

D-pression-

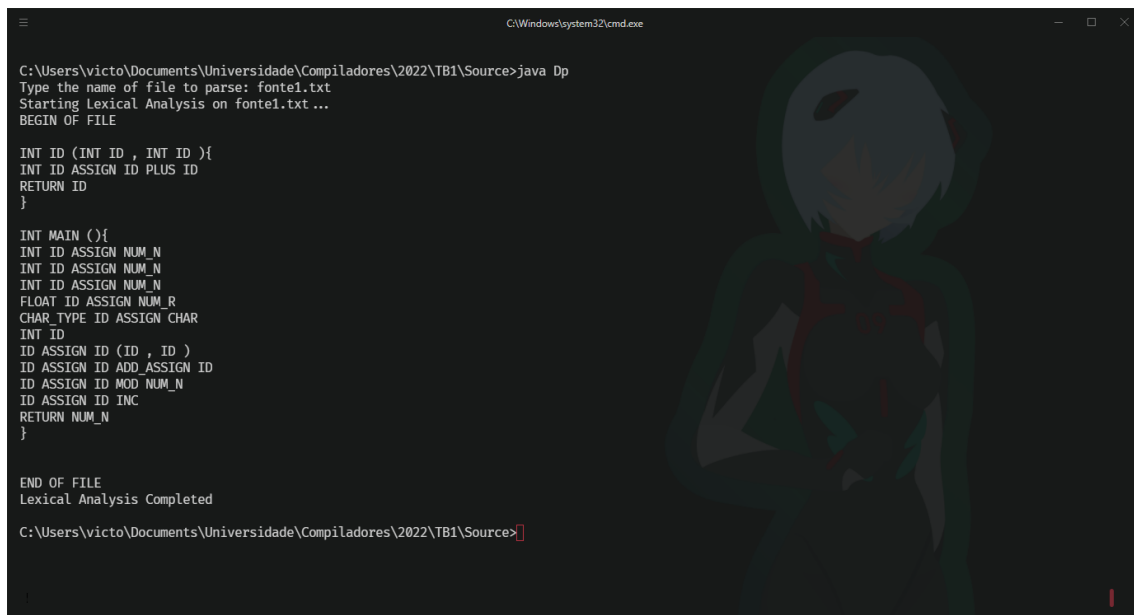
Gustavo G. Queiroz

Victor H. A. Alicino

A linguagem D-expression- é uma pequena linguagem de programação inspirada na linguagem de programação C (de onde vem seu nome curto D).

SOBRE O SOFTWARE

Foi criado a partir do JavaCC (ferramenta para auxílio de algumas fases na criação de compiladores) um analisador léxico para a linguagem que pode detectar erros na escrita do código-fonte. No atual estágio de desenvolvimento a linguagem pode apenas reconhecer os tokens realizando a análise léxica, dessa forma o software atualmente apenas imprime na tela os tipos de tokens encontrados pelo software.



```
C:\Windows\system32\cmd.exe

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>java Dp
Type the name of file to parse: fonte1.txt
Starting Lexical Analysis on fonte1.txt...
BEGIN OF FILE

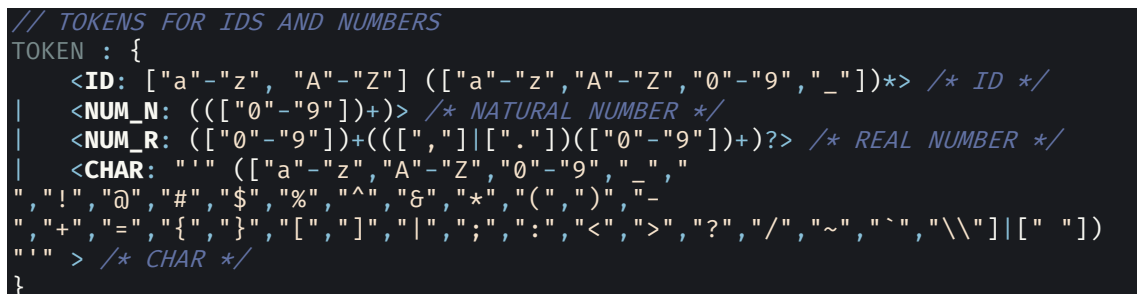
INT ID (INT ID , INT ID ){
INT ID ASSIGN ID PLUS ID
RETURN ID
}

INT MAIN (){
INT ID ASSIGN NUM_N
INT ID ASSIGN NUM_N
INT ID ASSIGN NUM_N
FLOAT ID ASSIGN NUM_R
CHAR_TYPE ID ASSIGN CHAR
INT ID
ID ASSIGN ID (ID , ID )
ID ASSIGN ID ADD_ASSIGN ID
ID ASSIGN ID MOD NUM_N
ID ASSIGN ID INC
RETURN NUM_N
}

END OF FILE
Lexical Analysis Completed

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>
```

