Protótipo de um gerador de analisador léxico

Michel Nogueira Rebelo

Orientando

Joyce Martins

Orientadora

Roteiro

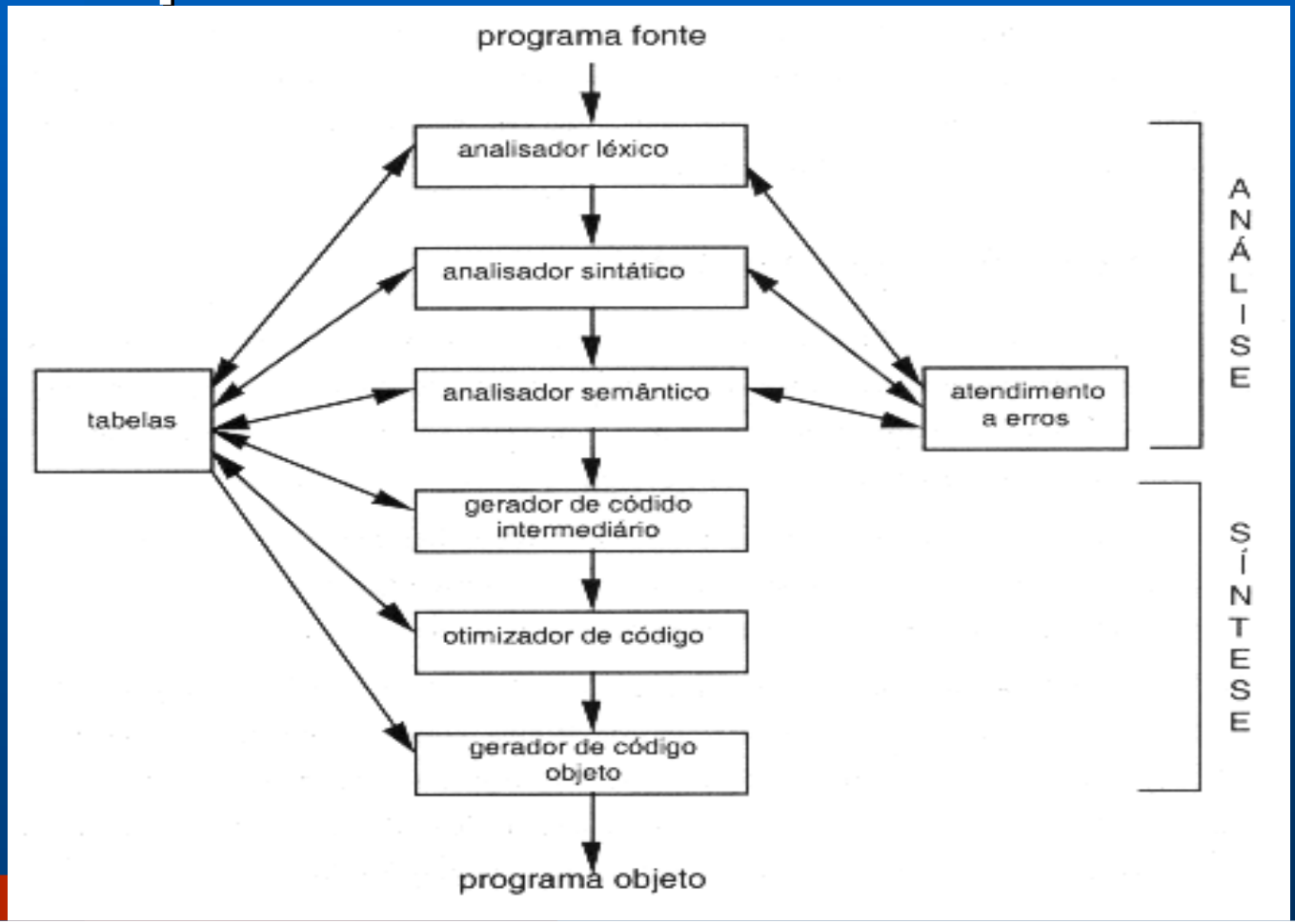
- **Introdução**
- **Fundamentação teórica**
- **Desenvolvimento**
- **Considerações finais**

