

**LINGUAGENS FORMAIS:**

Especificação e Implementação de um Analisador Léxico-Sintático

Rafael Silveira, Pablo Margreff e Letícia Sampaio

{rsilveira,pmargreff,lsampaio}@inf.ufpel.edu.br

**1. INTRODUÇÃO**

O Análisador léxico é o programa encarregado do processo de analisar a entrada de caracteres e produzir uma seqüência de símbolos chamado "símbolos léxicos" (tokens), que podem ser manipulados mais facilmente pelo analisador sintático.

Já o Analisador Sintático (*parsing*) é o programa encarregado do processo de analisar uma sequência de simbolos de entrada (obtidos do léxico) para determinar sua estrutura gramatical segundo uma determinada gramática formal. Essa análise faz parte de um compilador, junto com o análisador léxico e semântico.

**2. METODOLOGIA**

Primeiramente, foi desenvolvido o autômato finito com uso da ferramenta Jflap, o qual especifica o analisador léxico de acordo com a linguagem solicitada para este trabalho. Logo após, foi escolhida a linguagem de programação Python1, por conta de sua alta abstração e baixa curva de aprendizagem. Após a escolha da linguagem de programação, foi transcrito todo o autômato finito para código, com auxílio da ferramenta de controle de versão Git.

Ao fim do desenvolvimento da primeira parte do trabalho, inciou-se os testes com diversos tipos de entradas de dados. A segunda parte do trabalho, foi desenvolver a gramática livre de contexto, que é a principal parte do analisador sintático. Por fim, foram integrados e testados os códigos das duas partes do trabalho e criada uma interface gráfica, ilustrado na Figura 1, Figura 2 e Figura 3.

[[1]](#footnote-0)