Poucos são os que erros podem ser detectados na análise léxica, até por ainda não haver uma regra de como os tokens devem ser combinados, dado este fato o software é capaz de reconhecer dois tipos de erro, caracteres inválidos à linguagem como “@”, “#”, entre outros – devido a expressão regular de constantes numéricas especificado no JavaCC:



```
// TOKENS FOR IDS AND NUMBERS
TOKEN : {
  <ID: ["a"- "z", "A"- "Z"] ([ "a"- "z", "A"- "Z", "0"- "9", "_"])*> /* ID */
  | <NUM_N: (([ "0"- "9" ])+)> /* NATURAL NUMBER */
  | <NUM_R: ([ "0"- "9" ])+([ "." | [ "0"- "9" ])+)> /* REAL NUMBER */
  | <CHAR: "'" ([ "a"- "z", "A"- "Z", "0"- "9", " ", "!", "@", "#", "$", "%", "^", "&", "*", "(", ")", "-", "+", "=", "{", "}", "[", "]", "|", ";", ":", "<", ">", "?", "/", "~", "`", "\\"] | [ " " ])'> /* CHAR */
}
```

Ao encontrar algum erro o software dirá a posição – linha e coluna – e o caractere que ocasionou o erro.

```
C:\Windows\system32\cmd.exe

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>java Dp
Type the name of file to parse: fonte2.txt
Starting Lexical Analysis on fonte2.txt...
BEGIN OF FILE

FLOAT ID ASSIGN NUM_R
INT ID (INT ID , INT ID ){
INT ID ASSIGN ID PLUS ID
RETURN ID
}

INT MAIN (){
INT ID ASSIGN NUM_N
INT ID ASSIGN NUM_N
INT ID ASSIGN NUM_N
FLOAT ID ASSIGN NUM_N
Lexical error at line 12, column 16. Encountered: "#" (35), after : ""
Lexical Analysis Failed

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>
```

SOBRE A COMPILAÇÃO DO SOFTWARE

O software que realiza a análise léxica da linguagem D-pressure- foi gerado a partir do JavaCC, como aqui já citado, o JavaCC gera códigos na linguagem Java a partir de um arquivo de configuração de extensão “.jjt”, nesse arquivo se encontra a definição de tokens da linguagem e como ela deve aceitar algumas combinações de caracteres, é a partir deste arquivo que será compilado o software.