Introdução

- **Origem/Motivação**
- **Área**
- **Problema**
- **Objetivos**
- **Relevância**

Fundamentação Teórica

- **Compiladores**



Fundamentação Teórica

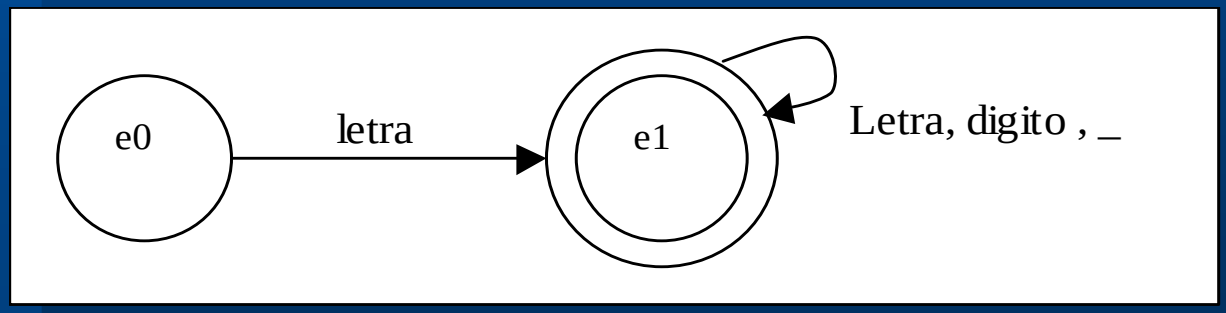
- **Ferramentas para construção de compiladores**
  - **Geradores de analisadores léxicos (Lex)**
  - **Geradores de analisadores sintáticos (Yacc)**
  - **Geradores de geradores de código**

Fundamentação Teórica

- **Analisadores léxicos**
  - **Definições Regulares**

```
letra → a | b | .. | z | A | B | .. | Z
dígito →  0 | 1 | .. | 9
identificador → letra ( letra | dígito | _ )*
```

  - **Autômatos Finitos**



Fundamentação Teórica

- **Glatz (2000)**
  - **Manna (1974)**
  - **Hopcroft (1979)**
  - **Silva (2000)**

Desenvolvimento do Protótipo

- **Especificação**
  - **Meta-Linguagem (Definições Regulares e BNF)**
  - **Diagramas *Use Case* (Especificação e Teste)**
  - **Diagrama de Classes**

Definições Regulares da Meta-Linguagem

letra → a | .. | z | A | .. | Z

dígito → 0 | 1 | .. | 9

identificador → letra (_ | letra | dígito)*

literal → " ASCII+ "