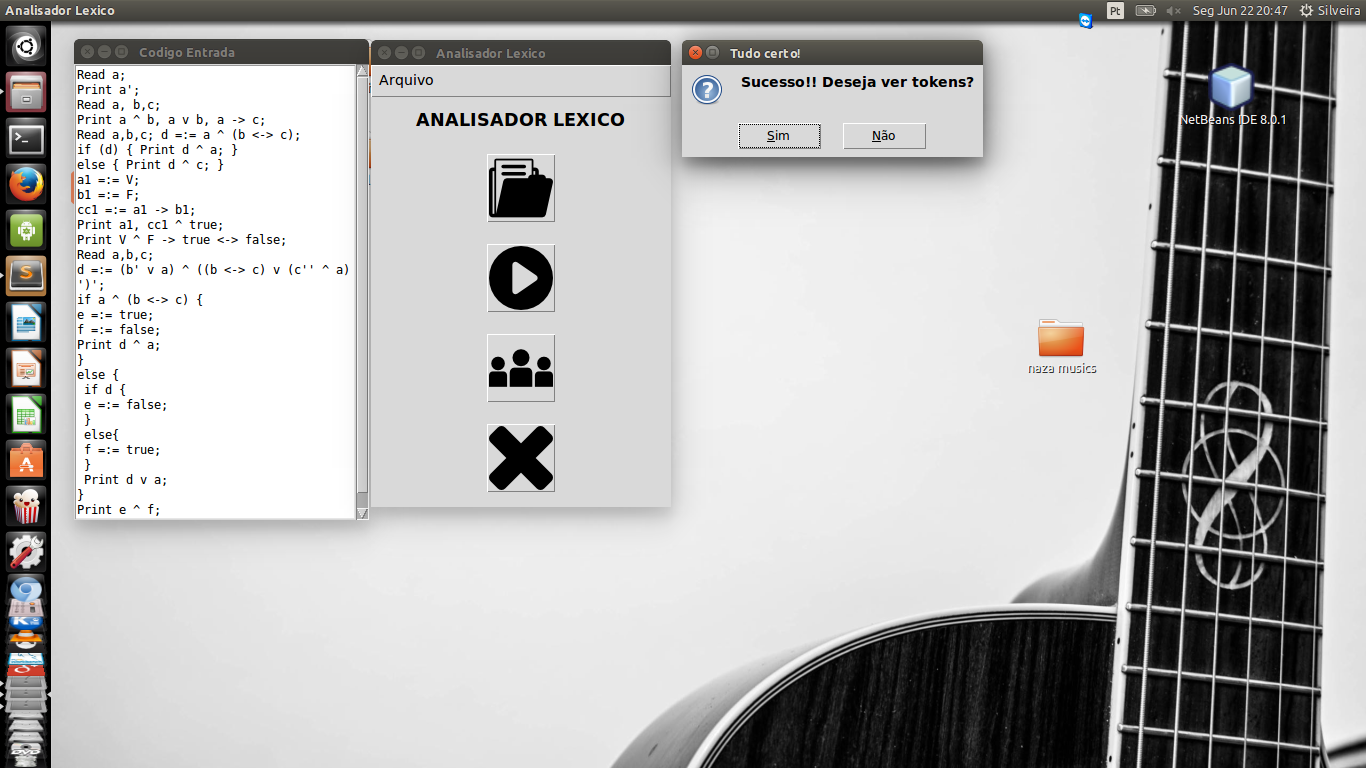


Figura 1: Analisador Léxico Sintático

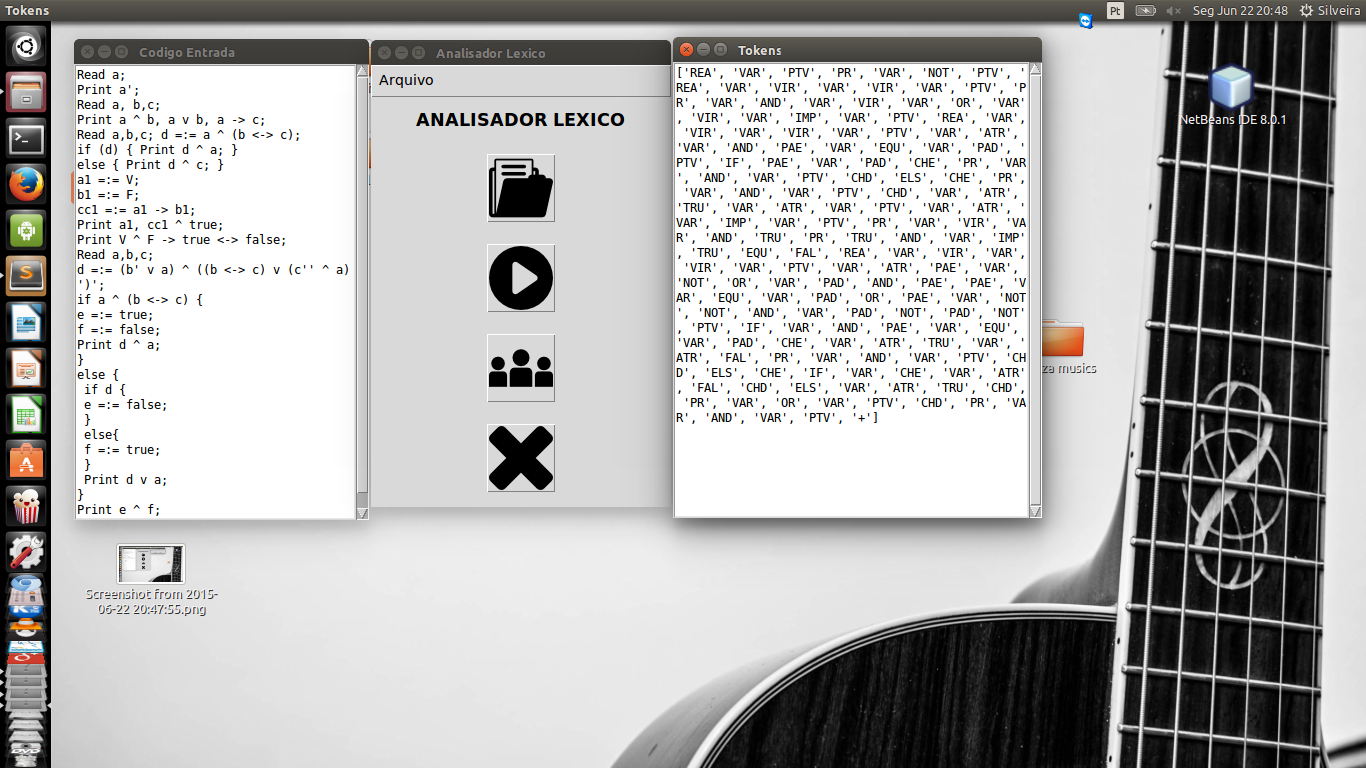


