of_string num in
66                        LITINT (numero, pos_atual lexbuf)  }
67  | "e"          { AND (pos_atual lexbuf) }
68  | "E"          { AND (pos_atual lexbuf) }
69  | "ou"         { OR (pos_atual lexbuf) }
70  | "OU"         { OR (pos_atual lexbuf) }
71  | "se"         { IF (pos_atual lexbuf) }
72  | "ent o"      { ENTAO (pos_atual lexbuf) }
73  | "sen o"      { ELSE (pos_atual lexbuf) }
74  | "fim_se"     { FIF (pos_atual lexbuf) }
75  | "in cio"     { INI (pos_atual lexbuf) }
76  | "fa a"       { DO (pos_atual lexbuf) }
77  | "fim_para"   { FFOR (pos_atual lexbuf) }
78  | "para"       { FOR (pos_atual lexbuf) }
79  | "de"         { BEGIN (pos_atual lexbuf) }
80  | "at "        { END (pos_atual lexbuf) }
81  | "fim_enquanto" { FWHILE (pos_atual lexbuf) }
82  | "enquanto"   { WHILE (pos_atual lexbuf) }
83  | "escolha"    { SWITCH (pos_atual lexbuf) }
84  | "fim_escolha" { FSWITCH (pos_atual lexbuf) }
85  | "caso"       { CASE (pos_atual lexbuf) }
86  | "outrocaso"  { CASE_DEFAULT (pos_atual lexbuf) }
87  | "algoritmo"  { PROGRAMA (pos_atual lexbuf) }
88  | "fim_algoritmo" { FPROGRAMA (pos_atual lexbuf) }
89  | "escreva"    { SAIDA (pos_atual lexbuf) }
90  | "escreval"   { SAIDA_LINHA (pos_atual lexbuf) }
91  | "leia"       { ENTRADA (pos_atual lexbuf) }
92  | "fim_fun  o"     { FFUNCTION (pos_atual lexbuf) }
93  | "retorne"    { RETURN (pos_atual lexbuf) }
94  | "fun  o"     { FUNCTION (pos_atual lexbuf) }
95  | "inteiro"    { INTEGER (pos_atual lexbuf) }
96  | "real"       { REAL (pos_atual lexbuf) }
97  | "caractere"  { CARACTER (pos_atual lexbuf) }
98  | "l gico"     { BOOL (pos_atual lexbuf) }
99  | "verdadeiro" { LITBOOL(true, pos_atual lexbuf ) }
100 | "falso"      { LITBOOL(false, pos_atual lexbuf ) }
```

```
101 | identificador as id { ID (id, pos_atual lexbuf) }
102 | "'"_"'" as s  { let c = String.get s 1 in LITCHAR (c, pos_atual lexbuf )
      }
103 | '"'         { let pos = lexbuf.lex_curr_p in
104                 let lin = pos.pos_lnum
105                 and col = pos.pos_cnum - pos.pos_bol - 1 in
106                 let buffer = Buffer.create 1 in
107                 let str = leia_string lin col buffer lexbuf in
108                   LITSTRING (str, pos_atual lexbuf) }
109 | _ as c  { failwith (msg_erro lexbuf c) }
110 | eof         { EOF }
111
112 and comentario_bloco n = parse
113    "*/"       { if n=0 then read lexbuf
114                 else comentario_bloco (n-1) lexbuf }
115 | "/*"        { comentario_bloco (n+1) lexbuf }
116 | novalinha   { incr_num_linha lexbuf; comentario_bloco n lexbuf }
117 | _           { comentario_bloco n lexbuf }
118 | eof         { let pos = lexbuf.lex_curr_p in
119                 let lin = pos.pos_lnum
120                 and col = pos.pos_cnum - pos.pos_bol in
121                  erro lin col  "Comentario nao fechado" }
122
123 and leia_string lin col buffer = parse
124    '"'      { Buffer.contents buffer}
125 | "\\t"     { Buffer.add_char buffer '\t'; leia_string lin col buffer lexbuf
      }
126 | "\\n"     { Buffer.add_char buffer '\n'; leia_string lin col buffer lexbuf
      }
127 | '\\' '"'  { Buffer.add_char buffer '"'; leia_string lin col buffer lexbuf
      }
128 | '\\' '\\' { Buffer.add_char buffer '\\'; leia_string lin col buffer
    lexbuf }
129 | _ as c     { Buffer.add_char buffer c; leia_string lin col buffer lexbuf }
130 | eof       { erro lin col "A string nao foi fechada"}
```

0.5.2 Execução

Para execução utilize o seguinte código do carregador.ml

```
1     #load "lexico.cmo";;
2
3  let rec tokens lexbuf =
4    let tok = Lexico.token lexbuf in
5    match tok with
6    | Lexico.EOF -> [Lexico.EOF]
7    | _ -> tok :: tokens lexbuf
```

```
8  ;;
9
10 let lexico str =
11   let lexbuf = Lexing.from_string str in
12   tokens lexbuf
13 ;;
14
15 let lex arq =
16   let ic = open_in arq in
17   let lexbuf = Lexing.from_channel ic in
18   let toks = tokens lexbuf in
19   let _ = close_in ic in
20   toks
```

1. Execute o parse com ocamllex

```
1          ocamllex lexico.mll
```

2. Compile o arquivo .ml gerado pelo parse

```
1          ocamlc -c lexico.ml
```

3. Entre no interpretador do ocaml

```
1          rlwrap ocaml
```

4. No interpretador carregue o carregador.ml

```
1          #use "carregador.ml";;
```

5. Execute para o seu arquivo passando o caminho para o mesmo

```
1          lex "../../exemplos/Portugol/nano/nano12.alg";;
```

6. Após a execução você terá uma saida como a abaixo, onde temos os tokens do nosso código de entrada

```
1           lex "../../exemplos/Portugol/nano/nano12.alg";;
2 - : Lexico.tokens list =
3 [Lexico.PROGRAMA; Lexico.LITSTRING "nano12"; Lexico.VAR; Lexico.ID "n
    ";
4  Lexico.VIRG; Lexico.ID "m"; Lexico.VIRG; Lexico.ID "x"; Lexico.DEF;
5  Lexico.INTEGER; Lexico.INI; Lexico.ID "n"; Lexico.ATRIB; Lexico.
     LITINT 1;
6  Lexico.ID "m"; Lexico.ATRIB; Lexico.LITINT 2; Lexico.ID "x"; Lexico.
     ATRIB;
7  Lexico.LITINT 5; Lexico.WHILE; Lexico.ID "x"; Lexico.GT; Lexico.ID "n
     ";
8  Lexico.DO; Lexico.IF; Lexico.ID "n"; Lexico.EQUALS; Lexico.ID "m";
```

```
9   Lexico.OR; Lexico.ID "n"; Lexico.GTE; Lexico.LITINT 0; Lexico.ENTAO;
10  Lexico.ID "escreva"; Lexico.APAR; Lexico.ID "n"; Lexico.FPAR; Lexico.
        ELSE;
11  Lexico.ID "escreva"; Lexico.APAR; Lexico.LITINT 0; Lexico.FPAR;
        Lexico.FIF;
12  Lexico.ID "x"; Lexico.ATRIB; Lexico.ID "x"; Lexico.MENOS; Lexico.
        LITINT 1;
13  Lexico.ID "fim_enquanto"; Lexico.FPROGRAMA; Lexico.EOF]
```

0.6 Analisador Sintático

É a analise realizada pelo compilador para identificar se o código pertence a linguagem.

0.6.1 Preparando o Ambiente

Antes de iniciarmos lembre-se de utilizar o lexico.mll atualizado que se encontrar na seção acima. Precisamos agora instalar o menhir que é quem irá gerar o analisador sintático LR(1) para a gramática que iremos criar. Para isso utilize

```
1     opam install menhir
```

Agora com o menhir instalado crie um arquivo .ocamlinit com o seguinte conteúdo

```
1  let () =
2    try Topdirs.dir_directory (Sys.getenv "OCAML_TOPLEVEL_PATH")
3    with Not_found -> ()
4  ;;
5
6  #use "topfind";;
7  #require "menhirLib";;
8  #directory "_build";;
9  #load "erroSint.cmo";;
10 #load "sintatico.cmo";;
11 #load "lexico.cmo";;
12 #load "ast.cmo";;
13 #load "sast.cmo";;
14 #load "tast.cmo";;
15 #load "tabsimb.cmo";;
16 #load "ambiente.cmo";;
17 #load "semantico.cmo";;
18 #load "semanticoTest.cmo";;
19
20 open Ast
21 open Ambiente
22 open SemanticoTest
```

0.6.2 Árvore Sintática Abstrata

Crie o arquivo ast.ml

```
1  (* The type of the abstract syntax tree (AST). *)
2  open Lexing
3
4  type ident = string
5  type 'a pos =  'a * Lexing.position (* tipo e posi  o no arquivo fonte *)
6
7  type 'expr programa = Programa of (ident pos * tipo) list * ('expr funcoes)
       * ('expr comandos)
8  and declaracoes = declaracao list
9  and 'expr funcoes = ('expr funcao) list
10 and 'expr comandos = ('expr comando) list
11 and declaracao = DecVar of (ident pos) * tipo
12 and 'expr expressoes = 'expr list
13
14 and tipo = TipoInt
15          | TipoString
16          | TipoReal
17          | TipoBool
18          | TipoVoid
19
20 and 'expr comando = CmdAtrib of variavel * 'expr
21              | CmdSe of 'expr * ('expr comandos) * ('expr comandos option)
22              | CmdEnquanto of 'expr * ('expr comandos)
23              | CmdReturn of 'expr
24              | CmdEntrada of ('expr expressoes)
25              | CmdChamada of 'expr
26              | CmdSaida of ('expr expressoes)
27              | CmdSaidaLine of ('expr expressoes)
28              | CmdFor of variavel * 'expr * 'expr * ('expr comandos)
29              | CmdSwitch of variavel * ('expr cases) * ('expr caseDefault
                    option)
30
31 and 'expr cases = ('expr case) list
32 and 'expr case = CmdCase of 'expr * ('expr comandos)
33 and 'expr caseDefault = CmdCaseDefault of ('expr comandos)
34
35 and variaveis = variavel list
36 and variavel = VarSimples of ident pos
37
38 and 'expr funcao = Funcao of ('expr decFuncao)
39
40 and 'expr decFuncao = {
41   fn_nome:         ident pos;
42   fn_nome_fecha:   ident pos;
```

```
43    fn_tiporet:     tipo;
44    fn_formais:     (ident pos * tipo) list;
45    fn_locais:      (ident pos * tipo) list;
46    fn_corpo:       'expr comandos
47  }
48
49  and oper = Mais
50           | Menos
51           | Mult
52           | Div
53           | Resto
54           | Potencia
55           | Menor
56           | Igual
57           | Difer
58           | Maior
59           | MaioIgual
60           | MenorIgual
61           | E
62           | Ou
63           | Concat
```

Cria mais um arquivo chamado sast.ml

```
1  open Ast
2
3  type expressao = ExpVar of variavel
4              | ExpInt of int pos
5              | ExpString of string pos
6              | ExpChar of char pos
7              | ExpBool of bool pos
8              | ExpReal of float pos
9              | ExpFuncao of ident pos * (expressao expressoes)
10             | ExpOp of oper pos * expressao * expressao
```

Cria mais um arquivo chamado tast.ml