Para transformar o arquivo “.jjt” em código-fonte Java, execute o comando:
`javacc D-pressure-.jjt`

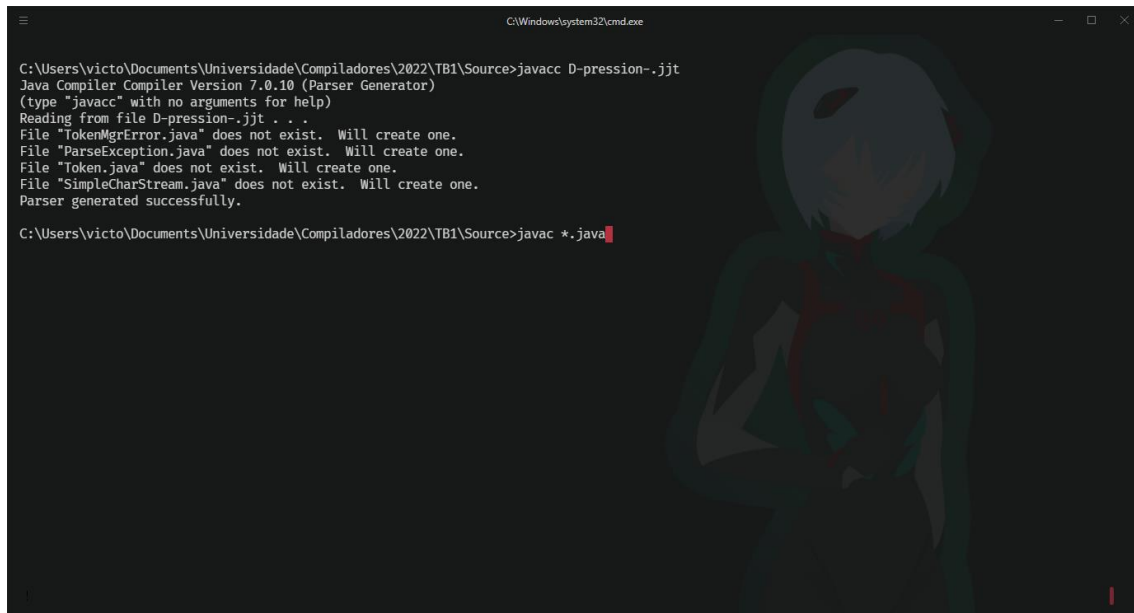
```
C:\Windows\system32\cmd.exe

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>javacc D-pressure-.jjt
Java Compiler Compiler Version 7.0.10 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file D-pressure-.jjt . . .
File "TokenMgrError.java" does not exist. Will create one.
File "ParseException.java" does not exist. Will create one.
File "Token.java" does not exist. Will create one.
File "SimpleCharStream.java" does not exist. Will create one.
Parser generated successfully.

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>
```

Após efetuado esse passo e gerado o código fonte Java, compile ele usando o compilador Java.

