UNIVERSIDADE FEDERAL
DE SANTA CATARINA

UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO - CTC

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA - INE

INE5622 - Introdução a Compiladores

ARTHUR NASCIMENTO MOTA
ERIC DE COL SALES
VITOR DE SOUZA CUNHA

Projeto Analisador Léxico - LCC

FLORIANÓPOLIS
2021

Projeto Analisador Léxico - LCC

Neste projeto foi utilizada a ferramenta ANTLR4 para a construção do analisador léxico da linguagem LCC-2021-1 e python 3.8 para a sua execução.

O ANTLR4 (Another Tool for Language Recognition) é um *parser generator* para leitura, processamento, execução e tradução de binários e texto estruturado. É largamente utilizado para a construção de linguagens, ferramentas e frameworks.

Esta ferramenta foi utilizada dada a sua documentação detalhada e conjunto de ferramentas que iam de encontro com a finalidade deste trabalho. Há outras excelentes ferramentas como LEX, FLEX, BISON e YACC, porém esta se mostrou mais interessante ao grupo.

A partir da criação de um arquivo *.g4* (extensão dos arquivos do ANTLR4), inserimos os *tokens* que serão utilizados na nossa linguagem e as definições regulares (mais detalhes abaixo). Após a criação dos ***tokens***, criamos as ***rules***, que são as regras que compõem a nossa linguagem (*funclist, funcdef, statement)*. Com os *tokens* e *rules* criados, definimos uma regra *start,* que será a regra executada no nosso arquivo principal e contém todas as outras regras.

Para o funcionamento completo da linguagem, chamamos o comando ***antlr4 -Dlanguage=Python3 LCC.g4,*** com isso o ANTLR4 gera os arquivos Lexer e Parser, que são os arquivos que serão importados dentro do nosso ***__main__.py,*** onde nós incluímos a lógica de receber uma string com o caminho do nosso código de exemplo, e com esse código utilizar o analisador léxico criado para, se não houver erros léxicos, retornar a lista de tokens (na mesma ordem em que eles ocorrem no arquivo de entrada de dados) e uma tabela de símbolos, e se houver erros léxicos, retornar uma mensagem de erro informando a coluna e linha onde ocorreu.

Exemplo de entrada e saída **com erros léxicos**:

```
1   def func1+(int A, int B)
2   {
3     int SM;
4     SM = A + B;
5     SM = B * C;
6     return;
7   }
8
9   def {}
10
```

```
line 1:9 extraneous input '+' expecting '('
line 9:4 no viable alternative at input 'def{'
```

Exemplo de entrada e saída **sem erros léxicos**:

```
def func1(int A, int B)
{
  int SM[2];
  SM[0] = A + B;
  SM[1] = B * C;
  return;
}

def principal()
{
  int C;
  int D;
  int R;
  C = 4;
  D = 5;
  R = func1(C, D);
  return;
}
```