```
1  open Ast
2
3  type expressao = ExpVar of variavel * tipo
4              | ExpInt of int * tipo
5              | ExpString of string * tipo
6              | ExpChar of char * tipo
7              | ExpBool of bool * tipo
8              | ExpReal of float * tipo
9              | ExpFuncao of ident * (expressao expressoes) * tipo
10             | ExpOp of (oper * tipo) * (expressao * tipo) * (expressao *
                 tipo)
```

0.6.3 Gramática

Crie o arquivo sintatico.mly com o seguinte conteúdo

```
%{
open Lexing
open Ast
open Sast
%}

%token <int * Lexing.position> LITINT
%token <string * Lexing.position> LITSTRING
%token <char * Lexing.position> LITCHAR
%token <string * Lexing.position> ID
%token <bool * Lexing.position> LITBOOL
%token <float * Lexing.position> LITREAL
%token <Lexing.position> FUNCTION FFUNCTION
%token <Lexing.position> PROGRAMA FPROGRAMA INI
%token <Lexing.position> MAIS MENOS MULTI DIV
%token <Lexing.position> INTEGER CARACTER REAL VAR BOOL
%token <Lexing.position> APAR FPAR
%token <Lexing.position> ATRIB
%token <Lexing.position> IF
%token <Lexing.position> ELSE
%token <Lexing.position> EQUALS DIFER GTE GT LTE LT POTEN RESTO
%token <Lexing.position> VIRG
%token <Lexing.position> AND OR
%token <Lexing.position> DEF
%token <Lexing.position> SAIDA ENTRADA SAIDA_LINHA
%token <Lexing.position> WHILE FWHILE FOR FFOR SWITCH FSWITCH CASE
    CASE_DEFAULT RETURN
%token <Lexing.position> DO
%token <Lexing.position> FIF
%token <Lexing.position> ENTAO BEGIN END
%token EOF

%left OR AND
%left EQUALS DIFER
%left LT LTE GT GTE
%right POTEN
%left MAIS MENOS RESTO
%left MULTI DIV


%start <Sast.expressao Ast.programa> prog

%%

```

```
44 prog:
45     | PROGRAMA LITSTRING
46         ds = declaracao
47       INI
48         cs = comando*
49       FPROGRAMA
50       dFunctions = decFunctions*
51       EOF {
52           Programa (List.flatten ds, dFunctions, cs) }
53     ;
54
55 declaracao:
56     | VAR
57       v=variaveis* { v }
58     | { [] }
59     ;
60
61 variaveis:
62     | ids=separated_nonempty_list(VIRG, ID) DEF t=tipo {
63           List.map (fun id -> (id,t)) ids
64     }
65
66 tipo:
67     | INTEGER   { TipoInt }
68     | REAL      { TipoReal }
69     | CARACTER  { TipoString }
70     | BOOL      { TipoBool }
71     ;
72
73 comando: c=comando_atribuicao { c }
74        | c=comando_se          { c }
75        | c=comando_entrada     { c }
76        | c=comando_saida       { c }
77        | c=comando_saida_l     { c }
78        | c=comando_enquanto    { c }
79        | c=comando_for         { c }
80        | c=comando_switch      { c }
81        | c=comando_return      { c }
82        | c=comando_chamada      { c }
83        ;
84
85 comando_return: RETURN e=expressao {
86       CmdReturn (e)
87 }
88
89 comando_chamada: exp=chamada { CmdChamada(exp) }
90
```

```
91  comando_atribuicao: v=variavel ATRIB e=expressao {
92        CmdAtrib (v,e)
93  }
94
95  comando_se: IF teste=expressao ENTAO
96                entao=comando+
97                senao=option(ELSE cs=comando+ {cs})
98              FIF {
99                CmdSe (teste, entao, senao)
100             }
101
102 comando_entrada: ENTRADA APAR xs=separated_nonempty_list(VIRG, expressao)
        FPAR {
103                   CmdEntrada xs
104               }
105
106 comando_saida: SAIDA APAR xs=separated_nonempty_list(VIRG, expressao) FPAR
        {
107                 CmdSaida xs
108             }
109
110 comando_saida_l: SAIDA_LINHA APAR xs=separated_nonempty_list(VIRG,
        expressao) FPAR {
111                 CmdSaidaLine xs
112             }
113
114 comando_enquanto: WHILE teste=expressao DO
115                       entao=comando+
116                   FWHILE {
117                     CmdEnquanto( teste, entao )
118                   }
119
120 comando_for:  FOR v=variavel BEGIN l=expressao END r=expressao DO
121               c=comando+
122             FFOR {
123               CmdFor(v, l, r, c)
124             }
125
126 comando_switch: SWITCH v=variavel
127                     c=caso+
128                     d=option(default)
129                 FSWITCH {
130                   CmdSwitch(v, c, d)
131                 }
132
133 caso: CASE e=expressao
134           c=comando+ {
```

```
135             CmdCase(e, c)
136           }
137
138 default: CASE_DEFAULT
139             c=comando+ {
140               CmdCaseDefault(c)
141             }
142
143 decFunctions: FUNCTION a=ID APAR args=variaveis* FPAR DEF t=tipo
144                   variables=declaracao
145                 INI
146                   corpo=comando*
147                 FFUNCTION b=ID {
148                       Funcao {
149                         fn_nome =         a;
150                         fn_nome_fecha =   b;
151                         fn_tiporet =      t;
152                         fn_formais =      List.flatten args;
153                         fn_locais =       List.flatten variables;
154                         fn_corpo =        corpo
155                       }
156                   }
157
158 chamada: i=ID APAR xs=separated_nonempty_list(VIRG, expressao) FPAR {
159             ExpFuncao(i, xs)
160           }
161
162 expressao:
163         | c=chamada      { c }
164         | v=variavel     { ExpVar v    }
165         | i=LITINT       { ExpInt i    }
166         | s=LITSTRING    { ExpString s }
167         | c=LITCHAR      { ExpChar c }
168         | b=LITBOOL      { ExpBool b }
169         | r=LITREAL      { ExpReal r }
170         | e1=expressao op=oper e2=expressao { ExpOp (op, e1, e2) }
171         | APAR e=expressao FPAR { e }
172
173 %inline oper:
174  | pos = MAIS  { (Mais, pos)  }
175   | pos = MENOS { (Menos, pos) }
176   | pos = MULTI  { (Mult, pos)  }
177   | pos = DIV   { (Div, pos)   }
178   | pos = RESTO { (Resto, pos) }
179   | pos = POTEN { (Potencia, pos) }
180   | pos = LT { (Menor, pos) }
181   | pos = GT { (Maior, pos) }
```

```
182     | pos = LTE { (MenorIgual, pos) }
183     | pos = GTE { (MaioIgual, pos) }
184     | pos = EQUALS { (Igual, pos) }
185     | pos = DIFER { (Difer, pos) }
186     | pos = AND  { (E, pos)      }
187     | pos = OR { (Ou, pos)     }
188
189
190 variavel:
191         | x=ID        { VarSimples x }
```

0.6.4 Execução

Para execução e detecção de erros cria o arquivo sintaticoTest.ml com o seguinte conteúdo.

```
1  open Printf
2  open Lexing
3
4  open Ast
5  exception Erro_Sintatico of string
6
7  module S = MenhirLib.General (* Streams *)
8  module I = Sintatico.MenhirInterpreter
9
10 open ErroSint
11
12 let posicao lexbuf =
13     let pos = lexbuf.lex_curr_p in
14     let lin = pos.pos_lnum
15     and col = pos.pos_cnum - pos.pos_bol - 1 in
16     sprintf "linha %d, coluna %d" lin col
17
18 (* [pilha checkpoint] extrai a pilha do aut mato LR(1) contida em
      checkpoint *)
19
20 let pilha checkpoint =
21   match checkpoint with
22   | I.HandlingError amb -> I.stack amb
23   | _ -> assert false (* Isso n o pode acontecer *)
24
25 let estado checkpoint : int =
26   match Lazy.force (pilha checkpoint) with
27   | S.Nil -> (* O parser est  no estado inicial *)
28      0
29   | S.Cons (I.Element (s, _, _, _), _) ->
30      I.number s
31
```

```
32 let sucesso v = v
33
34 let falha lexbuf (checkpoint : Ast.programa I.checkpoint) =
35   let estado_atual = estado checkpoint in
36   let msg = message estado_atual in
37   raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
38                                         (Lexing.lexeme_start lexbuf) msg))
39
40 let loop lexbuf resultado =
41   let fornecedor = I.lexer_lexbuf_to_supplier Lexico.read lexbuf in
42   I.loop_handle sucesso (falha lexbuf) fornecedor resultado
43
44
45
46 let parse_com_erro lexbuf =
47   try
48     Some (loop lexbuf (Sintatico.Incremental.prog lexbuf.lex_curr_p))
49   with
50   | Lexico.Erro msg ->
51      printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
52      None
53   | Erro_Sintatico msg ->
54      printf "Erro sint tico na %s %s\n" (posicao lexbuf) msg;
55      None
56
57 let parse s =
58   let lexbuf = Lexing.from_string s in
59   let ast = parse_com_erro lexbuf in
60   ast
61
62 let parse_arq nome =
63   let ic = open_in nome in
64   let lexbuf = Lexing.from_channel ic in
65   let ast = parse_com_erro lexbuf in
66   let _ = close_in ic in
67   ast
68
69 let rec tokens lexbuf =
70   let tok = Lexico.read lexbuf in
71   match tok with
72   | Sintatico.EOF -> [Sintatico.EOF]
73   | _ -> tok :: tokens lexbuf
74
75 let lexicoArq arq =
76   let ic = open_in arq in
77   let lexbuf = Lexing.from_channel ic in
78   let toks = tokens lexbuf in
```

```
79    let _ = close_in ic in
80    toks
```

0.6.5 Compilando

Para compilarmos o compilador, utilizei os comandos fornecidos pelo professor, são eles:

```
1  menhir -v --list-errors sintatico.mly > sintatico.msg
2
3  menhir sintatico.mly --compile-errors sintatico.msg >
4  erroSint.ml
5
6  ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --
7  table" -package menhirLib sintaticoTest.byte
```

Após isto utilize rlwrap ocaml na pasta do projeto onde se encontra o .ocamlinit para ter acesso ou código do compilador. No ocaml utilize parse_arq 'caminho para um arquivo' para gerar a árvore sintática

0.7 Analisador Semântico

Agora faremos o analisador semântico, responsável pela verificação de tipos da linguagem.

Criaremos os arquivos necessários para implementar o ambiente.

Crie um arquivo com o nome tabsimb.ml

```
1
2  type 'a tabela = {
3      tbl: (string, 'a) Hashtbl.t;
4      pai: 'a tabela option;
5  }
6
7  exception Entrada_existente of string;;
8
9  let insere amb ch v =
10   if Hashtbl.mem amb.tbl ch
11   then raise (Entrada_existente ch)
12   else Hashtbl.add amb.tbl ch v
13
14 let substitui amb ch v = Hashtbl.replace amb.tbl ch v
15
16 let rec atualiza amb ch v =
17     if Hashtbl.mem amb.tbl ch
18     then Hashtbl.replace amb.tbl ch v
19     else match amb.pai with
20        None -> failwith "tabsim atualiza: chave nao encontrada"
21      | Some a -> atualiza a ch v
```