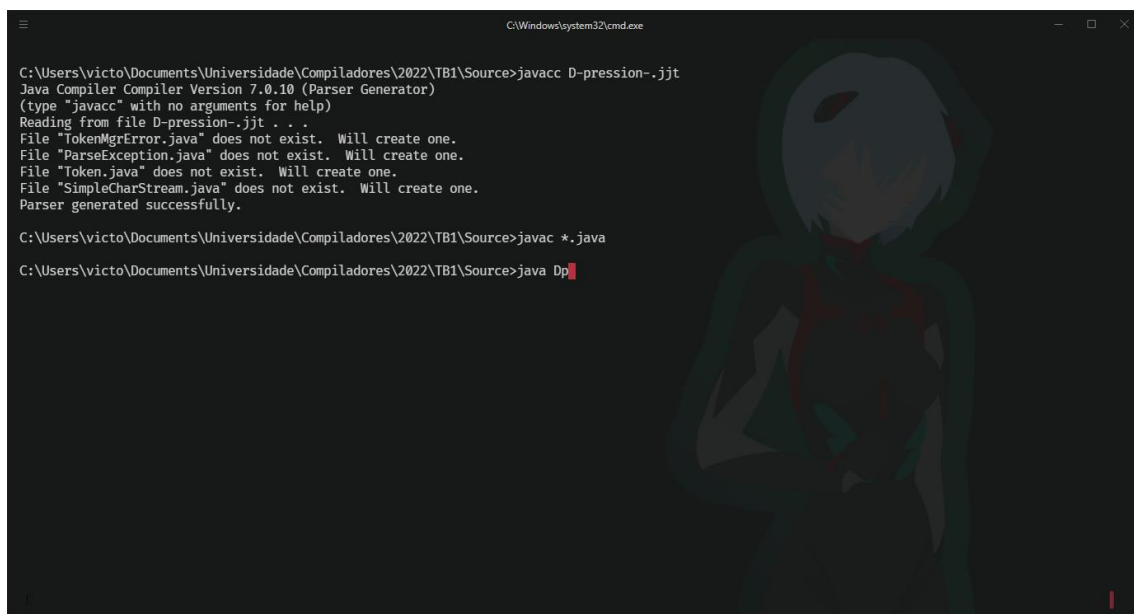
```
javac *.java
```

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the output of the command "javac D-pression-.jjt". The output indicates that the Java Compiler Version 7.0.10 (Parser Generator) was used, and it successfully generated the parser. The command prompt then shows the command "javac *.java" being entered, with a red cursor at the end of the line.

```
C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>javac D-pression-.jjt
Java Compiler Compiler Version 7.0.10 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file D-pression-.jjt . . .
File "TokenMgrError.java" does not exist. Will create one.
File "ParseException.java" does not exist. Will create one.
File "Token.java" does not exist. Will create one.
File "SimpleCharStream.java" does not exist. Will create one.
Parser generated successfully.

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>javac *.java
```

Com o código Java já compilado basta apenas executar o software, que se da através do comando `java Dp`

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the output of the command "javac D-pression-.jjt", which is identical to the previous screenshot. Below that, the command "javac *.java" is shown. At the bottom, the command "java Dp" is entered, with a red cursor at the end of the line.

```
C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>javac D-pression-.jjt
Java Compiler Compiler Version 7.0.10 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file D-pression-.jjt . . .
File "TokenMgrError.java" does not exist. Will create one.
File "ParseException.java" does not exist. Will create one.
File "Token.java" does not exist. Will create one.
File "SimpleCharStream.java" does not exist. Will create one.
Parser generated successfully.

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>javac *.java
C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>java Dp
```

O software esperará de entrada o caminho do arquivo fonte que deseja se executar a análise léxica, se um caminho inválido for escolhido o software encerrará.

```
C:\Windows\system32\cmd.exe

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>javacc D-pression-.jjt
Java Compiler Compiler Version 7.0.10 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file D-pression-.jjt . . .
File "TokenMgrError.java" does not exist. Will create one.
File "ParseException.java" does not exist. Will create one.
File "Token.java" does not exist. Will create one.
File "SimpleCharStream.java" does not exist. Will create one.
Parser generated successfully.

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>javac *.java

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>java Dp
Type the name of file to parse: fonte1.txt
ERROR
fonte1.txt (O sistema não pode encontrar o arquivo especificado)
Lexical Analysis Failed

C:\Users\victo\Documents\Universidade\Compiladores\2022\TB1\Source>
```