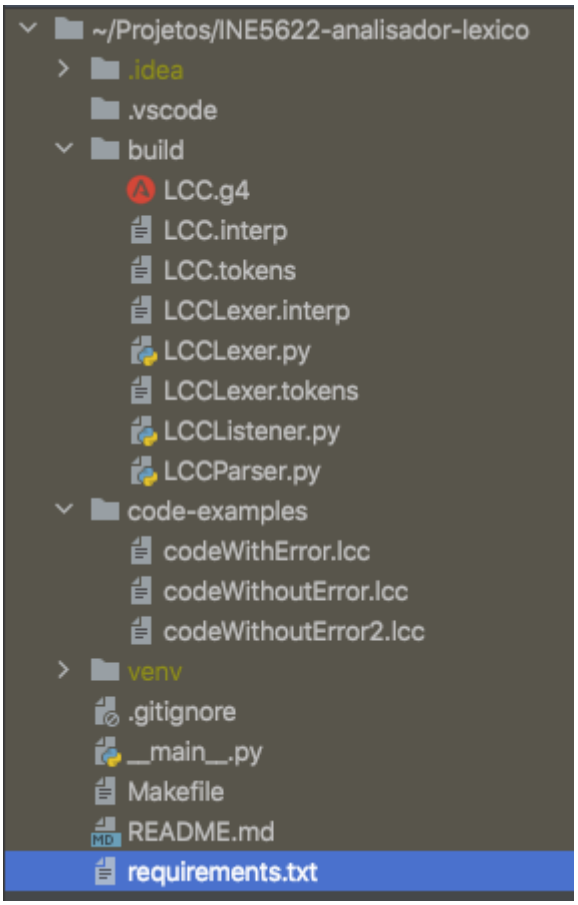
```
./venv/bin/python3 __main__.py ./code-examples/codeWithoutError.lcc
Lista de tokens na ordem que eles ocorren:

['DEF', 'IDENT', 'OPEN_PAREN', 'INT', 'IDENT', 'COMMA', 'INT', 'IDENT', 'CLOSE_PAREN', 'OPEN_BRACE', 'INT', 'IDENT', 'OPEN_BRACK', 'INT_CONSTANT', 'CLOSE_BRACK', 'SEMI_COLON', 'IDENT', 'OPEN_BRACK', 'INT_CONSTANT', 'CLOSE_BRACK', 'A
SSIGN', 'IDENT', 'PLUS', 'IDENT', 'SEMI_COLON', 'IDENT', 'OPEN_BRACK', 'INT_CONSTANT', 'CLOSE_BRACK', 'ASSIGN', 'IDENT', 'MULTIPLY', 'IDENT', 'SEMI_COLON', 'RETURN', 'SEMI_COLON', 'CLOSE_BRACE', 'DEF', 'IDENT', 'OPEN_PAREN', 'CLOSE_
PAREN', 'OPEN_BRACE', 'INT', 'IDENT', 'SEMI_COLON', 'INT', 'IDENT', 'SEMI_COLON', 'INT', 'IDENT', 'SEMI_COLON', 'IDENT', 'ASSIGN', 'INT_CONSTANT', 'SEMI_COLON', 'IDENT', 'ASSIGN', 'INT_CONSTANT', 'SEMI_COLON', 'IDENT', 'ASSIGN', 'ID
ENT', 'OPEN_PAREN', 'IDENT', 'COMMA', 'IDENT', 'CLOSE_PAREN', 'SEMI_COLON', 'RETURN', 'SEMI_COLON', 'CLOSE_BRACE', 'DIGIT']

Tabela de símbolos:

IDENT 'func1' ocorreu pela primeira vez na linha 1 coluna 4
IDENT 'A' ocorreu pela primeira vez na linha 1 coluna 14
IDENT 'B' ocorreu pela primeira vez na linha 1 coluna 21
IDENT 'SM' ocorreu pela primeira vez na linha 3 coluna 6
IDENT 'C' ocorreu pela primeira vez na linha 5 coluna 14
IDENT 'principal' ocorreu pela primeira vez na linha 9 coluna 4
IDENT 'D' ocorreu pela primeira vez na linha 12 coluna 6
IDENT 'R' ocorreu pela primeira vez na linha 13 coluna 6
```

Estrutura do código do projeto:



/build

Onde está localizado o arquivo principal da linguagem (**LCC.g4**) e os arquivos gerados através do ANTLR4.

/code-examples

Onde estão localizados os exemplos de código que utilizamos para testar a linguagem (exemplos que estavam disponíveis no Moodle da disciplina).

__main__.py

Arquivo principal, onde está a lógica que importa o Lexer e Parser, recebe uma entrada que é o caminho de algum arquivo com código que será analisado pelo Analisador Léxico, e a lógica que imprime a lista de tokens e a tabela de símbolos.

As definições regulares produzidas para os tokens foram:

DEF: 'def';
INT: 'int';
FLOAT: 'float';
STRING: 'string';
BREAK: 'break';
PRINT: 'print';
READ: 'read';
RETURN: 'return';
IF: 'if';
ELSE: 'else';
FOR: 'for';
NEW: 'new';
OPEN_PAREN : '(';
CLOSE_PAREN : ')';
OPEN_BRACE : '{';
CLOSE_BRACE : '}';
OPEN_BRACK : '[';
CLOSE_BRACK : ']';
COMMA: ',';
SEMI_COLON : ';';
ASSIGN : '=';
PLUS: '+';
MINUS: '-';
MULTIPLY: '*';
DIVIDE: '/';
MOD: '%';
NULL: 'null';
LESS_THAN: '<';
GREATER_THAN: '>';
LESS_THAN_OR_EQUAL: '<=';
GREATER_THAN_OR_EQUAL: '>=';
EQUALS: '==';
DIFFERENT: '!=';
IDENT : [a-zA-Z][a-zA-Z0-9]*;
INT_CONSTANT: [0-9]+;
FLOAT_CONSTANT: [0-9]?'.'[0-9]*;
STRING_CONSTANT: ""('\\"" | '\\\\'|.)*? "";;
WS : [\r\n\t]+ -> skip;