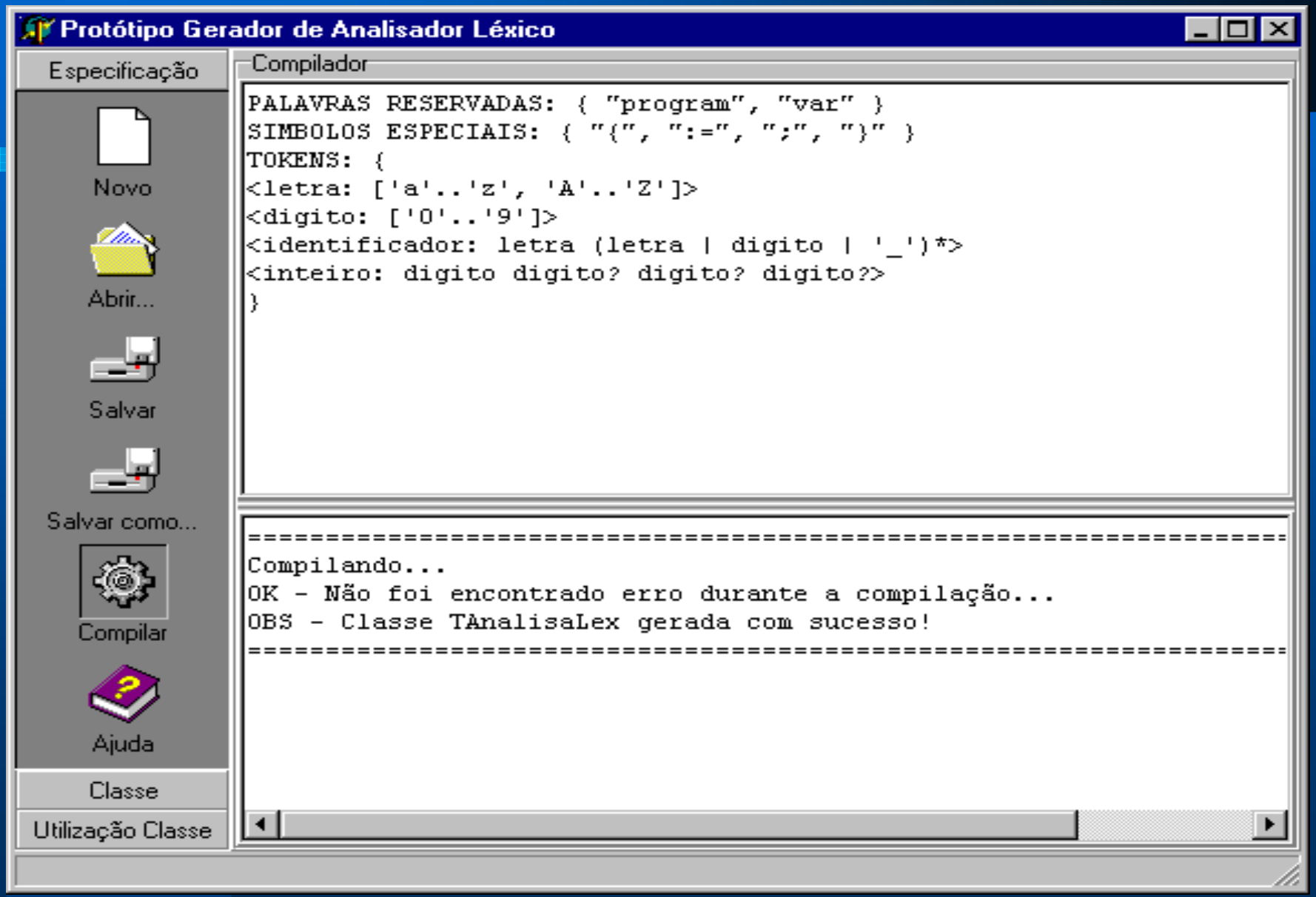
caracter → ' ASCII '

onde ASCII representa todos os caracteres válidos da tabela ASCII, exceto os seguintes caracteres {‘&’, ‘(’, ‘)’, ‘*’, ‘^’, ‘|’}

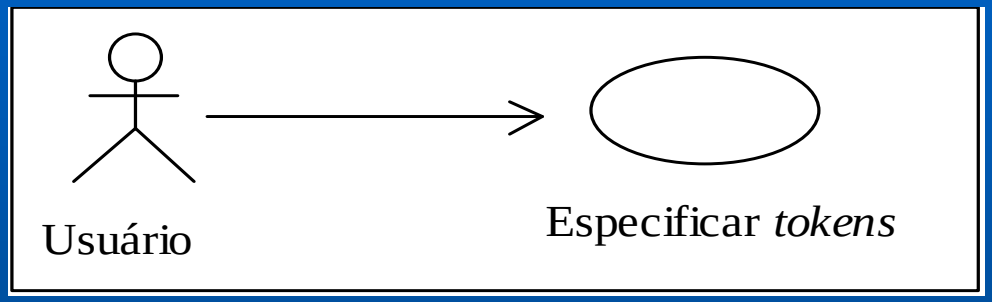
BNF da Meta-Linguagem

<especificação> ::= **PALAVRAS RESERVADAS : <caixa> { <conjunto de literais> }**
SIMBOLOS ESPECIAIS : { <conjunto de literais> }
TOKENS : { <conjunto de tokens> }
<caixa> ::= **(CAIXA SENSÍVEL)** | ε
<conjunto de literais> ::= <lista de literais> | ε
<lista de literais> ::= *literal* | *literal* , <lista de literais>
<conjunto de tokens>::= <token> | <token> <conjunto de tokens>
<token>::= **<** *identificador* **:** <expressão regular> **>**
<expressão regular>::= **(** <expressão regular> **)** <expressão regular'>
| <termo> <expressão regular'>
<expressão regular'>::= <operador> <expressão regular'>
| **|** <expressão regular> | <expressão regular> | ε
<termo>::= **[** <faixa> **]**
| **~ [** <faixa> **]**
| *identificador*
| *literal*
| *caracter*
<operador> ::= **+** | ***** | **?**
<faixa> ::= <sub-faixa> | <sub-faixa>, <faixa>
<sub-faixa> ::= *caracter* | *caracter* .. *caracter*