SOBRE A ESTRUTURA DO ANALISADOR

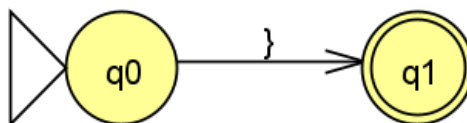
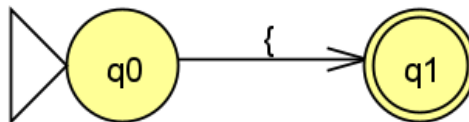
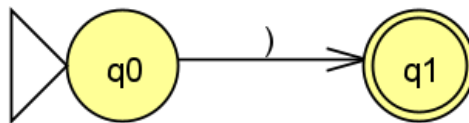
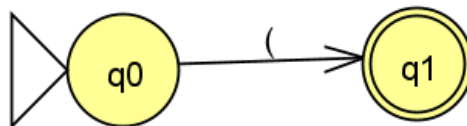
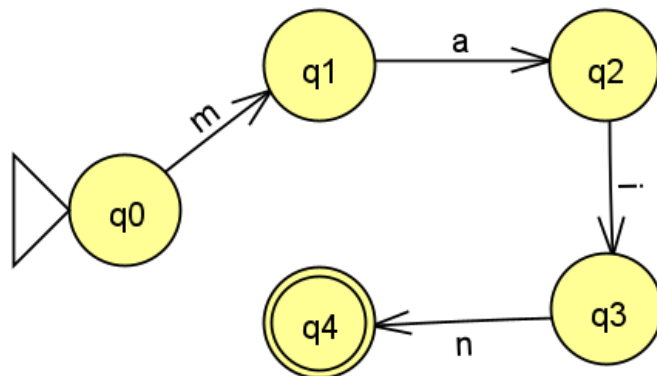
A linguagem D-pression- possui mais de 40 tokens divididos em:

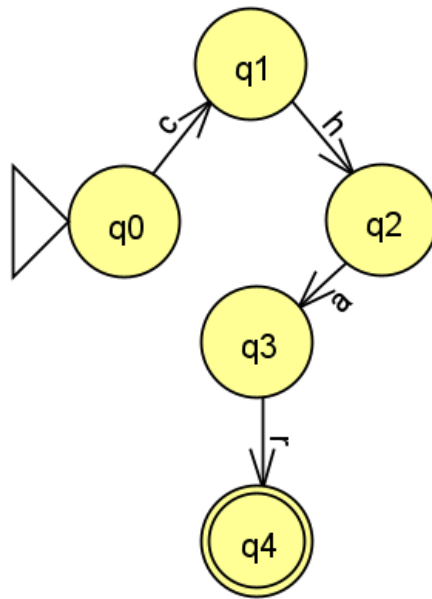
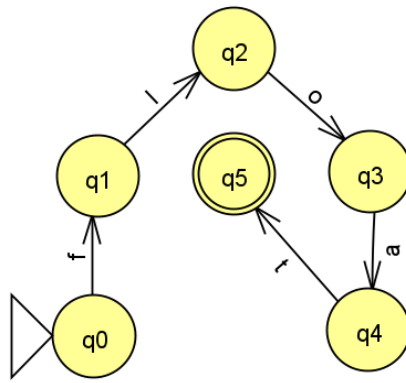
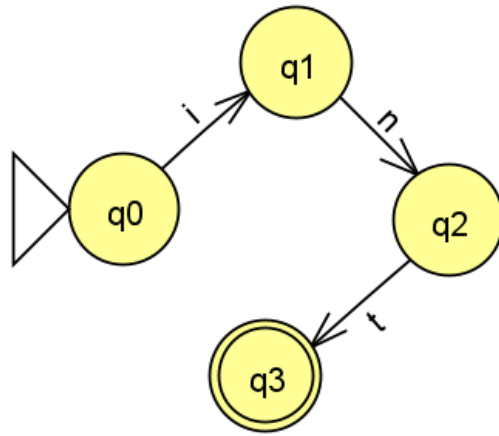
- Tokens para Palavras Reservadas: que incluem tokens para estruturas de controle, tipos de dados e palavras especiais como “main” e “print”.
- Tokens para Separadores: parênteses, chaves e colchetes, bem como vírgula, ponto e ponto e vírgula são separadores.
- Tokens para ID e Números: tokens com expressões regulares que permitem identificar cadeias de identificadores e números.
- Tokens para Operadores: com mais de 20 operadores, os tokens de operadores reconhecem sinais como adição e subtração.