Diagramas de transição

Diagrama de transição do token **DEF**

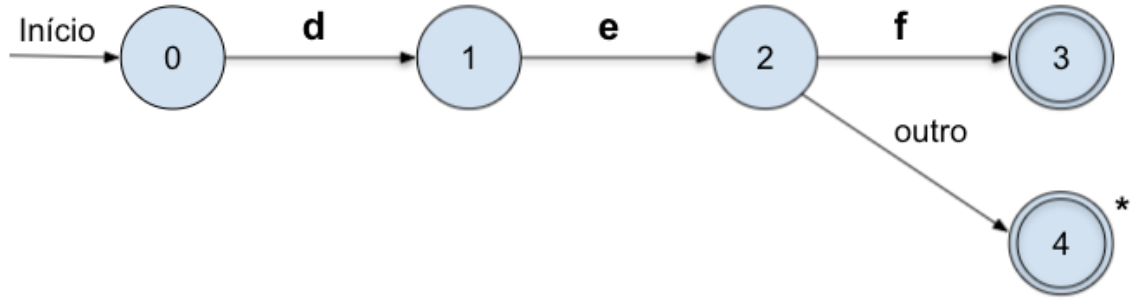


Diagrama de transição do token **INT**

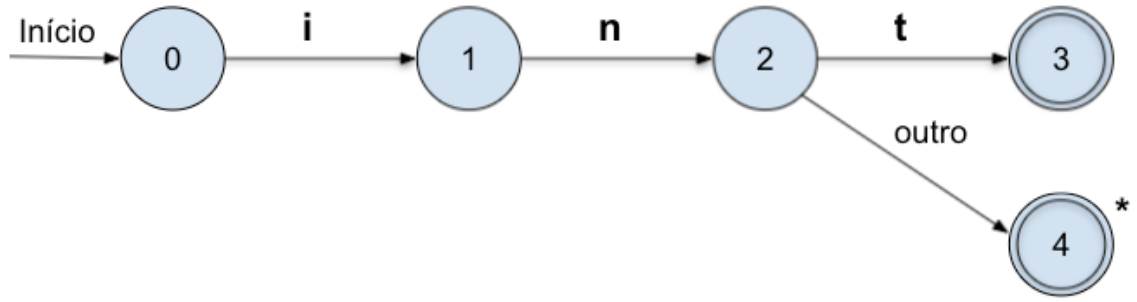


Diagrama de transição do token **FLOAT**

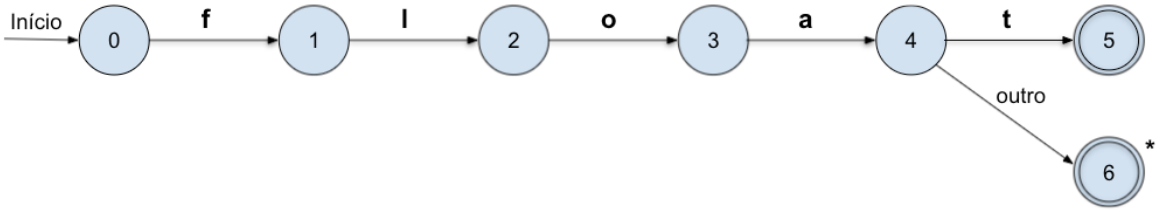


Diagrama de transição do token **STRING**

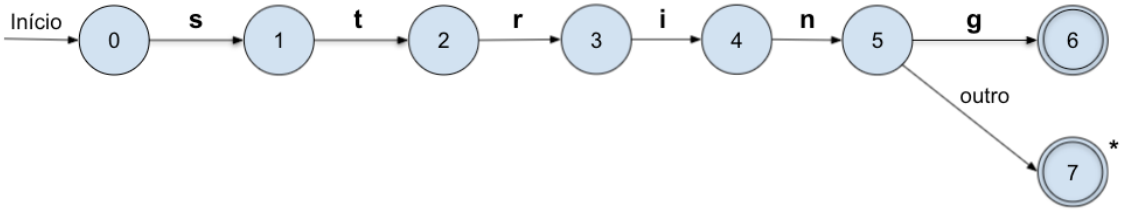


Diagrama de transição do token **BREAK**

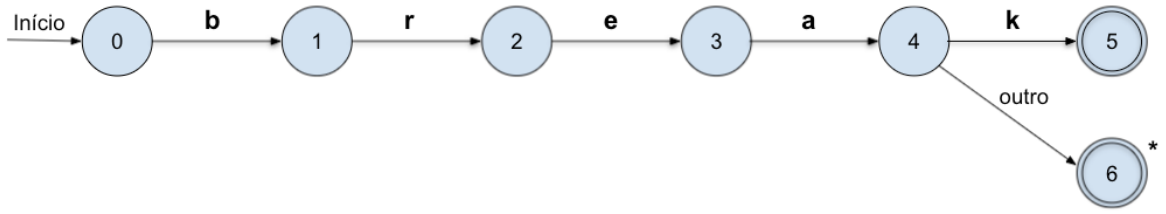


Diagrama de transição do token **PRINT**