Figura 2: Analisador Léxico Sintático e Tokens

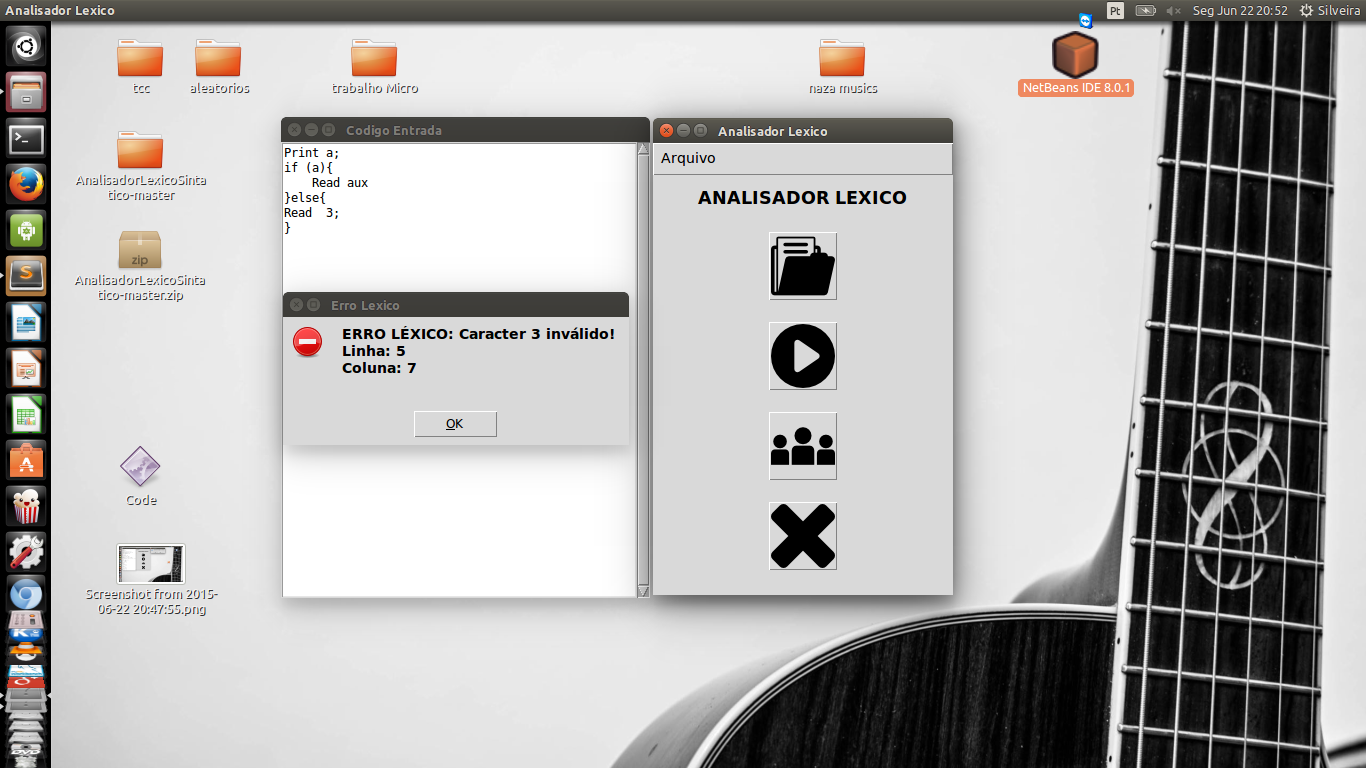
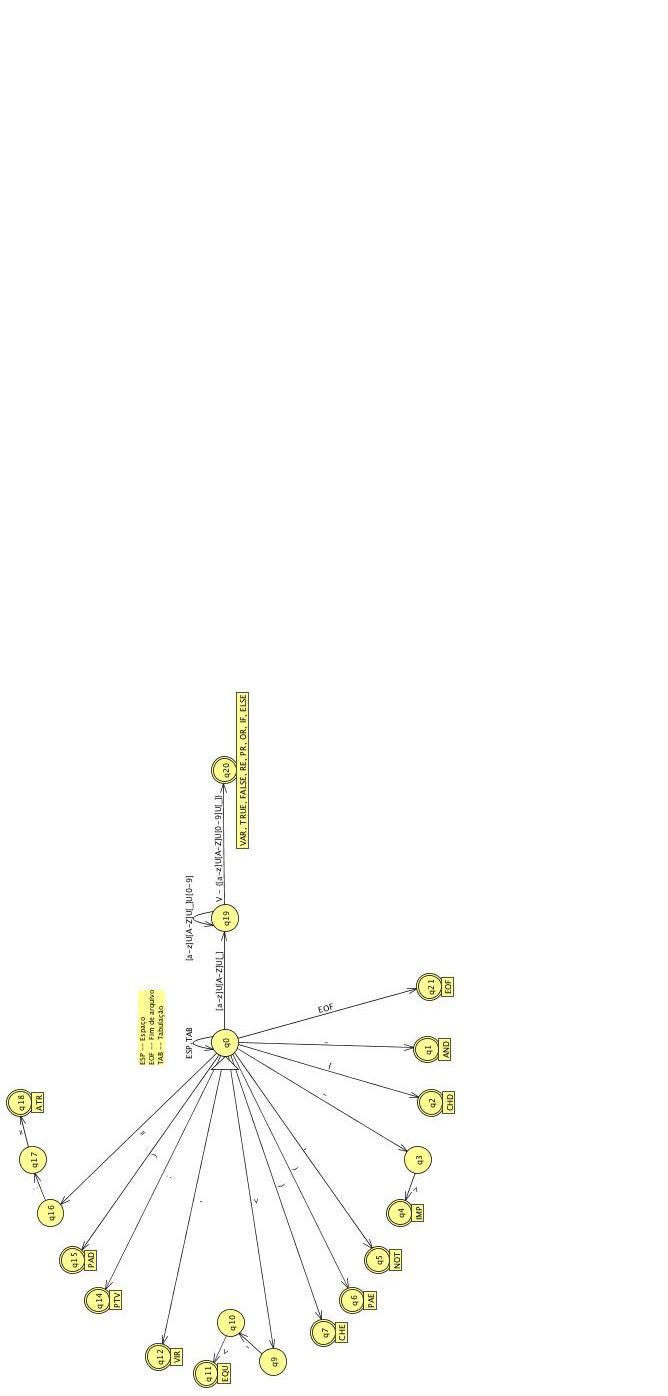