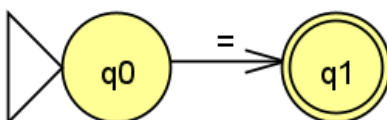
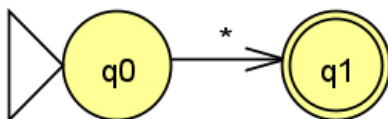
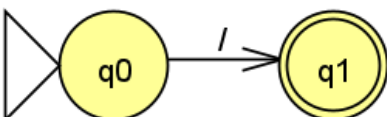
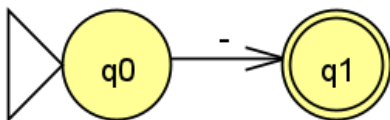
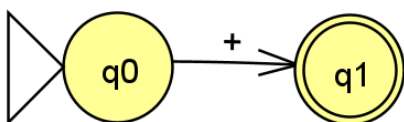
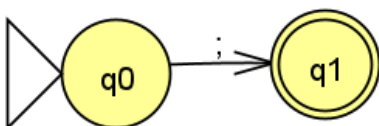
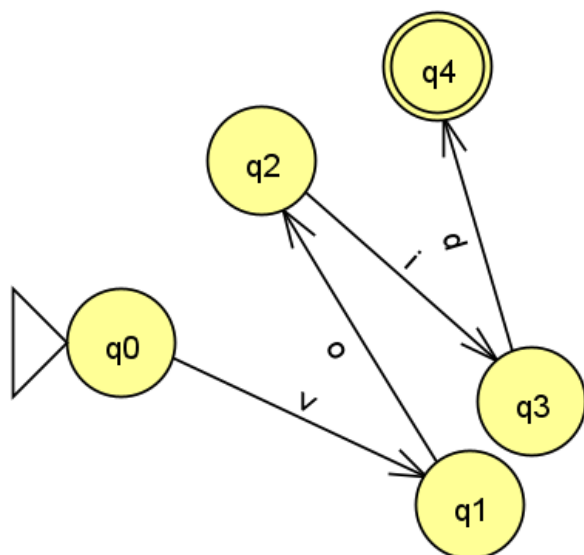
Um bloco de código que identifica tokens tem essa estrutura:

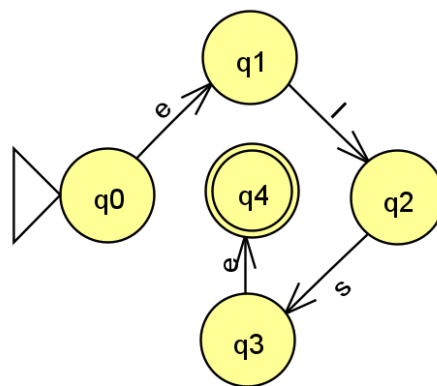
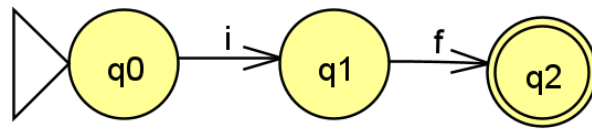
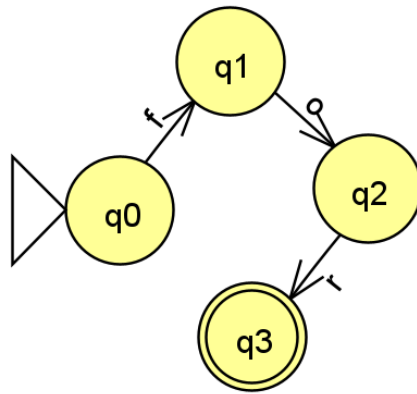
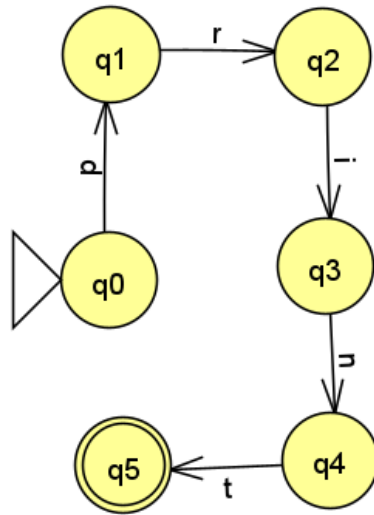
```
// TOKENS FOR SEPARATORS
TOKEN: {
    <LPAREN: "("> /* LEFT PARENTHESIS */
|   <RPAREN: ")"> /* RIGHT PARENTHESIS */
|   <LBRACE: "{"> /* LEFT BRACE */
|   <RBRACE: "}"> /* RIGHT BRACE */
|   <LBRACKET: "["> /* LEFT BRACKET */
|   <RBRACKET: "]"> /* RIGHT BRACKET */
|   <DOT: "."> /* DOT */
|   <COMMA: ","> /* COMMA */
|   <SEMICOLON: ";"> /* SEMICOLON */
}
```