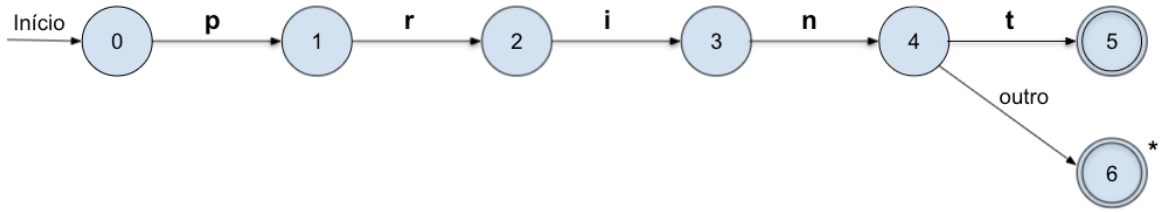


Diagrama de transição do token **READ**

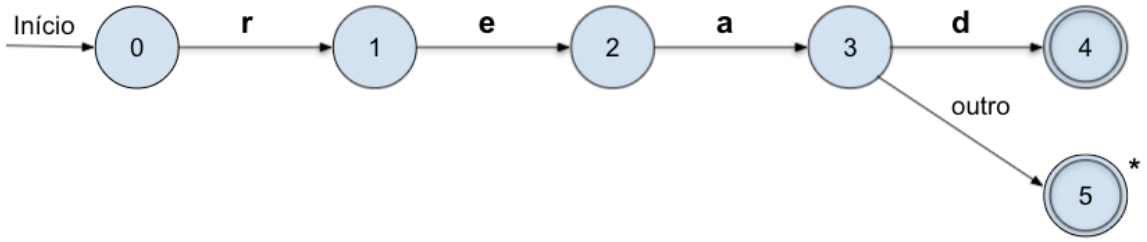


Diagrama de transição do token **RETURN**

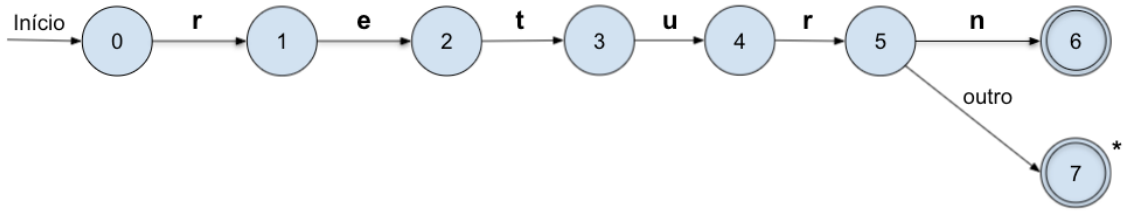


Diagrama de transição do token **IF**

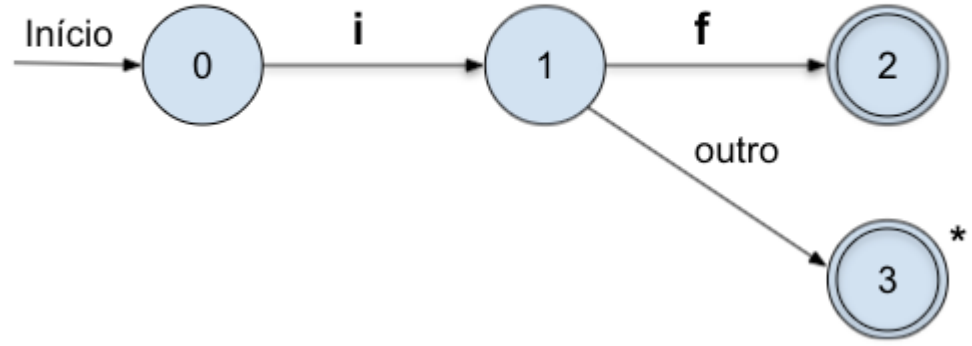


Diagrama de transição do token **ELSE**

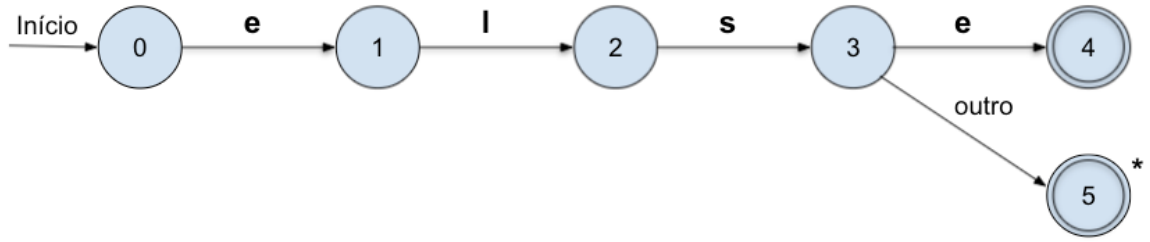


Diagrama de transição do token **FOR**

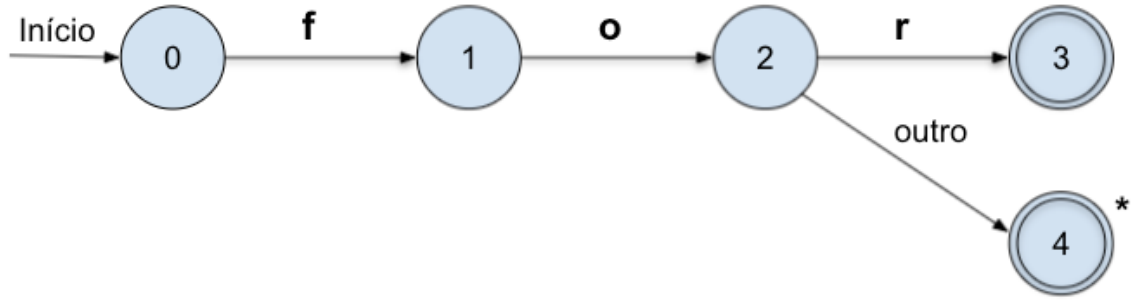