Figura 2: Analisador Léxico Sintático acusando erro

**3. AUTÔMATO FINITO**



**3. TABELA DE EQUIVALÊNCIA**

|  |  |  |
| --- | --- | --- |
| **SÍMBOLO** | **SIGNIFICADO** | **TOKEN** |
| ‘ | Negação | NOT |
| ^ | Conjunção | AND |
| v | Disjunção | OR |
| -> | Implicação | IMP |
| <-> | Equivalência | EQU |
| =:= | Atribuição | ATR |
| , | Vírgula | VIR |
| { | Chave Esquerda | CHE |
| } | Chave Direita | CHD |
| Print | Print | PRI |
| Read | Read | REA |
| EOF | Fim de Arquivo | EOF |
| ( | Parênteses Esquerdo | PAE |
| ) | Parênteses Direito | PAD |
| ; | Ponto e Vírgula | PTV |
| [0] U [F] U [False] | Falso | FAL |
| [1] U [T] U [True] | Verdadeiro | TRU |
| [a-z] U [A-Z] U [0-9] | Variável | VAR |

**4. GRAMÁTICA LIVRE DE CONTEXTO**

Nesta Seção, poderá ser observado passo a passo o desenvolvimento da gramática deste trabalho.

**4.1 GRAMÁTICA INICIAL**

G = (A,B,P,S)

A = {S, E0, E1, E2, A }.

B = { VAR, NOT, AND, OR, IMP, EQU, ATR, VIR, CHE, CHD, PRI, REA, EOF, PAE, PAD, PTV, FAL, TRU, IF, ELS }.

P={

S → <PRI> | E0 <PTV> | <PRI> E0 <EOF>

E0 → E1 | E0 <VIR> E0

E1 → <PAE> E1 <PAD> | E1 <AND> E1 | E1 <OR> E1 | E1 <IMP> E1 | E1 <EQU> E1 | E1 <NOT> | E2

E2 → <TRU> | <FAL> | <VAR>

S → <VAR> <ATR> E1 <PTV> | <VAR> <ATR> E1 <EOF>

S → <REA> A <PTV> | <REA> A <EOF>

A → <VIR> <VAR> | <VIR> <VAR> A

S → <IF> <CHE> S <CHD> <ELS> <CHE> S <CHD>

}

**4.2 GRAMÁTICA APÓS ELIMINAÇÃO DA AMBIGUIDADE**

G = (A,B,P,S)

A = {S, E0, E1, E2, E3, E4, FINAL }.