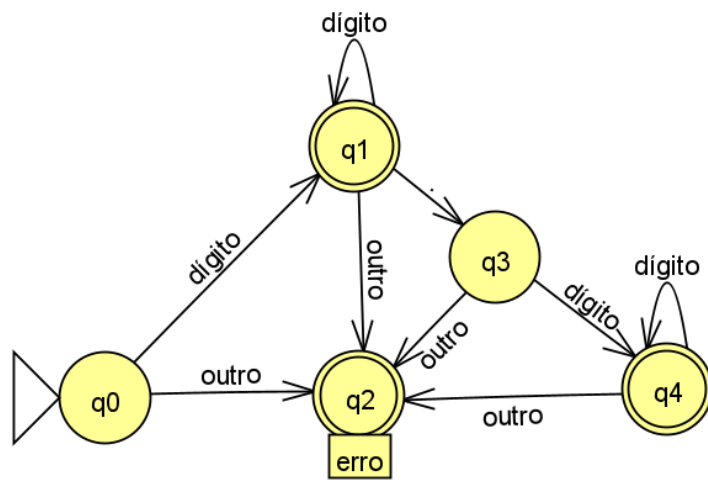
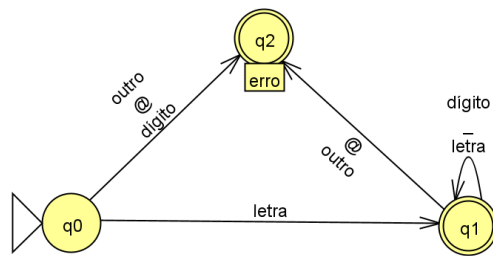
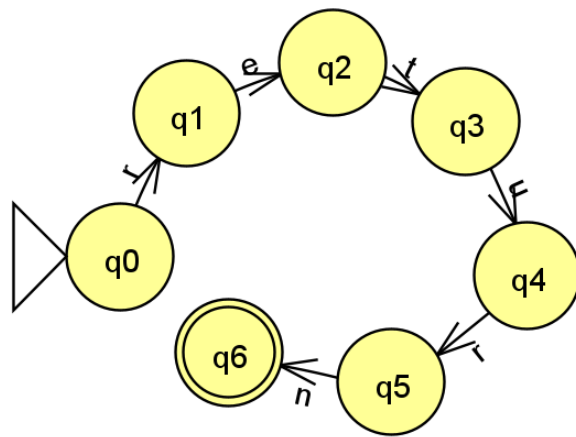
Como já citado, a linguagem D-expression- possui mais de 40 tokens, é incabível colocar todos os autômatos que os representam neste documento, a maior parte desses autômatos segue uma mesma estrutura, sendo AFDs bem simples, exemplos de alguns seguem abaixo:











Referências bibliográficas:

The JavaCC FAQ - Theodore S. Norvell

Compiladores - Regina Fedozzi

Como Construir um Compilador Utilizando Ferramentas Java - Márcio Delamaro