Diagrama de transição do token **NEW**

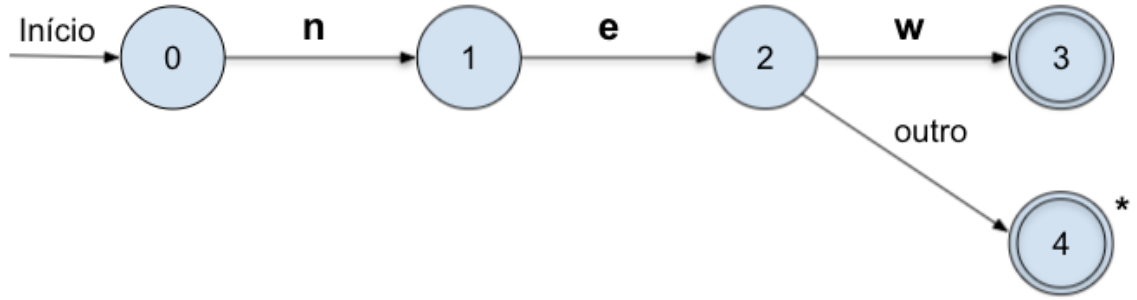


Diagrama de transição do token **OPEN_PAREN**

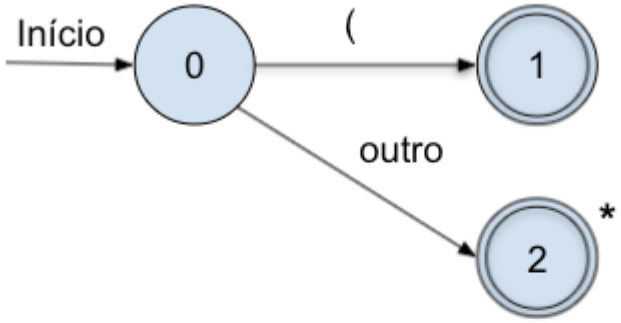


Diagrama de transição do token **CLOSE_PAREN**

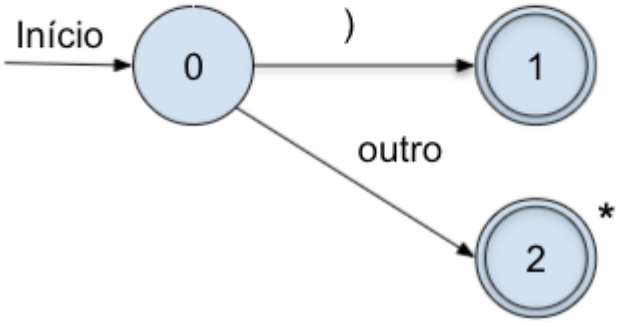


Diagrama de transição do token **OPEN_BRACE**

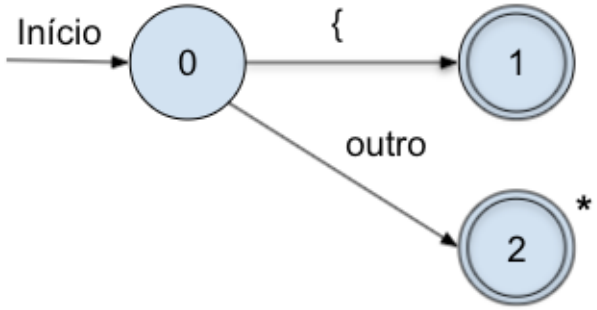


Diagrama de transição do token **CLOSE_BRACE**

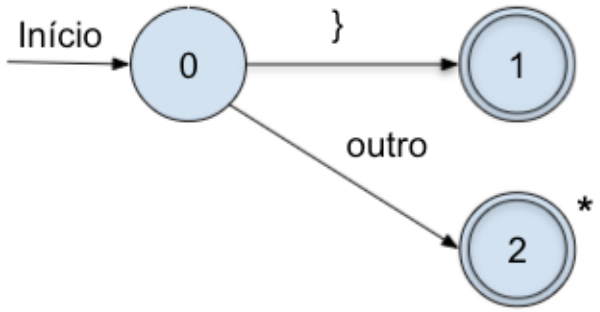


Diagrama de transição do token **OPEN_BRACK**