B = { VAR, NOT, AND, OR, IMP, EQU, ATR, VIR, CHE, CHD, PRI, REA, EOF, PAE, PAD, PTV, FAL, TRU, IF, ELS }.

P={

S → PRINT | VAR | READ | S

READ → <REA> E0

PRINT → <PRI> E0

VAR → <VAR> <ATR> E1

E0 → E0 <VIR> E1 | E1

E0\* → E0\* <VIR> E0\*

S\* →<IF> <CHE> S <CHD> <ELS> <CHE> S <CHD>

E1 → E1 <EQU> E2 | E2

E2 → E2 <IMP> E3 | E3

E3 → E3 <AND> E4 | E3 <OR> E4 | E4

E4 → E4 | <PAE> E4 <PAD> | FINAL

FINAL → <VAR> | <TRU> | <FALS>

}

**4.2 GRAMÁTICA APÓS ELIMINAÇÃO DE RECURSÃO À ESQUERDA**

G = (A,B,P,S)

A = {S, S’, E0, E1, E1\*, E2, E2\*, E3, E3\*, E4, E4\*, FINAL }.

B = { VAR, NOT, AND, OR, IMP, EQU, ATR, VIR, CHE, CHD, PRI, REA, EOF, PAE, PAD, PTV, FAL, TRU, IF, ELS }.

P={

S` → S <EOF>

S → S PRINT | READ | <IF> <PAE> E1 <PAD> <CHE> S <CHD> <ELS> <CHE> S <CHD> | VAR

READ → <REA> E1

PRINT → <PRI> E0

VAR → <VAR> <ATR> E1

E0 → E1

E0 → E1 E0\*

E0\* → <VIR> E1 E0\* | <VIR> E1

E1 → E2 | E2 E1\*

E1\* → <EQU> E2 E1\* | <EQU> E2

E2 → E2 | E3 E2\*

E2\* → <IMP> E3 E2\* | <IMP> E3

E3 → E4 | E4 E3\*

E3\* → <AND> E4 E3\* | <AND> E4 | <OR> E4 E3\* | <OR> E4

E4 → FINAL | <PAE> E1 <PAD> | FINAL E4\* | <OR> E4 | <PAE> E1 <PAD> E4\*

E4\* → NOT E4\* | NOT

FINAL → <VAR> | <TRU> | <FALS>

}

**4.3 GRAMÁTICA APÓS FATORAÇÃO**

G = (A,B,P,S’).

A = {S, S’, PRINT, READ, VAR, IF, E0, E0`, E0\*, E1, E1`, E1\*, E2, E2`, E2\*, E3, E3’, E3\*, E4, E4`, E4\*, FINAL }.

B = { VAR, NOT, AND, OR, IMP, EQU, ATR, VIR, CHE, CHD, PRI, REA, EOF, PAE, PAD, PTV, FAL, TRU, IF, ELS }.

P= {

S` → S <EOF>

S → PRINT | READ | VAR | IF

PRINT→ <PR> E1 <PTV> S

READ→ <REA> E0 <PTV> S

VAR→ <VAR> <ATR> E1 <PTV> S

E0→ E1E0`

E0`→ E0\* | ⋲

E0\*→ <VIR> E1 E0`

E1→ E2 E1`

E1`→ E1\* | ⋲

E1\*→ <EQU> E2 E1`

E2→ E3 E2`

E2`→ E2\* | ⋲

E2\*→ <IMP> E3 E2`

E3→ E4 E3`

E3`→ E3\* | ⋲

E3\*→ <AND> E4 E3` | <OR> E4 E3`

E4→ FINAL E4` | <PAE> E1 <PAD> E4`

E4`→ E4\* | ⋲

E4\*→ <NOT> E4`

FINAL→ <VAR> | <FAL> | <TRU>

IF→ <IF> E1 <CHE> S <CHD> IF`

IF`→ <ELS> <CHE> S <CHD> S | S

}

1. ***Python*** é uma linguagem de programação de alto nível,, interpretada, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991.Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. [↑](#footnote-ref-0)