Exemplo de Especificação de Entrada



Diagramas *Use Case*



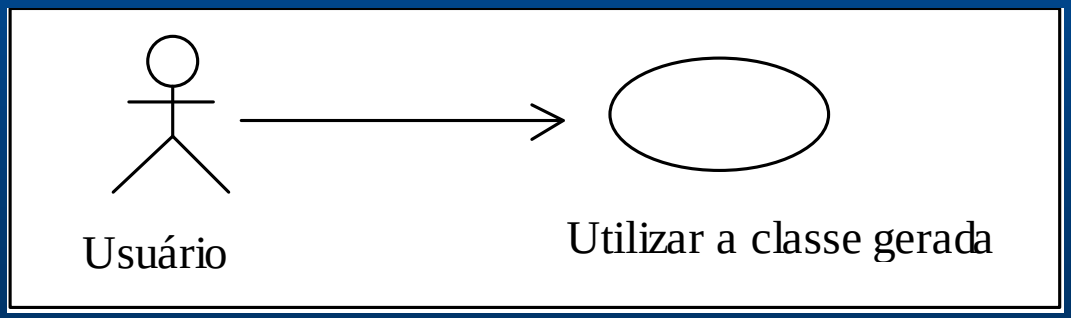
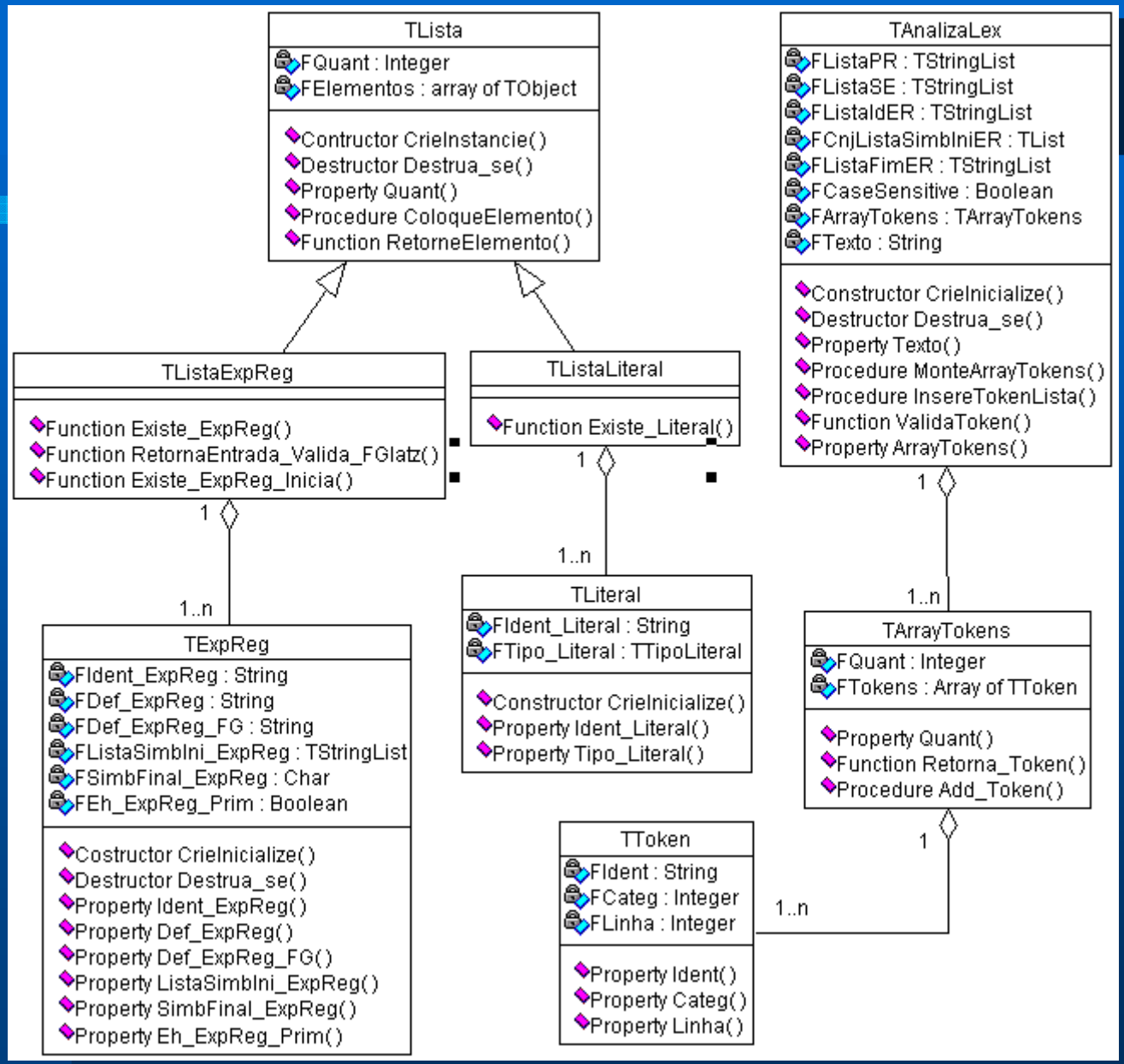