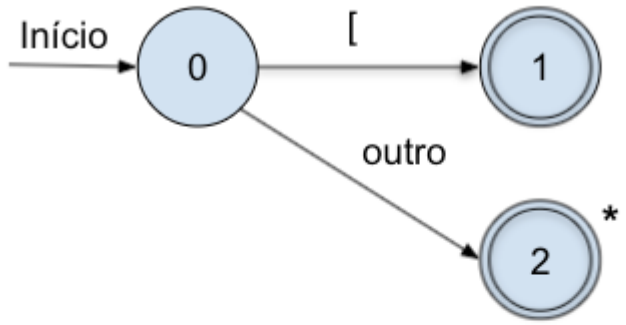


Diagrama de transição do token **CLOSE_BRACK**

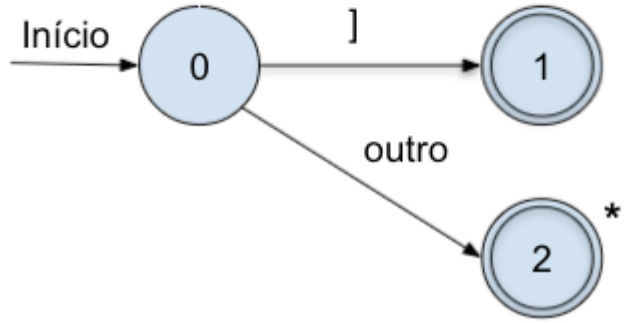


Diagrama de transição do token **COMMA**

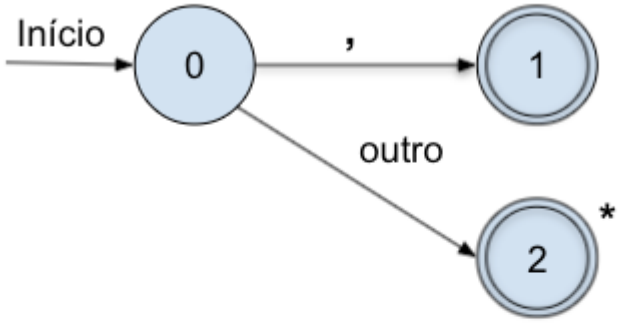


Diagrama de transição do token **SEMI_COLON**

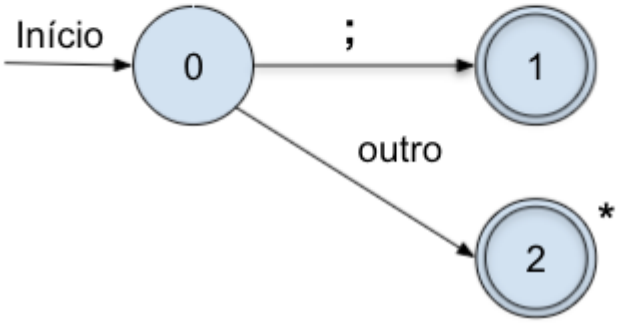


Diagrama de transição do token **ASSIGN**

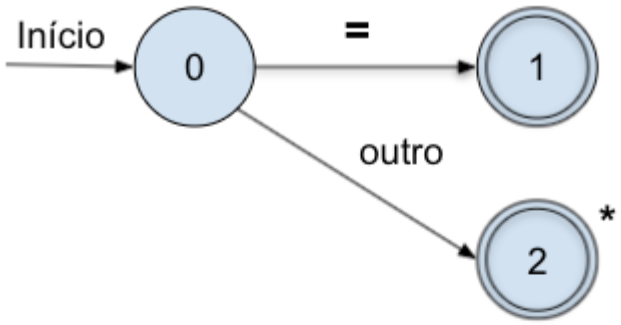


Diagrama de transição do token **PLUS**

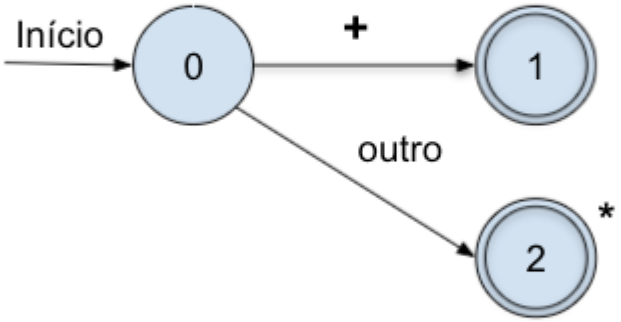