```
22
23 let rec busca amb ch =
24   try Hashtbl.find amb.tbl ch
25   with Not_found ->
26     (match amb.pai with
27        None -> raise Not_found
28      | Some a -> busca a ch)
29
30 let rec cria cvs =
31   let amb = {
32     tbl = Hashtbl.create 5;
33     pai = None
34   } in
35   let _ = List.iter (fun (c,v) -> insere amb c v) cvs
36   in amb
37
38 let novo_escopo amb_pai = {
39   tbl = Hashtbl.create 5;
40   pai = Some amb_pai
41 }
```

E crie o tabsimb.mli

```
1
2 type 'a tabela
3
4 exception Entrada_existente of string
5
6 val insere : 'a tabela -> string -> 'a -> unit
7 val substitui : 'a tabela -> string -> 'a -> unit
8 val atualiza : 'a tabela -> string -> 'a -> unit
9 val busca  : 'a tabela -> string -> 'a
10 val cria   : (string * 'a) list -> 'a tabela
11
12 val novo_escopo : 'a tabela -> 'a tabela
```

Criando o ambiente agora. Crie ambiente.ml

```
1 module Tab = Tabsimb
2 module A = Ast
3
4 type entrada_fn = { tipo_fn:  A.tipo;
5                     formais: (string * A.tipo) list;
6 }
7
8 type entrada =  EntFun of entrada_fn
9              |  EntVar of A.tipo
10
11 type t = {
12   ambv : entrada Tab.tabela
```

```
}

let novo_amb xs = { ambv = Tab.cria xs }

let novo_escopo amb = { ambv = Tab.novo_escopo amb.ambv }

let busca amb ch = Tab.busca amb.ambv ch

let insere_local amb ch t =
  Tab.insere amb.ambv ch (EntVar t)

let insere_param amb ch t =
  Tab.insere amb.ambv ch (EntVar t)

let insere_fun amb nome params resultado =
  let ef = EntFun { tipo_fn = resultado;
                    formais = params }
  in Tab.insere amb.ambv nome ef
```

Agora o ambiente.mli

```
type entrada_fn = { tipo_fn:  Ast.tipo;
                    formais: (string * Ast.tipo) list;
          (*      locais: (string * Asabs.tipo) list *)
}

type entrada =  EntFun of entrada_fn
             |  EntVar of Ast.tipo

type t

val novo_amb :  (string * entrada) list -> t
val novo_escopo : t -> t
val busca: t -> string -> entrada
val insere_local : t -> string -> Ast.tipo -> unit
val insere_param : t -> string -> Ast.tipo -> unit
val insere_fun : t -> string -> (string * Ast.tipo) list -> Ast.tipo ->
    unit
```

Agora implementaremos o analisador semântico.

Crie o semantico.ml

```
module Amb = Ambiente
module A = Ast
module S = Sast
module T = Tast

let rec posicao_var var = let open A in
  match var with
```

```
8    | VarSimples (_, pos) -> pos
9
10
11  let rec posicao exp = let open S in
12    match exp with
13    | ExpVar v -> (match v with
14        | A.VarSimples (_,pos) -> pos
15      )
16    | ExpInt (_,pos) -> pos
17    | ExpString  (_,pos) -> pos
18    | ExpChar (_, pos) -> pos
19    (* | ExpNegativo (_, (_,pos)) -> pos *)
20    | ExpBool (_,pos) -> pos
21    | ExpReal (_,pos) -> pos
22    | ExpOp ((_,pos),_,_)  -> pos
23    | ExpFuncao ((_,pos), _) -> pos
24
25  type classe_op = Aritmetico | Relacional | Logico | Cadeia
26
27  let classifica op =
28    let open A in
29    match op with
30      Mais
31    | Menos
32    | Mult
33    | Div
34    | Resto
35    | Potencia -> Aritmetico
36    | Menor
37    | Igual
38    | Difer
39    | Maior
40    | MaioIgual
41    | MenorIgual -> Relacional
42    | E
43    | Ou  ->  Logico
44    | Concat  -> Cadeia
45
46  let msg_erro_pos pos msg =
47    let open Lexing in
48    let lin = pos.pos_lnum
49    and col = pos.pos_cnum - pos.pos_bol - 1 in
50    Printf.sprintf "Semantico -> linha %d, coluna %d: %s" lin col msg
51
52  let msg_erro nome msg =
53    let pos = snd nome in
54    msg_erro_pos pos msg
```

```
55
56 let nome_tipo t =
57   let open A in
58     match t with
59       TipoInt -> "inteiro"
60     | TipoString -> "string"
61     | TipoBool -> "bool"
62     | TipoVoid -> "void"
63     | TipoReal -> "real"
64
65 let mesmo_tipo pos msg tinf tdec =
66   if tinf <> tdec
67   then
68     let msg = Printf.sprintf msg (nome_tipo tinf) (nome_tipo tdec) in
69     failwith (msg_erro_pos pos msg)
70
71 let busca_por_id nome amb id =
72         (try (match (Amb.busca amb id) with
73               | Amb.EntVar tipo -> (T.ExpVar (A.VarSimples nome, tipo),
                    tipo)
74               | Amb.EntFun _ ->
75                 let msg = "nome de funcao usado como nome de variavel: " ^
                        id in
76                  failwith (msg_erro nome msg)
77            )
78          with Not_found ->
79                 let msg = "A variavel " ^ id ^ " nao foi declarada" in
80                 failwith (msg_erro nome msg)
81         )
82
83 let busca_variavel_por_id nome amb id =
84         (try (match (Amb.busca amb id) with
85               | Amb.EntVar tipo -> (A.VarSimples nome, tipo)
86               | Amb.EntFun _ ->
87                 let msg = "nome de funcao usado como nome de variavel: " ^
                        id in
88                  failwith (msg_erro nome msg)
89            )
90          with Not_found ->
91                 let msg = "A variavel " ^ id ^ " nao foi declarada" in
92                 failwith (msg_erro nome msg)
93         )
94
95 let rec infere_exp amb exp =
96   match exp with
97     S.ExpInt n    -> (T.ExpInt (fst n, A.TipoInt),       A.TipoInt)
98   | S.ExpString s -> (T.ExpString (fst s, A.TipoString), A.TipoString)
```

```
99  | S.ExpChar s -> (T.ExpChar (fst s, A.TipoString), A.TipoString)
100 | S.ExpBool b  -> (T.ExpBool (fst b, A.TipoBool),    A.TipoBool)
101 | S.ExpReal b  -> (T.ExpReal (fst b, A.TipoReal),    A.TipoReal)
102 | S.ExpVar v ->
103   (match v with
104      A.VarSimples nome ->
105      (* Tenta encontrar a defini  o da vari vel no escopo local, se
            n o      *)
106      (* encontar tenta novamente no escopo que engloba o atual. Prossegue
            -se *)
107      (* assim at  encontrar a defini  o em algum escopo englobante ou
            at     *)
108      (* encontrar o escopo global. Se em algum lugar for encontrado,
                 *)
109      (* devolve-se a defini  o. Em caso contr rio, devolve uma
            exce  o      *)
110      let id = fst nome in busca_por_id nome amb id
111   )
112 | S.ExpOp (op, esq, dir) ->
113   let (esq, tesq) = infere_exp amb esq
114   and (dir, tdir) = infere_exp amb dir in
115
116   let verifica_aritmetico () =
117     (match tesq with
118        A.TipoInt
119      | A.TipoReal ->
120        let _ = mesmo_tipo (snd op)
121                    "O operando esquerdo eh do tipo %s mas o direito eh
                           do tipo %s"
122                    tesq tdir
123        in tesq (* O tipo da express o aritm tica como um todo *)
124      | t -> let msg = "um operador aritmetico nao pode ser usado com o
               tipo " ^
125                       (nome_tipo t)
126        in failwith (msg_erro_pos (snd op) msg)
127     )
128
129   and verifica_relacional () =
130     (match tesq with
131        A.TipoInt
132      | A.TipoReal
133      | A.TipoString ->
134        let _ = mesmo_tipo (snd op)
135                  "O operando esquerdo eh do tipo %s mas o direito eh do
                         tipo %s"
136                  tesq tdir
137        in A.TipoBool (* O tipo da express o relacional    sempre
```

```
            booleano *)
138
139         | t -> let msg = "um operador relacional nao pode ser usado com o
              tipo " ^
140                           (nome_tipo t)
141           in failwith (msg_erro_pos (snd op) msg)
142       )
143
144     and verifica_logico () =
145       (match tesq with
146          A.TipoBool ->
147          let _ = mesmo_tipo (snd op)
148                    "O operando esquerdo eh do tipo %s mas o direito eh do
                        tipo %s"
149                    tesq tdir
150          in A.TipoBool (* O tipo da express o l gica    sempre booleano
               *)
151
152         | t -> let msg = "um operador logico nao pode ser usado com o tipo "
                ^
153                           (nome_tipo t)
154              in failwith (msg_erro_pos (snd op) msg)
155       )
156     and verifica_cadeia () =
157       (match tesq with
158          A.TipoString ->
159          let _ = mesmo_tipo (snd op)
160                    "O operando esquerdo eh do tipo %s mas o direito eh do
                        tipo %s"
161                    tesq tdir
162          in A.TipoString (* O tipo da express o relacional    sempre
               string *)
163
164         | t -> let msg = "um operador relacional nao pode ser usado com o
              tipo " ^
165                           (nome_tipo t)
166              in failwith (msg_erro_pos (snd op) msg)
167       )
168
169     in
170     let op = fst op in
171     let tinf = (match (classifica op) with
172           Aritmetico -> verifica_aritmetico ()
173         | Relacional -> verifica_relacional ()
174         | Logico -> verifica_logico ()
175         | Cadeia -> verifica_cadeia ()
176       )
```

```
177     in
178       (T.ExpOp ((op,tinf), (esq, tesq), (dir, tdir)), tinf)
179
180   | S.ExpFuncao (nome, args) ->
181     let rec verifica_parametros ags ps fs =
182        match (ags, ps, fs) with
183         (a::ags), (p::ps), (f::fs) ->
184           let _ = mesmo_tipo (posicao a)
185                   "O parametro eh do tipo %s mas deveria ser do tipo %s"
                          p f
186           in verifica_parametros ags ps fs
187       | [], [], [] -> ()
188       | _ -> failwith (msg_erro nome "Numero incorreto de parametros")
189     in
190     let id = fst nome in
191     try
192       begin
193         let open Amb in
194
195         match (Amb.busca amb id) with
196         (* verifica se 'nome' est  associada a uma fun   o *)
197           Amb.EntFun {tipo_fn; formais} ->
198           (* Infere o tipo de cada um dos argumentos *)
199           let argst = List.map (infere_exp amb) args
200           (* Obtem o tipo de cada par metro formal *)
201           and tipos_formais = List.map snd formais in
202           (* Verifica se o tipo de cada argumento confere com o tipo
                 declarado *)
203           (* do par metro formal correspondente.
                                                *)
204           let _ = verifica_parametros args (List.map snd argst)
                 tipos_formais
205            in (T.ExpFuncao (id, (List.map fst argst), tipo_fn), tipo_fn)
206         | Amb.EntVar _ -> (* Se estiver associada a uma vari vel, falhe
               *)
207           let msg = id ^ " eh uma variavel e nao uma funcao" in
208           failwith (msg_erro nome msg)
209       end
210     with Not_found ->
211       let msg = "Nao existe a funcao de nome " ^ id in
212       failwith (msg_erro nome msg)
213
214 let rec verifica_cmd amb tiporet cmd =
215   let open A in
216   match cmd with
217     CmdReturn exp ->
218       let (e1,tinf) = infere_exp amb exp in
```