Diagrama de Classes



Desenvolvimento do Protótipo

- **Implementação**
  - **Compilador (Coco/R)**
  - **Algoritmo do Desmonte (Silva, 2000)**
  - **Teste da classe no protótipo (IFPS3)**

Definições Regulares da Meta-Linguagem no Coco/R

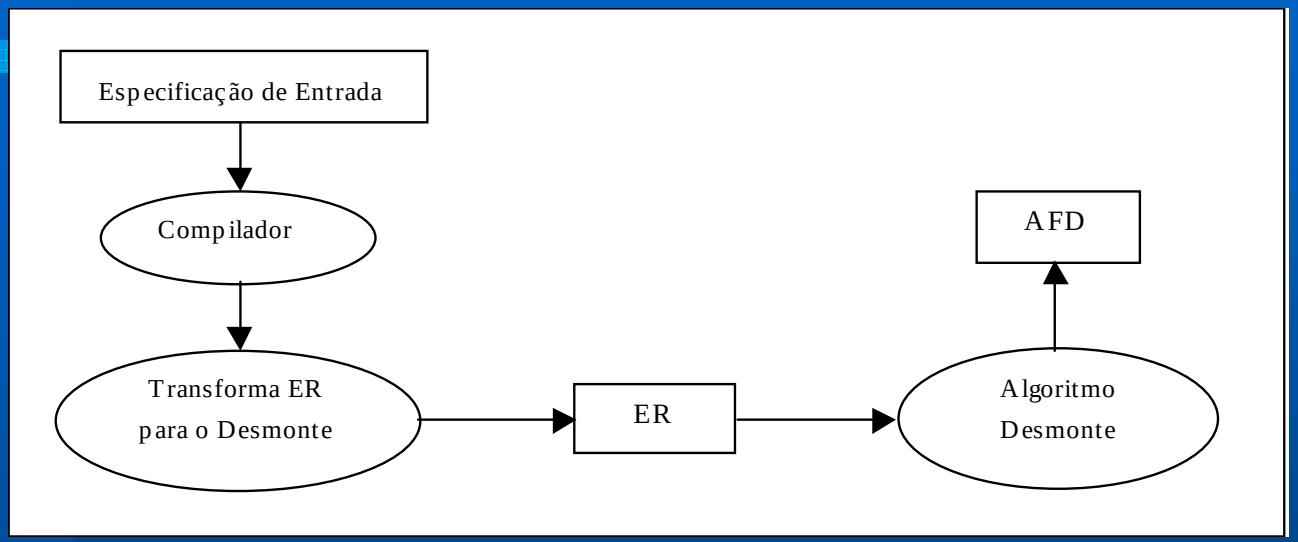
```
IGNORE CASE
CHARACTERS
  Maius   = CHR(65)..CHR(90) .
  Minus   = CHR(97)..CHR(122) .
  Digit   = CHR(48)..CHR(57) .
  Aspa    = CHR(39).
  Aspas   = CHR(34).
  EOL     = CHR(13).
  N1      = CHR(38).  // &
  N2      = CHR(40).  // (
  N3      = CHR(41).  // )
  N4      = CHR(42).  // *
  N5      = CHR(94).  // ^
  N6      = CHR(124). // |
  ASCII   = ANY - EOL.
  Def_Lit = ASCII - Aspas - N1 - N2 - N3 - N4 - N5 - N6.
  Def_Car = ASCII - N1 - N2 - N3 - N4 - N5 - N6.
TOKENS
  Ident     = Maius | Minus { "_" | Maius | Minus | Digit}.
  Literal   = Aspas Def_Lit {Def_Lit} Aspas .
  Caracter = Aspa Def_Car Aspa.
COMMENTS FROM "//" TO EOL
COMMENTS FROM "/*" TO "*/" NESTED
IGNORE CHR(1)..CHR(32)
```