Diagrama de transição do token **MINUS**

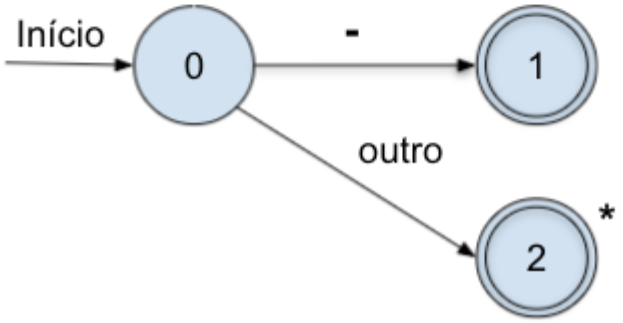


Diagrama de transição do token **MULTIPLY**

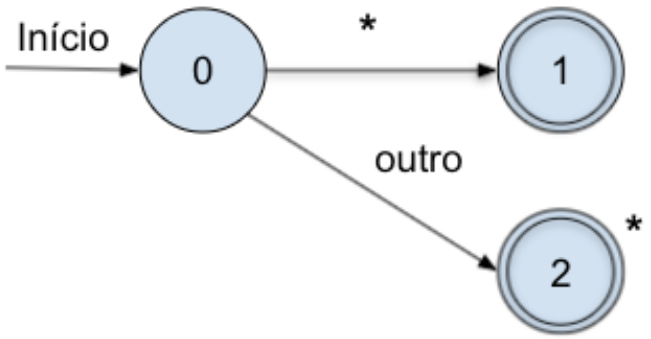


Diagrama de transição do token **DIVIDE**

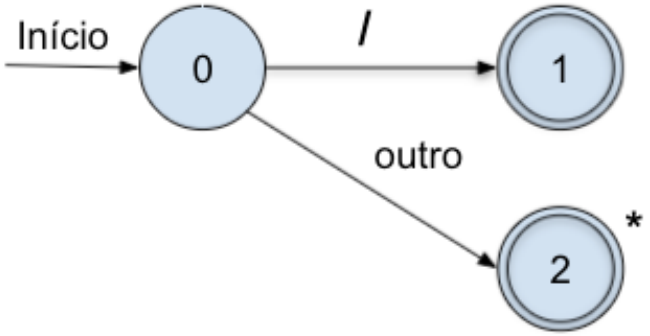


Diagrama de transição do token **MOD**

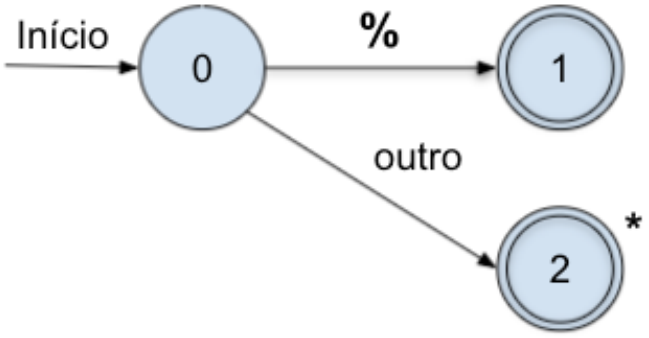


Diagrama de transição do token **LESS_THAN**

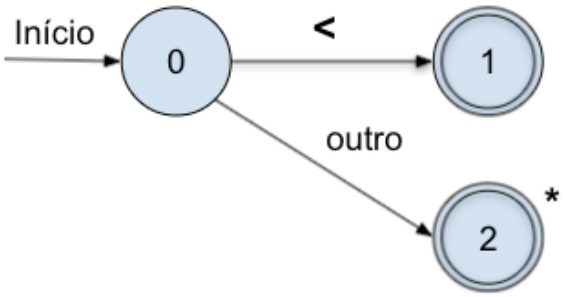


Diagrama de transição do token **GREATER_THAN**