```
219             let _ = mesmo_tipo (posicao exp)
220                                 "O tipo retornado eh %s mas foi declarado
                                        como %s"
221                                 tinf tiporet
222             in CmdReturn (e1)
223     | CmdSe (teste, entao, senao) ->
224       let (teste1,tinf) = infere_exp amb teste in
225       (* O tipo inferido para a express o 'teste' do condicional deve ser
             booleano *)
226       let _ = mesmo_tipo (posicao teste)
227                "O teste do if deveria ser do tipo %s e nao %s"
228                TipoBool tinf in
229       (* Verifica a validade de cada comando do bloco 'ent o' *)
230       let entao1 = List.map (verifica_cmd amb tiporet) entao in
231       (* Verifica a validade de cada comando do bloco 'sen o', se houver *)
232       let senao1 =
233           match senao with
234             None -> None
235           | Some bloco -> Some (List.map (verifica_cmd amb tiporet) bloco)
236        in
237        CmdSe (teste1, entao1, senao1)
238
239     | CmdAtrib (elem, exp) ->
240       (* Infere o tipo da express o no lado direito da atribui  o *)
241       let (exp,  tdir) = infere_exp amb exp
242       (* Faz o mesmo para o lado esquerdo *)
243       and (elem1, tesq) =
244         (
245           match elem with
246               A.VarSimples name ->  let id = fst name
247                   in busca_variavel_por_id name amb id
248             (* | _ -> failwith "O lado esquerdo de uma atribui  o precisa
                   ser uma variavel!" *)
249         ) in
250
251       (* Os dois tipos devem ser iguais *)
252       let _ = mesmo_tipo (posicao_var elem)
253                          "Atribuicao com tipos diferentes: %s = %s" tesq tdir
254       in CmdAtrib (elem1, exp)
255
256     | CmdChamada exp ->
257        let (exp,tinf) = infere_exp amb exp in
258        CmdChamada exp
259
260     | CmdEntrada exps ->
261       (* Verifica o tipo de cada argumento da fun  o 'entrada' *)
262       let exps = List.map (infere_exp amb) exps in
```

```
263       CmdEntrada (List.map fst exps)
264
265     | CmdSaida exps ->
266       (* Verifica o tipo de cada argumento da fun   o 'saida' *)
267       let exps = List.map (infere_exp amb) exps in
268       CmdSaida (List.map fst exps)
269     | CmdSaidaLine exps ->
270       (* Verifica o tipo de cada argumento da fun   o 'saida' *)
271       let exps = List.map (infere_exp amb) exps in
272       CmdSaidaLine (List.map fst exps)
273     | CmdFor( variavel, inicio, fim, comandos ) ->
274         let (variavel1, tvar) =
275           (
276             match variavel with
277                 A.VarSimples name ->  let id = fst name
278                     in busca_variavel_por_id name amb id
279               (* | _ -> failwith "O lado esquerdo de uma atribui   o precisa
                      ser uma variavel!" *)
280           ) in
281           let (inicio1, tinicio) = infere_exp amb inicio
282             and (fim1, tfim) = infere_exp amb fim in
283
284           let _ = mesmo_tipo (posicao inicio)
285                "A expressao de inicio e fim devem se do mesmo tipo e sao: %s
                     e %s" tinicio tfim
286               and _ = mesmo_tipo (posicao fim)
287                "A variavel deve ser do mesmo tipo das expressoes inicio e
                     fim mas sao: varialvel=%s e expressoes=%s"
288               tvar tinicio in
289           (* Verifica a validade de cada comando do bloco *)
290           let comandos1 = List.map (verifica_cmd amb tiporet) comandos in
291           CmdFor (variavel1, inicio1, fim1, comandos1)
292     | CmdSwitch ( variavel, cases, default ) ->
293           let (variavel1, tvar) =
294           (
295             match variavel with
296                 A.VarSimples name ->  let id = fst name
297                     in busca_variavel_por_id name amb id
298               (* | _ -> failwith "O lado esquerdo de uma atribui   o precisa
                      ser uma variavel!" *)
299           ) in
300           let gera_case exp cmds =
301               let (exp1, texp) = infere_exp amb exp
302               and comandos1 = List.map (verifica_cmd amb tiporet) cmds in
303               let _ = mesmo_tipo (posicao exp)
304                       "A expressao do caso precisa ser igual ao tipo da
                            variavel do escolha %s %s"
```

```
305                     tvar texp in
306               CmdCase(exp1, comandos1) in
307         let verifica_case case =
308             (match case with
309                   A.CmdCase (exp,cmds) -> gera_case exp cmds
310             ) in
311         let verifica_case_default case =
312             (
313               match case with
314                 None -> None
315               | Some A.CmdCaseDefault (cmds) -> Some (CmdCaseDefault(List.
                    map (verifica_cmd amb tiporet) cmds))
316             )
317           in
318         let default1 = verifica_case_default default
319         in
320         let cases1 = List.map (verifica_case) cases in
321
322         CmdSwitch( variavel1, cases1, default1 )
323   | CmdEnquanto (teste, entao) ->
324     let (teste1,tinf) = infere_exp amb teste in
325     (* O tipo inferido para a express o 'teste' do condicional deve ser
          booleano *)
326     let _ = mesmo_tipo (posicao teste)
327               "O teste do enquanto deveria ser do tipo %s e nao %s"
328               TipoBool tinf in
329     (* Verifica a validade de cada comando do bloco 'ent o' *)
330     let entao1 = List.map (verifica_cmd amb tiporet) entao in
331      CmdEnquanto (teste1, entao1)
332   (* | _ -> failwith "Comando n o implementado ainda!" *)
333 and verifica_fun amb ast =
334   let open A in
335   match ast with
336     A.Funcao {fn_nome; fn_nome_fecha;fn_tiporet; fn_formais; fn_locais;
          fn_corpo} ->
337     (* Estende o ambiente global, adicionando um ambiente local *)
338     let ambfn = Amb.novo_escopo amb in
339     (* Insere os par metros no novo ambiente *)
340     let insere_parametro (v,t) = Amb.insere_param ambfn (fst v) t in
341     let _ = List.iter insere_parametro fn_formais in
342     (* Insere as vari veis locais no novo ambiente *)
343     let insere_local = function
344         (v,t) -> Amb.insere_local ambfn (fst v)  t in
345     let _ = List.iter insere_local fn_locais in
346     (* Verifica cada comando presente no corpfn_formaiso da fun   o usando
          o novo ambiente *)
347     let corpo_tipado = List.map (verifica_cmd ambfn fn_tiporet) fn_corpo in
```

```
348       A.Funcao {fn_nome; fn_nome_fecha; fn_tiporet; fn_formais; fn_locais;
              fn_corpo = corpo_tipado}
349
350
351 let rec verifica_dup xs =
352   match xs with
353     [] -> []
354   | (nome,t)::xs ->
355     let id = fst nome in
356     if (List.for_all (fun (n,t) -> (fst n) <> id) xs)
357     then (id, t) :: verifica_dup xs
358     else let msg = "Parametro duplicado " ^ id in
359       failwith (msg_erro nome msg)
360
361 let insere_declaracao_var amb dec =
362   let open A in
363     match dec with
364         (nome, tipo) -> Amb.insere_local amb (fst nome) tipo
365
366 let insere_declaracao_fun amb dec =
367   let open A in
368     match dec with
369     Funcao {fn_nome; fn_tiporet; fn_formais; fn_corpo} ->
370         (* Verifica se n o h  par metros duplicados *)
371         let formais = verifica_dup fn_formais in
372         let nome = fst fn_nome in
373         Amb.insere_fun amb nome formais fn_tiporet
374
375
376 (* Lista de cabe alhos das fun  es pr  definidas *)
377 let fn_predefs = let open A in [
378    ("entrada", [("x", TipoInt); ("y", TipoInt)], TipoVoid);
379    ("saida",   [("x", TipoInt); ("y", TipoInt)], TipoVoid)
380 ]
381
382 (* insere as fun  es pr  definidas no ambiente global *)
383 let declara_predefinidas amb =
384   List.iter (fun (n,ps,tr) -> Amb.insere_fun amb n ps tr) fn_predefs
385
386 let semantico ast =
387   (* cria ambiente global inicialmente vazio *)
388   let amb_global = Amb.novo_amb [] in
389   let _ = declara_predefinidas amb_global in
390   let (A.Programa (decs_globais, decs_funs, corpo)) = ast in
391   let _ = List.iter (insere_declaracao_var amb_global) decs_globais in
392   let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
393   (* Verifica  o de tipos nas fun  es *)
```

```
394   let decs_funs = List.map (verifica_fun amb_global) decs_funs in
395   (* Verifica  o de tipos na fun  o principal *)
396   let corpo = List.map (verifica_cmd amb_global A.TipoVoid) corpo in
397     (A.Programa (decs_globais, decs_funs, corpo), amb_global)
```

E o semantico.mli

```
1 val semantico : (Sast.expressao Ast.programa) -> Tast.expressao Ast.
    programa * Ambiente.t
```

Para execução do semântico usamos o semanticoTest.ml

```
1  open Printf
2  open Lexing
3
4  open Ast
5  exception Erro_Sintatico of string
6
7  module S = MenhirLib.General (* Streams *)
8  module I = Sintatico.MenhirInterpreter
9
10 open ErroSint
11 open Semantico
12
13 let posicao lexbuf =
14     let pos = lexbuf.lex_curr_p in
15     let lin = pos.pos_lnum
16     and col = pos.pos_cnum - pos.pos_bol - 1 in
17     sprintf "linha %d, coluna %d" lin col
18
19 (* [pilha checkpoint] extrai a pilha do aut mato LR(1) contida em
    checkpoint *)
20
21 let pilha checkpoint =
22   match checkpoint with
23   | I.HandlingError amb -> I.stack amb
24   | _ -> assert false (* Isso n o pode acontecer *)
25
26 let estado checkpoint : int =
27   match Lazy.force (pilha checkpoint) with
28   | S.Nil -> (* O parser est  no estado inicial *)
29     0
30   | S.Cons (I.Element (s, _, _, _), _) ->
31     I.number s
32
33 let sucesso v = Some v
34
35 let falha lexbuf (checkpoint : (Sast.expressao Ast.programa) I.checkpoint)
    =
36   let estado_atual = estado checkpoint in
```

```
37    let msg = message estado_atual in
38    raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
39                                          (Lexing.lexeme_start lexbuf) msg))
40
41  let loop lexbuf resultado =
42    let fornecedor = I.lexer_lexbuf_to_supplier Lexico.read lexbuf in
43    I.loop_handle sucesso (falha lexbuf) fornecedor resultado
44
45
46  let parse_com_erro lexbuf =
47    try
48      Some (loop lexbuf (Sintatico.Incremental.prog lexbuf.lex_curr_p))
49    with
50    | Lexico.Erro msg ->
51       printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
52       None
53    | Erro_Sintatico msg ->
54       printf "Erro sint tico na %s %s\n" (posicao lexbuf) msg;
55       None
56
57  let parse s =
58    let lexbuf = Lexing.from_string s in
59    let ast = parse_com_erro lexbuf in
60    ast
61
62  let parse_arq nome =
63    let ic = open_in nome in
64    let lexbuf = Lexing.from_channel ic in
65    let ast = parse_com_erro lexbuf in
66    let _ = close_in ic in
67    ast
68
69  let verifica_tipos nome =
70    let ast = parse_arq nome in
71    match ast with
72      Some (Some ast) -> semantico ast
73    | _ -> failwith "Nada a fazer!\n"
74
75  (* Para compilar:
76       ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -
             package menhirLib semanticoTest.byte
77
78     Para usar, entre no ocaml
79
80       rlwrap ocaml
81
82     e se desejar ver apenas a  rvore  sint tica que sai do analisador
```