Sintax da Meta-Linguagem no Coco/R

```
Meta_Linguagem =                                (. Inicializar; .)
                 "PALAVRAS"
                 "RESERVADAS"
                 ":"
                 { "("
                   "CAIXA"
                   "SENSIVEL"
                   ")"                          (. CaseSensitive := True; .)
                 }
                 "{"                            (. TipoLiteral := tPR; .)
                 [Cnj_Literal]
                 "}"
                 "SIMBOLOS"
                 "ESPECIAIS"
                 ":"
                 "{"                            (. TipoLiteral := tSE; .)
                 [Cnj_Literal]
                 "}"
                 "TOKENS"
                 ":"
                 "{"
                 Cnj_Token
                 "}"
                                                (. Finalizar; .)
(...)
```

Algoritmo Desmonte

Fonte: Glatz 2000

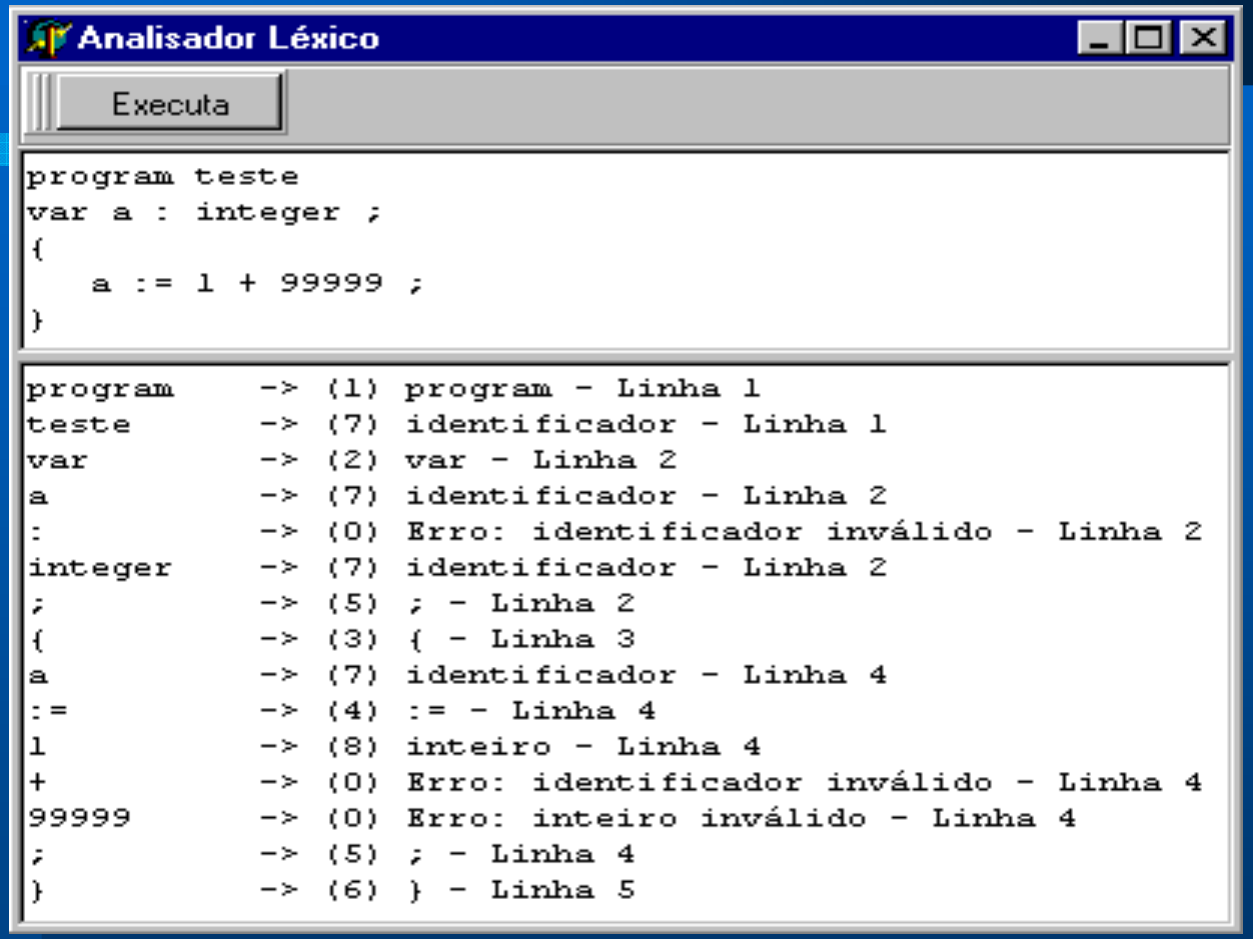


OPERADOR OU NOTAÇÃO	EXEMPLO DE ESPECIFICAÇÃO DE ENTRADA	ESPECIFICAÇÃO TRANSFORMADA
literal	"abc"	abc
caracter	'a'	a
+	a +	(a a*)
?	a ?	(a | ^)
~	~ ['#']	(! | " | $ | .. todos os caracteres da tabela ASCII) conforme descrito na tabela 2.1, menos o caracter '#'
[ ]	['a'..'z']	(a | b | c | . . | z)

Utilização da classe final

- Criar um novo projeto em Delphi
- Adicionar as units das classes ao projeto