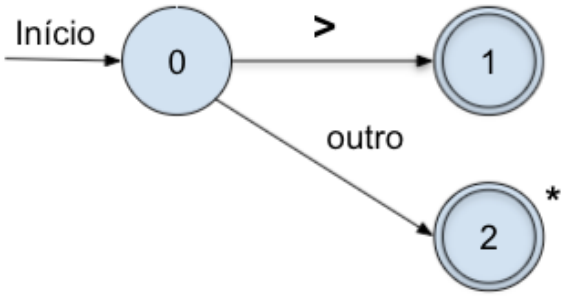


Diagrama de transição do token **LESS_THAN_OR_EQUAL**

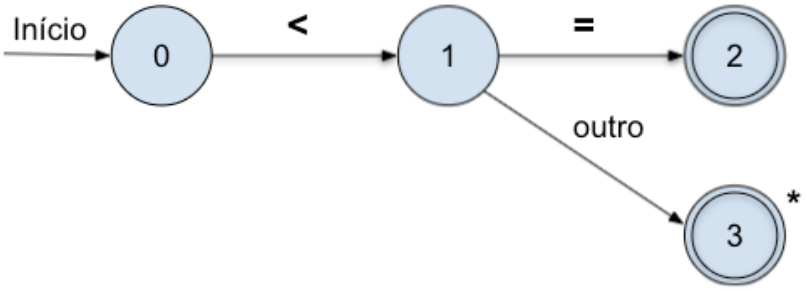


Diagrama de transição do token **GREATER_THAN_OR_EQUAL**

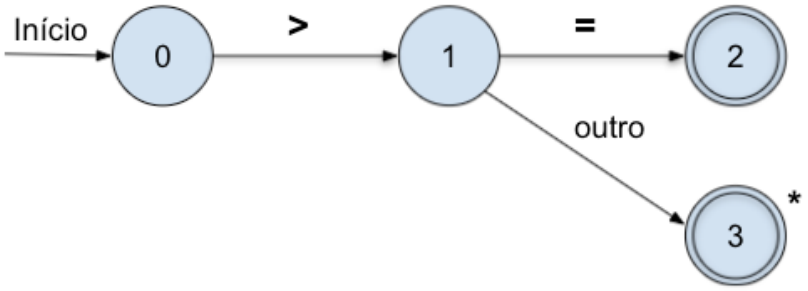


Diagrama de transição do token **EQUALS**

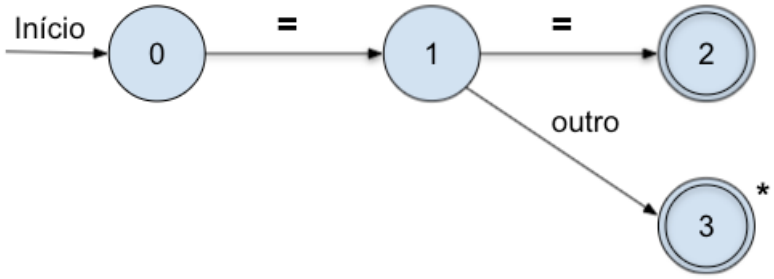


Diagrama de transição do token **DIFFERENT**

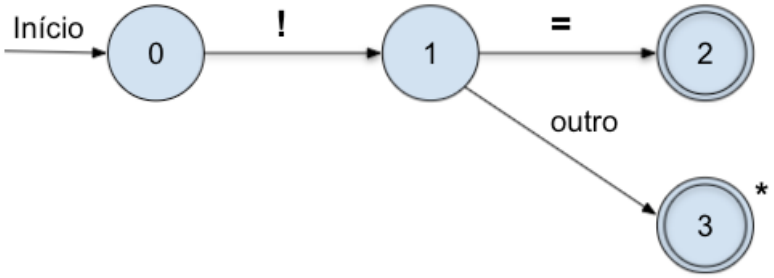


Diagrama de transição do token **NULL**