```
        sint tico , digite

      parse_arq "exemplos/ex2.tip";;

    Depois, para ver a sa da do analisador sem ntico j  com a  rvore anotada com
    o tipos, digite:

    verifica_tipos "exemplos/ex2.tip";;

    Note que o analisador sem ntico est  retornando tamb m o ambiente global. Se
    quiser separ -los, digite:

    let (arv, amb) = verifica_tipos "exemplos/ex2.tip";;


 *)
```

0.8 Interprete

Para criar o interprete utilizaremos como base os arquivos criados no analisador semântico. Para tal iremos copiar os arquivos ambiente.ml e .mli, semantico.ml e .mli, nomeandos os para ambInterp.ml e .mli, interprete.ml e .mli

0.8.1 ambinterp.ml

```
module Tab = Tabsimb
module A = Ast
module T = Tast

type entrada_fn = {
  tipo_fn: A.tipo;
  formais: (A.ident * A.tipo) list;
  locais: (A.ident * A.tipo) list;
  corpo: T.expressao A.comandos
}

type entrada = EntFun of entrada_fn
             | EntVar of A.tipo * (T.expressao option)


type t = {
  ambv : entrada Tab.tabela
```

```
18 }
19
20 let novo_amb xs = { ambv = Tab.cria xs }
21
22 let novo_escopo amb = { ambv = Tab.novo_escopo amb.ambv }
23
24 let busca amb ch = Tab.busca amb.ambv ch
25
26 let atualiza_var amb ch t v =
27   Tab.atualiza amb.ambv ch (EntVar (t,v))
28
29 let insere_local amb nome t v =
30   Tab.insere amb.ambv nome (EntVar (t,v))
31
32 let insere_param amb nome t v =
33   Tab.insere amb.ambv nome (EntVar (t,v))
34
35 let insere_fun amb nome params locais resultado corpo =
36   let ef = EntFun { tipo_fn = resultado;
37                     formais = params;
38                     locais = locais;
39                     corpo = corpo }
40   in Tab.insere amb.ambv nome ef
```

0.8.2 ambInterp.mli

```
1 type entrada_fn = {
2   tipo_fn:  Ast.tipo;
3   formais: (string * Ast.tipo) list;
4   locais:  (string * Ast.tipo) list;
5   corpo: Tast.expressao Ast.comandos
6 }
7
8 type entrada =
9   |  EntFun of entrada_fn
10  |  EntVar of Ast.tipo *  (Tast.expressao option)
11
12 type t
13
14 val novo_amb :  (string * entrada) list -> t
15 val novo_escopo : t -> t
16 val busca:              t -> string -> entrada
17 val atualiza_var:    t -> string -> Ast.tipo ->  (Tast.expressao option) ->
       unit
18 val insere_local :    t -> string -> Ast.tipo -> (Tast.expressao option) ->
       unit
19 val insere_param : t -> string -> Ast.tipo -> (Tast.expressao option) ->
      unit
```

```
20 val insere_fun :  t ->
21   string ->
22   (string * Ast.tipo) list ->
23   (string * Ast.tipo) list ->
24   Ast.tipo -> (Tast.expressao Ast.comandos) -> unit
```

0.8.3 interprete.ml

```
1  module Amb = AmbInterp
2  module A = Ast
3  module S = Sast
4  module T = Tast
5
6  exception Valor_de_retorno of T.expressao
7
8  let obtem_nome_tipo_var exp = let open T in
9    match exp with
10   | ExpVar (v,tipo) ->
11     (match v with
12       | A.VarSimples (nome,_) -> (nome,tipo)
13     )
14   | _ -> failwith "obtem_nome_tipo_var: nao eh variavel"
15
16 let pega_nome_var var = let open A in
17   match var with
18     | VarSimples (nome,_) -> nome
19
20 let pega_int exp =
21   match exp with
22   |  T.ExpInt (i,_) -> i
23   | _ -> failwith "pega_int: nao eh inteiro"
24
25 let pega_string exp =
26   match exp with
27   |  T.ExpString (s,_) -> s
28   | _ -> failwith "pega_string: nao eh string"
29
30 let pega_bool exp =
31   match exp with
32   |  T.ExpBool (b,_) -> b
33   | _ -> failwith "pega_bool: nao eh booleano"
34
35 let pega_real exp =
36   match exp with
37   |  T.ExpReal (r,_) -> r
38   | _ -> failwith "pega_real: nao eh real"
39
40 let rec pow base exponent =
```

```
41    (match exponent with
42      | 0 -> 1
43      | 1 -> base
44      | _ -> base * pow base (exponent - 1)
45    )
46
47  type classe_op = Aritmetico | Relacional | Logico | Cadeia
48
49  let classifica op =
50    let open A in
51    match op with
52      Mais
53    | Menos
54    | Mult
55    | Div
56    | Resto
57    | Potencia -> Aritmetico
58    | Menor
59    | Igual
60    | Difer
61    | Maior
62    | MaioIgual
63    | MenorIgual -> Relacional
64    | E
65    | Ou  ->  Logico
66    | Concat  -> Cadeia
67
68  let rec interpreta_exp amb exp =
69    let open A in
70    let open T in
71    match exp with
72      ExpVoid
73    | ExpInt _
74    | ExpString _
75    | ExpChar _
76    | ExpBool _
77    | ExpReal _ -> exp
78    | ExpVar _ ->
79      let (id,tipo) = obtem_nome_tipo_var exp in
80      (match (Amb.busca amb id) with
81        | Amb.EntVar (tipo, v) ->
82          (match v with
83            | None -> failwith ("vari vel nao inicializada: " ^ id)
84            | Some valor -> valor
85          )
86        |  _ -> failwith "interpreta_exp: expvar"
87      )
```

```
88  | ExpOp ((op,top), (esq,tesq), (dir,tdir)) ->
89    let vesq = interpreta_exp amb esq
90    and vdir = interpreta_exp amb dir in
91
92    let interpreta_aritmetico () =
93      (match tesq with
94         TipoReal ->
95         (match op with
96            | Mais      ->     ExpReal (pega_real vesq +. pega_real vdir,
                 top)
97            | Menos     ->     ExpReal (pega_real vesq -. pega_real vdir,
                 top)
98            | Mult      ->     ExpReal (pega_real vesq *. pega_real vdir,
                 top)
99            | Div       ->     ExpReal (pega_real vesq /. pega_real vdir,
                 top)
100           | Potencia  ->     ExpReal (pega_real vesq ** pega_real vdir,
                 top)
101           | _ -> failwith "Expressao aritmetica real invalida"
102         )
103       | TipoInt ->
104          (match op with
105            | Mais      ->     ExpInt (pega_int vesq + pega_int vdir, top)
106            | Menos     ->     ExpInt (pega_int vesq - pega_int vdir, top)
107            | Mult      ->     ExpInt (pega_int vesq * pega_int vdir, top)
108            | Div       ->     ExpInt (pega_int vesq / pega_int vdir, top)
109            | Resto     ->     ExpInt (pega_int vesq mod pega_int vdir, top
                 )
110            | Potencia  ->     ExpInt ((pow (pega_int vesq) (pega_int vdir)
                 ), top)
111           | _ -> failwith "Expressao aritmetica real invalida"
112         )
113       | _ -> failwith "Tipo de expressao aritmetica invalido"
114     )
115
116   and interpreta_relacional () =
117     (match tesq with
118     | TipoInt ->
119       (match op with
120       | Menor -> ExpBool (pega_int vesq < pega_int vdir, top)
121       | Maior  -> ExpBool (pega_int vesq > pega_int vdir, top)
122       | MenorIgual -> ExpBool (pega_int vesq <= pega_int vdir, top)
123       | MaioIgual  -> ExpBool (pega_int vesq >= pega_int vdir, top)
124       | Igual   -> ExpBool (pega_int vesq == pega_int vdir, top)
125       | Difer   -> ExpBool (pega_int vesq != pega_int vdir, top)
126       | _ -> failwith "interpreta_relacional"
127       )
```

```
128       | TipoString ->
129         (match op with
130         | Menor -> ExpBool (pega_string vesq < pega_string vdir, top)
131         | Maior  -> ExpBool (pega_string vesq > pega_string vdir, top)
132         | MenorIgual -> ExpBool (pega_string vesq <= pega_string vdir, top)
133         | MaioIgual  -> ExpBool (pega_string vesq >= pega_string vdir, top)
134         | Igual   -> ExpBool (pega_string vesq = pega_string vdir, top)
135         | Difer   -> ExpBool (pega_string vesq <> pega_string vdir, top)
136         | _ -> failwith "interpreta_relacional"
137         )
138       | TipoBool ->
139         (match op with
140         | Menor -> ExpBool (pega_bool vesq < pega_bool vdir, top)
141         | Maior  -> ExpBool (pega_bool vesq > pega_bool vdir, top)
142         | MenorIgual -> ExpBool (pega_bool vesq <= pega_bool vdir, top)
143         | MaioIgual  -> ExpBool (pega_bool vesq >= pega_bool vdir, top)
144         | Igual   -> ExpBool (pega_bool vesq == pega_bool vdir, top)
145         | Difer   -> ExpBool (pega_bool vesq != pega_bool vdir, top)
146         | _ -> failwith "interpreta_relacional"
147         )
148       | TipoReal ->
149         (match op with
150         | Menor -> ExpBool (pega_real vesq < pega_real vdir, top)
151         | Maior  -> ExpBool (pega_real vesq > pega_real vdir, top)
152         | MenorIgual -> ExpBool (pega_real vesq <= pega_real vdir, top)
153         | MaioIgual  -> ExpBool (pega_real vesq >= pega_real vdir, top)
154         | Igual   -> ExpBool (pega_real vesq = pega_real vdir, top)
155         | Difer   -> ExpBool (pega_real vesq <> pega_real vdir, top)
156         | _ -> failwith "interpreta_relacional"
157         )
158       | _ ->  failwith "interpreta_relacional"
159       )
160
161     and interpreta_logico () =
162       (match tesq with
163        | TipoBool ->
164          (match op with
165           | Ou -> ExpBool (pega_bool vesq || pega_bool vdir, top)
166           | E ->   ExpBool (pega_bool vesq && pega_bool vdir, top)
167           | _ ->  failwith "interpreta_logico"
168          )
169        | _ ->  failwith "interpreta_logico"
170       )
171     and interpreta_cadeia () =
172       (match tesq with
173        | TipoString ->
174          (match op with
```