- Declarar, instanciar e utilizar os objetos das classes TAnalisaLex,TArrayTokens e TToken

Novo projeto em Delphi



Código do botão Executa

```
procedure TFrmTesteClasse.BitBtnExecAnalisaLexClick(Sender: TObject);
var
  i : Integer;
  Seta, Ident, DesCateg : String;
  NumCateg, Linha : Integer;
begin
  Memo2.Clear;
  vAnalisaLex.Texto := Memo1.Text;
  vAnalisaLex.MonteArrayTokens;
  for i := 1 to vAnalisaLex.ArrayTokens.Quant do
    begin
      Ident := TToken(vAnalisaLex.ArrayTokens.RetornaToken(i)).Ident;
      DesCateg :=
TToken(vAnalisaLex.ArrayTokens.RetornaToken(i)).DesCateg;
      NumCateg :=
TToken(vAnalisaLex.ArrayTokens.RetornaToken(i)).NumCateg;
      Linha := TToken(vAnalisaLex.ArrayTokens.RetornaToken(i)).Linha;
      Seta := '          ->';
      Seta := Copy(Seta, Length(Ident), Length(Seta) - Length(Ident) + 1);
      if NumCateg <> 0 then
        Memo2.Lines.Add(Ident+Seta+' ('+IntToStr(NumCateg)+') '+DesCateg+'
- Linha '+IntToStr(Linha))
      else
        Memo2.Lines.Add(Ident+Seta+' ('+IntToStr(NumCateg)+') Erro:
'+DesCateg+' inválido - Linha '+IntToStr(Linha));
    end;
end;
end.
```

Considerações Finais

- Resultados obtidos
- Potencial e utilização
- Dificuldades encontradas:
  - Coco/R e IFPS3
- Restrições do protótipo:
  - Símbolos Especiais
  - Início das Definições Regulares
  - Caracters ‘&’, ‘|’, ‘*’, ‘^’, ‘(’ e ‘)’
- Extensões