```
175             | Concat -> ExpString (pega_string vesq ^ pega_string vdir, top)
176             | _ ->  failwith "interpreta_cadeia"
177           )
178         | _ ->  failwith "interpreta_cadeia"
179       )
180
181     in
182     let valor = (match (classifica op) with
183           Aritmetico -> interpreta_aritmetico ()
184         | Relacional -> interpreta_relacional ()
185         | Logico -> interpreta_logico ()
186         | Cadeia -> interpreta_cadeia ()
187       )
188     in
189       valor
190
191   | ExpFuncao (nome, args, tipo) ->
192      let open Amb in
193     ( match (Amb.busca amb nome) with
194       | Amb.EntFun {tipo_fn; formais; locais; corpo} ->
195             (* Interpreta cada um dos argumentos *)
196             let vargs = List.map (interpreta_exp amb) args in
197             (* Associa os argumentos aos par metros formais *)
198             let vformais = List.map2 (fun (n,t) v -> (n, t, Some v)) formais
                    vargs
199             in interpreta_fun amb nome vformais locais corpo
200       | _ -> failwith "interpreta_exp: expchamada"
201     )
202
203 and interpreta_cmd amb cmd =
204   let open A in
205   let open T in
206   match cmd with
207     CmdReturn exp ->
208       let e1 = interpreta_exp amb exp in
209            raise (Valor_de_retorno e1)
210   | CmdSe (teste, entao, senao) ->
211     let teste1 = interpreta_exp amb teste in
212       (match teste1 with
213         ExpBool (true,_) ->
214           (* Interpreta cada comando do bloco 'ent o' *)
215           List.iter (interpreta_cmd amb) entao
216         | _ ->
217           (* Interpreta cada comando do bloco 'sen o', se houver *)
218           (match senao with
219               None -> ()
220             | Some bloco -> List.iter (interpreta_cmd amb) bloco
```

```
221             )
222         )
223
224     | CmdAtrib (elem, exp) ->
225       (* Interpreta o lado direito da atribui  o *)
226       let exp = interpreta_exp amb exp in
227       (* Faz o mesmo para o lado esquerdo *)
228       let name = (pega_nome_var elem)
229       in (match (Amb.busca amb name) with
230             | Amb.EntVar (tipo, v) ->
231               Amb.atualiza_var amb name tipo (Some exp)
232             | _ -> failwith "cmd_atrib: EntVar"
233           )
234
235     | CmdChamada exp -> ignore( interpreta_exp amb exp)
236
237     | CmdEntrada exps ->
238       (* Obtem os nomes e os tipos de cada um dos argumentos *)
239       let nts = List.map (obtem_nome_tipo_var) exps in
240       let leia_var (nome,tipo) =
241         let valor =
242           (match tipo with
243            | A.TipoInt    -> T.ExpInt    (read_int (), tipo)
244            | A.TipoString -> T.ExpString (read_line (), tipo)
245            | A.TipoReal ->   T.ExpReal   (read_float (), tipo)
246            | _ -> failwith "leia_var: nao implementado"
247           )
248         in Amb.atualiza_var amb nome tipo (Some valor)
249       in
250       (* L  o valor para cada argumento e atualiza o ambiente *)
251       List.iter leia_var nts
252
253
254     | CmdSaida exps ->
255       (* Interpreta cada argumento da fun  o 'saida' *)
256       let exps = List.map (interpreta_exp amb) exps in
257       let imprima exp =
258         (match exp with
259          | T.ExpInt (n,_) ->      let _ = print_int n in print_string " "
260          | T.ExpReal (n,_) ->     let _ = print_float n in print_string " "
261          | T.ExpString (s,_) -> let _ = print_string s in print_string " "
262          | T.ExpBool (b,_) ->
263            let _ = print_string (if b then "true" else "false")
264            in print_string " "
265          | _ -> failwith "imprima: nao implementado"
266         )
267       in List.iter imprima exps
```

```
268   | CmdSaidaLine exps ->
269     (* Interpreta cada argumento da fun   o 'saida' *)
270     let exps = List.map (interpreta_exp amb) exps in
271     let imprima exp =
272       (match exp with
273        | T.ExpInt (n,_) ->      let _ = print_int n in print_string " "
274        | T.ExpReal (n,_) ->     let _ = print_float n in print_string " "
275        | T.ExpString (s,_) -> let _ = print_string s in print_string " "
276        | T.ExpBool (b,_) ->
277          let _ = print_string (if b then "true" else "false")
278          in print_string " "
279        | _ -> failwith "imprima: nao implementado"
280       )
281     in
282     let _ = List.iter imprima exps in
283     print_newline ()
284   | CmdFor( variavel, inicio, fim, comandos ) ->
285       ( match inicio with
286           | ExpInt _ ->
287               let vfim = pega_int (interpreta_exp amb fim)
288               and vinicio = pega_int (interpreta_exp amb inicio) in
289                 let incremento = if vfim > vinicio
290                                  then 1
291                                else (-1)
292                 in
293               let rec para atual cmds =
294                 if atual <> (pega_int fim)
295                   then let _ = List.iter (interpreta_cmd amb) cmds in
296                         para (atual + incremento) cmds
297               in para vinicio comandos
298           | ExpReal _ ->
299               let vfim = pega_real (interpreta_exp amb fim)
300               and vinicio = pega_real (interpreta_exp amb inicio) in
301                 let incremento = if vfim > vinicio then 1.
302                                  else -1.
303                 in
304               let rec para atual cmds =
305                 if atual <> pega_real(fim)
306                   then let _ = List.iter (interpreta_cmd amb) cmds in
307                         para (atual +. incremento) cmds
308               in para vinicio comandos
309           | _ -> failwith "Tipo invalido para o inicio e o fim do comando
                    para"
310       )
311   | CmdSwitch ( variavel, cases, default ) ->
312       let rec verifica_cases v cs =
313           (match cs with
```

```
314             | [] -> false
315             | (CmdCase (exp, cmds))::cs ->
316               let exp1 = interpreta_exp amb exp in
317                 if exp1 = v
318                   then let _ = List.iter (interpreta_cmd amb) cmds in
319                   true
320                 else verifica_cases v cs
321           )
322       in
323       let id = pega_nome_var variavel in
324       (match (Amb.busca amb id) with
325         | Amb.EntVar (tipo, v) ->
326           (match v with
327             | None -> failwith ("vari vel nao inicializada: " ^ id)
328             | Some valor -> if not (verifica_cases valor cases)
329                 then  (
330                         match default with
331                           | Some CmdCaseDefault cmds -> List.iter (
                                interpreta_cmd amb) cmds
332                           | _ -> failwith "Switch: CaseDefault"
333                       )
334           )
335         |  _ -> failwith "interpreta_exp: expvar"
336       )
337   | CmdEnquanto (teste, entao) ->
338     let rec laco cmds =
339         let condValue = interpreta_exp amb teste in
340           (
341             match condValue with
342               | ExpBool (true,_) ->
343                 let _ = List.iter (interpreta_cmd amb) cmds in
344                   laco cmds
345               | _ -> ()
346           )
347       in laco entao
348 and interpreta_fun amb fn_nome fn_formais fn_locais fn_corpo =
349   let open A in
350  (* Estende o ambiente global, adicionando um ambiente local *)
351   let ambfn = Amb.novo_escopo amb in
352    let insere_local  d =
353     match d with
354       (v,t) -> Amb.insere_local ambfn v  t None
355   in
356   (* Associa os argumentos aos par metros e insere no novo ambiente *)
357   let insere_parametro (n,t,v) = Amb.insere_param ambfn n t v in
358   let _ = List.iter insere_parametro fn_formais in
359   (* Insere as vari veis locais no novo ambiente *)
```

```
360     let _ = List.iter insere_local fn_locais in
361     (* Interpreta cada comando presente no corpo da fun   o usando o novo
362        ambiente *)
363   try
364     let _ = List.iter (interpreta_cmd ambfn) fn_corpo in T.ExpVoid
365     with
366        Valor_de_retorno expret -> expret
367
368
369 let rec obtem_formais xs =
370   match xs with
371     [] -> []
372   | (nome,t)::xs ->
373     let id = fst nome in
374     if (List.for_all (fun (n,t) -> (fst n) <> id) xs)
375     then (id, t) :: obtem_formais xs
376     else let msg = "Parametro duplicado " ^ id in
377       failwith msg
378
379 let insere_declaracao_var amb dec =
380   let open A in
381     match dec with
382         (nome, tipo) ->  Amb.insere_local amb (fst nome) tipo None
383
384 let insere_declaracao_fun amb dec =
385   let open A in
386     match dec with
387     Funcao {fn_nome; fn_nome_fecha; fn_tiporet; fn_formais; fn_locais;
          fn_corpo} ->
388         let nome = fst fn_nome in
389         let formais = List.map (fun (n,t) -> ((fst n), t)) fn_formais in
390         let locais = List.map (fun (n,t) -> ((fst n), t)) fn_locais in
391         Amb.insere_fun amb nome
392         formais
393         locais
394         fn_tiporet
395         fn_corpo
396
397
398 (* Lista de cabe alhos das fun   es pr   definidas *)
399 let fn_predefs = let open A in [
400    ("entrada", [("x", TipoInt); ("y", TipoInt)], TipoVoid, []);
401    ("saida",   [("x", TipoInt); ("y", TipoInt)], TipoVoid, [])
402 ]
403
404 (* insere as fun   es pr   definidas no ambiente global *)
405 let declara_predefinidas amb =
```

```
406   List.iter (fun (n,ps,tr,c) -> Amb.insere_fun amb n ps [] tr c) fn_predefs
407
408 let interprete ast =
409   (* cria ambiente global inicialmente vazio *)
410   let amb_global = Amb.novo_amb [] in
411   let _ = declara_predefinidas amb_global in
412   let (A.Programa (decs_globais, decs_funs, corpo)) = ast in
413   let _ = List.iter (insere_declaracao_var amb_global) decs_globais in
414   let _ = List.iter (insere_declaracao_fun amb_global) decs_funs in
415   (* Interpreta a fun   o principal *)
416   let resultado = List.iter (interpreta_cmd amb_global) corpo in
417   resultado
```

0.8.4 interprete.mli

```
1 val interprete :  Tast.expressao Ast.programa -> unit
```

0.8.5 interpreteTest.ml

```
1  open Printf
2  open Lexing
3
4  open Ast
5  exception Erro_Sintatico of string
6
7  module S = MenhirLib.General (* Streams *)
8  module I = Sintatico.MenhirInterpreter
9
10 open ErroSint
11 open Semantico
12
13 let posicao lexbuf =
14     let pos = lexbuf.lex_curr_p in
15     let lin = pos.pos_lnum
16     and col = pos.pos_cnum - pos.pos_bol - 1 in
17     sprintf "linha %d, coluna %d" lin col
18
19 (* [pilha checkpoint] extrai a pilha do aut mato LR(1) contida em
      checkpoint *)
20
21 let pilha checkpoint =
22   match checkpoint with
23   | I.HandlingError amb -> I.stack amb
24   | _ -> assert false (* Isso n o pode acontecer *)
25
26 let estado checkpoint : int =
27   match Lazy.force (pilha checkpoint) with
28   | S.Nil -> (* O parser est  no estado inicial *)
```

```
29        0
30    | S.Cons (I.Element (s, _, _, _), _) ->
31        I.number s
32
33  let sucesso v = Some v
34
35  let falha lexbuf (checkpoint : (Sast.expressao Ast.programa) I.checkpoint)
      =
36    let estado_atual = estado checkpoint in
37    let msg = message estado_atual in
38    raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
39                                          (Lexing.lexeme_start lexbuf) msg))
40
41  let loop lexbuf resultado =
42    let fornecedor = I.lexer_lexbuf_to_supplier Lexico.read lexbuf in
43    I.loop_handle sucesso (falha lexbuf) fornecedor resultado
44
45
46  let parse_com_erro lexbuf =
47    try
48      Some (loop lexbuf (Sintatico.Incremental.prog lexbuf.lex_curr_p))
49    with
50    | Lexico.Erro msg ->
51        printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
52        None
53    | Erro_Sintatico msg ->
54        printf "Erro sint tico na %s %s\n" (posicao lexbuf) msg;
55        None
56
57  let parse s =
58    let lexbuf = Lexing.from_string s in
59    let ast = parse_com_erro lexbuf in
60    ast
61
62  let parse_arq nome =
63    let ic = open_in nome in
64    let lexbuf = Lexing.from_channel ic in
65    let ast = parse_com_erro lexbuf in
66    let _ = close_in ic in
67    ast
68
69  let verifica_tipos nome =
70    let ast = parse_arq nome in
71    match ast with
72      Some (Some ast) -> semantico ast
73    | _ -> failwith "Nada a fazer!\n"
74
```

```
75 let interprete nome =
76   let tast,amb = verifica_tipos nome in
77   Interprete.interprete tast
```

0.9 Geração de Código Intermediário

Na geração de código intermediário por sua vez utilizaremos como base o interprete para isso copiamos o interprete.ml e criamos o Cod3End.ml, precisaremos criar também o cod3endTest.ml e o Codigo.ml

0.9.1 Cod3End.ml

```
1  open Printf
2
3  open Ast
4  open Tast
5  open Codigo
6
7  let conta_temp = ref 0
8  let conta_rotulos = ref (Hashtbl.create 5)
9
10 let zera_contadores () =
11   begin
12     conta_temp := 0;
13     conta_rotulos := Hashtbl.create 5
14   end
15
16 let novo_temp () =
17    let numero = !conta_temp in
18    let _ = incr conta_temp in
19    Temp numero
20
21 let novo_rotulo prefixo =
22   if Hashtbl.mem !conta_rotulos prefixo
23   then
24      let numero = Hashtbl.find !conta_rotulos prefixo in
25      let _ = Hashtbl.replace !conta_rotulos prefixo (numero + 1) in
26      Rotulo (prefixo ^ (string_of_int numero))
27   else
28      let _ = Hashtbl.add !conta_rotulos prefixo 1 in
29      Rotulo (prefixo ^ "0")
30
31 (* Codigo para impress o *)
32
33 let endr_to_str = function
34    | Nome s -> s
```

```
35    | ConstInt n -> string_of_int n
36    | ConstReal n -> string_of_float n
37    | ConstString s -> s
38    | ConstChar c -> String.make 1 c
39    | ConstBool b -> string_of_bool b
40    | Temp n  -> "t" ^ string_of_int n
41
42 let tipo_to_str t =
43     match t with
44       TipoInt -> "inteiro"
45     | TipoString -> "string"
46     | TipoBool -> "bool"
47     | TipoVoid -> "void"
48     | TipoReal -> "real"
49
50 let op_to_str op =
51   match op with
52   | Mais  -> "+"
53   | Menos -> "-"
54   | Mult  -> "*"
55   | Div   -> "/"
56   | Resto   -> "%"
57   | Potencia   -> "^"
58   | Menor -> "<"
59   | Igual -> "="
60   | Difer -> "!="
61   | Maior -> ">"
62   | MaioIgual -> ">="
63   | MenorIgual -> "<="
64   | E     -> "&&"
65   | Ou    -> "||"
66   | Concat -> "^"
67
68 let rec args_to_str ats =
69    match ats with
70    | [] -> ""
71    | [(a,t)] ->
72      let str = sprintf "(%s,%s)" (endr_to_str a) (tipo_to_str t) in
73      str
74    | (a,t) :: ats ->
75      let str = sprintf "(%s,%s)" (endr_to_str a) (tipo_to_str t) in
76      str ^ ", " ^ args_to_str ats
77
78 let rec escreve_cod3 c =
79   match c with
80   | AtribBin (x,y,op,z) ->
81       sprintf "%s := %s %s %s\n" (endr_to_str x)
```

```
82                                    (endr_to_str y) (op_to_str (fst op)) (
                                          endr_to_str z)
83    | AtribUn (x,op,y) ->
84        sprintf "%s := %s %s\n" (endr_to_str x) (op_to_str (fst op)) (
              endr_to_str y)
85    | Copia (x,y) ->
86        sprintf "%s := %s\n" (endr_to_str x) (endr_to_str y)
87    | Goto l ->
88        sprintf "goto %s\n" (escreve_cod3 l)
89    | If (x,l) ->
90        sprintf "if %s goto %s\n" (endr_to_str x) (escreve_cod3 l)
91    | IfFalse (x,l) ->
92        sprintf "ifFalse %s goto %s\n" (endr_to_str x) (escreve_cod3 l)
93    | IfRelgoto (x,oprel,y,l) ->
94        sprintf "if %s %s %s goto %s\n" (endr_to_str x) (op_to_str (fst oprel
              ))
95                                          (endr_to_str y) (escreve_cod3 l)
96    | Call (p,ats,t) -> sprintf "call %s(%s): %s\n" p (args_to_str ats) (
        tipo_to_str t)
97    | Recebe (x,t) -> sprintf "recebe %s,%s\n" x (tipo_to_str t)
98    | Local (x,t)  -> sprintf "local %s,%s\n" x (tipo_to_str t)
99    | Global (x,t) -> sprintf "global %s,%s\n" x (tipo_to_str t)
100   | CallFn (x,p,ats,t) ->
101       sprintf "%s := call %s(%s): %s\n" (endr_to_str x) p (args_to_str ats)
              (tipo_to_str t)
102   | Return x -> sprintf "return %s\n" (endr_to_str x)
103   | BeginFun (id,np,nl) -> sprintf "beginFun %s(%d,%d)\n" id np nl
104   | EndFun -> "endFun\n\n"
105   | Rotulo r -> sprintf "%s: " r
106
107
108 let rec escreve_codigo cod =
109   match cod with
110   | [] -> printf "\n"
111   | c::cs -> printf "%s" (escreve_cod3 c);
112              escreve_codigo cs
113
114 (* C digo do tradutor para c digo de 3 endere o *)
115
116 let obtem_nome_tipo_var exp =
117   (
118     match exp with
119     | ExpVar (v,tipo) ->
120       (match v with
121         | VarSimples (nome,_) -> (nome,tipo)
122       )
123     | _ -> failwith "obtem_nome_tipo_var: nao eh variavel"
```

```
124   )
125 let pega_nome_var var =
126   (
127     match var with
128       | VarSimples (nome,_) -> nome
129   )
130
131 let pega_tipo exp =
132   match exp with
133   | ExpVar (v, t) -> t
134   | ExpInt (n, t) -> t
135   | ExpString (s, t) -> t
136   | ExpChar (c, t) -> t
137   | ExpBool (b, t) -> t
138   | ExpReal (r, t) -> t
139   | ExpFuncao (id, args, t) -> t
140   | ExpOp ((op,t),_,_) -> t
141   | ExpVoid -> TipoVoid
142
143
144 let rec traduz_exp exp =
145   match exp with
146   | ExpInt (n, TipoInt) ->
147      let t = novo_temp () in
148     (t, [Copia (t, ConstInt n)])
149   | ExpString (n, TipoString) ->
150      let t = novo_temp () in
151     (t, [Copia (t, ConstString n)])
152   | ExpChar  (n, TipoString) ->
153      let t = novo_temp () in
154     (t, [Copia (t, ConstChar n)])
155   | ExpBool  (n, TipoBool) ->
156      let t = novo_temp () in
157     (t, [Copia (t, ConstBool n)])
158   | ExpReal  (n, TipoReal) ->
159      let t = novo_temp () in
160     (t, [Copia (t, ConstReal n)])
161   | ExpVar (v, tipo) ->
162     (match v with
163        VarSimples nome ->
164        let id = fst nome in
165        ((Nome id), [])
166     )
167   | ExpOp (op, exp1, exp2) ->
168     let (endr1, codigo1) = let (e1,t1) = exp1 in traduz_exp e1
169     and (endr2, codigo2) = let (e2,t2) = exp2 in traduz_exp e2
170     and t = novo_temp () in
```

```
171       let codigo = codigo1 @ codigo2 @ [AtribBin (t, endr1, op, endr2)] in
172       (t, codigo)
173     | ExpFuncao (id, args, tipo_fn) ->
174         let (enderecos, codigos) = List.split (List.map traduz_exp args) in
175         let tipos = List.map pega_tipo args in
176         let endr_tipos = List.combine enderecos tipos
177         and t = novo_temp () in
178         let codigo = (List.concat codigos) @
179                      [CallFn (t, id, endr_tipos, tipo_fn)]
180         in
181           (t, codigo)
182     | _ -> failwith "traduz_exp: n o implementado"
183
184 and traduz_cmd cmd =
185   match cmd with
186     CmdReturn exp ->
187       let (endr_exp, codigo_exp) = traduz_exp exp in
188       codigo_exp @ [Return (endr_exp)]
189   | CmdSe (teste, entao, senao) ->
190     let (endr_teste, codigo_teste) = traduz_exp teste
191     and codigo_entao = traduz_cmds entao
192     and rotulo_falso = novo_rotulo "L" in
193     (match senao with
194         | None -> codigo_teste @
195                   [IfFalse (endr_teste, rotulo_falso)] @
196                   codigo_entao @
197                   [rotulo_falso]
198         | Some cmds ->
199           let codigo_senao = traduz_cmds cmds
200           and rotulo_fim = novo_rotulo "L" in
201               codigo_teste @
202               [IfFalse (endr_teste, rotulo_falso)] @
203               codigo_entao @
204               [Goto rotulo_fim] @
205               [rotulo_falso] @ codigo_senao @
206               [rotulo_fim]
207     )
208
209   | CmdAtrib (elem, exp) ->
210     let (endr_exp, codigo_exp) = traduz_exp exp
211     and id = pega_nome_var elem in
212     let codigo = codigo_exp @ [Copia (Nome id, endr_exp)]
213     in codigo
214
215   | CmdChamada (ExpFuncao (id, args, tipo_fn)) ->
216       let (enderecos, codigos) = List.split (List.map traduz_exp args) in
217       let tipos = List.map pega_tipo args in
```

```
218       let endr_tipos = List.combine enderecos tipos in
219       (List.concat codigos) @
220       [Call (id, endr_tipos, tipo_fn)]
221   | CmdEntrada args ->
222       let (enderecos, codigos) = List.split (List.map traduz_exp args) in
223       let tipos = List.map pega_tipo args in
224       let endr_tipos = List.combine enderecos tipos in
225       (List.concat codigos) @
226       [Call ("read", endr_tipos, TipoVoid)]
227
228   | CmdSaida args ->
229       let (enderecos, codigos) = List.split (List.map traduz_exp args) in
230       let tipos = List.map pega_tipo args in
231       let endr_tipos = List.combine enderecos tipos in
232       (List.concat codigos) @
233       [Call ("print", endr_tipos, TipoVoid)]
234   | CmdSaidaLine args ->
235       let (enderecos, codigos) = List.split (List.map traduz_exp args) in
236       let tipos = List.map pega_tipo args in
237       let endr_tipos = List.combine enderecos tipos in
238       (List.concat codigos) @
239       [Call ("print_line", endr_tipos, TipoVoid)]
240
241   | CmdFor( variavel, inicio, fim, comandos ) ->
242       let t = novo_temp()
243       and tIncremento = novo_temp()
244       and (end_inicio, codigo_inicio) = traduz_exp inicio
245       and (end_fim, codigo_fim) = traduz_exp fim
246       and corpo = traduz_cmds comandos
247       and id = pega_nome_var variavel
248       and tipo = pega_tipo inicio
249       and rotulo_for = novo_rotulo "R"
250       and rotulo_fim = novo_rotulo "R"
251       and rotulo_decrementa = novo_rotulo "R"
252       and rotulo_inicio_for = novo_rotulo "R" in
253       let incremento =
254         (
255           match tipo with
256             | TipoInt -> ConstInt 1
257             | TipoReal -> ConstReal 1.0
258             | _ -> failwith "Tipo nao pode ser usado no for"
259         )
260       and decremento =
261         (
262           match tipo with
263             | TipoInt -> ConstInt (-1)
264             | TipoReal -> ConstReal (-1.0)
```

```
265             | _ -> failwith "Tipo nao pode ser usado no for"
266         ) in
267
268
269       (* codigo para verificar se o passo do for
270       esta incrementando ou decrementando *)
271
272       codigo_inicio @
273       codigo_fim @
274       [AtribBin (t, end_inicio, (Menor, tipo), end_fim)] @
275       [IfFalse(t, rotulo_decrementa)] @
276       [Copia(tIncremento, incremento)] @
277       [Goto rotulo_inicio_for] @
278       [rotulo_decrementa] @ [Copia(tIncremento, decremento)] @
279
280       (* fim do codigo do incremento ou decremento *)
281
282       [rotulo_inicio_for] @
283       [Copia (Nome id, end_inicio)] @
284       [rotulo_for] @
285       codigo_fim @
286
287       [AtribBin (t, Nome id, (Igual, tipo), end_fim)] @
288       [IfFalse(t, rotulo_fim)] @
289       corpo @
290       [AtribBin (t, Nome id, (Mais, tipo), tIncremento)] @
291       [Copia (Nome id, t)] @
292       [Goto rotulo_for] @
293       [rotulo_fim]
294
295   | CmdSwitch ( variavel, cases, default ) ->
296     let traduz_cases tempVar rotuloFinal case =
297       (
298         match case with
299           | CmdCase (exp, corpo) ->
300             let (endr_exp, codigo_exp) = traduz_exp exp
301             and codigo_corpo = traduz_cmds corpo
302             and tipo_exp = pega_tipo exp
303             and t = novo_temp()
304             and rotulo_false = novo_rotulo "S" in
305
306             codigo_exp @
307             [AtribBin (t, tempVar, (Igual, tipo_exp), endr_exp)] @
308             [IfFalse (t, rotulo_false)] @
309             codigo_corpo @
310             [Goto rotuloFinal] @
311             [rotulo_false]
```

```
312       )
313     in
314     let tvar = novo_temp ()
315     and rotulo_final = novo_rotulo "S" in
316     let cases_codigo = List.concat (List.map (traduz_cases tvar
            rotulo_final) cases) in
317     let codigo =  [Copia (tvar, Nome (pega_nome_var variavel))] @
318                   cases_codigo
319     in
320     (match default with
321       | None -> codigo @
322                 [rotulo_final]
323       | Some CmdCaseDefault(cmds) ->
324         let codigo_default = traduz_cmds cmds in
325         codigo @
326         codigo_default @
327         [rotulo_final]
328     )
329
330   | CmdEnquanto (teste, entao) ->
331       let (endr_teste, codigo_teste) = traduz_exp teste
332       and codigo_entao = traduz_cmds entao
333       and rotulo_inicio = novo_rotulo "E"
334       and rotulo_falso  = novo_rotulo "E" in
335       [rotulo_inicio] @
336       codigo_teste @
337       [IfFalse (endr_teste, rotulo_falso)] @
338       codigo_entao @
339       [Goto rotulo_inicio] @
340       [rotulo_falso]
341   | _ -> failwith "comando nao implementado"
342
343 and traduz_cmds cmds =
344   match cmds with
345   | [] -> []
346   | cmd :: cmds ->
347      let codigo = traduz_cmd cmd in
348      codigo @ traduz_cmds cmds
349
350 let traduz_fun ast =
351   match ast with
352     Funcao {fn_nome; fn_tiporet; fn_formais; fn_locais; fn_corpo} ->
353       let nome = fst fn_nome
354       and formais = List.map (fun ((id,pos),tipo) -> Recebe (id,tipo))
              fn_formais
355       and nformais = List.length fn_formais
```

```
356         and locais = List.map (fun ((id,pos),tipo) -> Local (id,tipo))
                fn_locais
357         and nlocais = List.length fn_locais
358         and corpo = traduz_cmds fn_corpo
359         in
360         [BeginFun (nome,nformais,nlocais)] @ formais @ locais @ corpo @ [
                EndFun]
361
362
363 let tradutor ast_tipada =
364   let _ = zera_contadores () in
365   let (Programa (decs_globais, decs_funs, corpo)) = ast_tipada in
366   let globais_trad = List.map (fun ((id,pos),tipo) -> Global (id,tipo))
          decs_globais in
367   let funs_trad = List.map traduz_fun decs_funs in
368   let corpo_trad = traduz_cmds corpo in
369   globais_trad @ (List.concat funs_trad) @
370   [BeginFun ("main",0,0)] @ corpo_trad @ [EndFun]
```

0.9.2 cod3endTest.ml

```
1  open Printf
2  open Lexing
3
4  open Ast
5  exception Erro_Sintatico of string
6
7  module S = MenhirLib.General (* Streams *)
8  module I = Sintatico.MenhirInterpreter
9
10 open ErroSint
11 open Semantico
12 open Codigo
13 open Cod3End
14
15 let posicao lexbuf =
16     let pos = lexbuf.lex_curr_p in
17     let lin = pos.pos_lnum
18     and col = pos.pos_cnum - pos.pos_bol - 1 in
19     sprintf "linha %d, coluna %d" lin col
20
21 (* [pilha checkpoint] extrai a pilha do aut mato LR(1) contida em
       checkpoint *)
22
23 let pilha checkpoint =
24   match checkpoint with
25   | I.HandlingError amb -> I.stack amb
26   | _ -> assert false (* Isso n o pode acontecer *)
```

```
27
28 let estado checkpoint : int =
29   match Lazy.force (pilha checkpoint) with
30   | S.Nil -> (* O parser est  no estado inicial *)
31       0
32   | S.Cons (I.Element (s, _, _, _), _) ->
33       I.number s
34
35 let sucesso v = Some v
36
37 let falha lexbuf (checkpoint : (Sast.expressao Ast.programa) I.checkpoint)
     =
38   let estado_atual = estado checkpoint in
39   let msg = message estado_atual in
40   raise (Erro_Sintatico (Printf.sprintf "%d - %s.\n"
41                                         (Lexing.lexeme_start lexbuf) msg))
42
43 let loop lexbuf resultado =
44   let fornecedor = I.lexer_lexbuf_to_supplier Lexico.read lexbuf in
45   I.loop_handle sucesso (falha lexbuf) fornecedor resultado
46
47
48 let parse_com_erro lexbuf =
49   try
50     Some (loop lexbuf (Sintatico.Incremental.prog lexbuf.lex_curr_p))
51   with
52   | Lexico.Erro msg ->
53       printf "Erro lexico na %s:\n\t%s\n" (posicao lexbuf) msg;
54       None
55   | Erro_Sintatico msg ->
56       printf "Erro sint tico na %s %s\n" (posicao lexbuf) msg;
57       None
58
59 let parse s =
60   let lexbuf = Lexing.from_string s in
61   let ast = parse_com_erro lexbuf in
62   ast
63
64 let parse_arq nome =
65   let ic = open_in nome in
66   let lexbuf = Lexing.from_channel ic in
67   let ast = parse_com_erro lexbuf in
68   let _ = close_in ic in
69   ast
70
71 let verifica_tipos nome =
72   let ast = parse_arq nome in
```

```
73    match ast with
74      Some (Some ast) -> semantico ast
75    | _ -> failwith "Nada a fazer!\n"
76
77  let traduz nome =
78    let (arv,tab) = verifica_tipos nome in
79    tradutor arv
80
81  let imprime_traducao cod =
82     let _ = printf "\n" in
83     escreve_codigo cod
84
85
86  (* Para compilar:
87        ocamlbuild -use-ocamlfind -use-menhir -menhir "menhir --table" -
              package menhirLib cod3endTest.byte
88
89     Para usar, entre no ocaml
90
91       rlwrap ocaml
92
93     e se desejar ver apenas a  rvore  sint tica que sai do analisador
           sint tico, digite
94
95       parse_arq "exemplos/Tipos/ex8.tip";;
96
97     Depois, para ver a sa da do analisador sem ntico j  com a  rvore
           anotada com
98     o tipos, digite:
99
100    verifica_tipos "exemplos/Tipos/ex8.tip";;
101
102    Note que o analisador sem ntico est  retornando tamb m o ambiente
           global. Se
103    quiser separ -los, digite:
104
105    let (arv, amb) = verifica_tipos "exemplos/Tipos/ex8.tip";;
106
107    Para ver o c digo de 3 endere os:
108    traduz "exemplos/Tipos/ex8.tip";;
109
110    ou se quiser ver em um formato mais leg vel:
111
112    let cod = traduz "exemplos/Tipos/ex8.tip" in imprime_traducao cod;;
113
114
115
```

```
116
117  *)
```

0.9.3 Codigo.ml

```
1  open Ast
2  open Tast
3
4  type endereco =
5       Nome of string
6     | ConstInt of int
7     | ConstString of string
8     | ConstChar of char
9     | ConstBool of bool
10    | ConstReal of float
11    | Temp of int
12 and instrucao =
13      AtribBin of endereco * endereco * opBin * endereco  (* x = y op z *)
14    | AtribUn  of endereco * opUn * endereco              (* x = op y   *)
15    | Copia of endereco * endereco                        (* x = y      *)
16    | Goto of instrucao                                   (* goto L     *)
17    | If of endereco *  instrucao                         (* if x goto L *)
18    | IfFalse of endereco * instrucao                     (* ifFalse x goto
        L *)
19    | IfRelgoto of endereco * opRel * endereco * instrucao
20                                                          (* if x oprel y goto
                                                             L *)
21    | Call of string * (endereco * Ast.tipo) list * Ast.tipo (* call p,[(x,t
        )],t *)
22    | Recebe of string * Ast.tipo
23    | Local of string * Ast.tipo
24    | Global of string * Ast.tipo
25    | CallFn of endereco * string *  (endereco * Ast.tipo) list * Ast.tipo
            (* x = call p,n,t *)
26    | Return of endereco
27    | BeginFun of string * int * int      (* beginFun p,nparam, nlocais *)
28    | EndFun
29    | Rotulo of string
30
31 and opBin = Ast.oper * Ast.tipo
32
33 and opUn = Ast.oper * Ast.tipo
34
35 and opRel = Ast.oper * Ast.tipo
```