Universidade Federal de Santa Catarina - UFSC
Departamento de Informática e Estatística
Curso de Ciências da Computação
INE5426 - Construção de Compiladores

Matheus Leonel Balduino
Marcelo Pietro Grützmacher Contin

Relatório EP1 - Analisador Léxico

Florianópolis, 26 de Novembro de 2021

1 Introdução

Este relatório tem por finalidade descrever a forma como foi implementado um Analisador Léxico para a disciplina de Compiladores, no curso de Ciências da Computação da UFSC. O analisador implementado tem como alvo a gramática disponibilizada pelo professor, chamada CC-2021-2.

Durante a etapa de desenvolvimento, o grupo optou por utilizar uma ferramenta na construção e geração de alguns trechos de código, facilitando e acelerando o processo. A ferramenta selecionada foi o ANTLR, versão 4.

Para realizar a implementação, JAVA foi escolhido como a linguagem de programação.

2 Identificação dos Tokens

Para identificar os tokens, foi necessário analisar a gramática CC-2021-2, disponibilizada na página da disciplina. Após uma análise da gramática, os símbolos foram separados em terminais e não-terminais.

Ao término da análise, foi possível separar os tokens, já identificados, em outros grupos, como: Palavras reservadas, tipos, símbolos, etc. Foi necessário, também, criar regras para a identificação de espaços em branco.

2.1 Palavras Reservadas

IF	if
FOR	for
ELSE	else
RETURN	return
READ	read
PRINT	print
NEW	new
BREAK	break
DEF	def

2.2 Tipos

STRING	string
FLOAT	float
INT	int
NULL	null

2.3 Símbolos e Pontuação

LPAREN	(
RPAREN	)
LBRACE	{
RBRACE	}
LBRACK	[
RBRACK	]
SEMI	;
COMMA	,
DOT	.
COLON	:

2.4 Operadores Aritméticos

ASSIGN	=
ADD	+
SUB	-
MUL	*
DIV	/
MOD	%

2.5 Operadores Relacionais

GREATER_THAN	>
LESS_THAN	<
EQUAL	==
LESS_EQUAL	<=
GREATER_EQUAL	>=
NOT_EQUAL	!=

2.6 Tokens não-triviais

Foram utilizadas regras bem simples para a definição de números: inteiros são compostos por dígitos, enquanto que números reais (identificados pelo tipo *float*) são compostos de dígitos seguidos por um ponto e mais dígitos, ou apenas por um ponto seguido de dígitos.

INT_CONSTANT	DIGITS
FLOAT_CONSTANT	DIGITS.DIGITS | .DIGITS

Strings são definidas como tudo aquilo que encontra-se dentro de aspas duplas, exceto o próprio caractere que representa aspas duplas. Para identificadores, optou-se por uma abordagem parecida com a definida pela linguagem C, que acabou por se tornar uma espécie de padrão nas linguagens modernas: identificadores começam com uma letra ou “_” (underline), podendo letras, underlines ou dígitos após isso.

STRING_CONSTANT	““ ~(““)* ““
IDENT	[a-zA-Z_]([a-zA-Z0-9_])*

2.7 Espaço em Branco

Foi necessário fazer a aplicação de um método, chamado *skip*, para que o analisador léxico compreendesse que deveria ignorar os tokens formados por essa regra. Sendo o *skip* uma palavra reservada no ANTLR que garante que a sequência identificada pela expressão regular será desconsiderada.

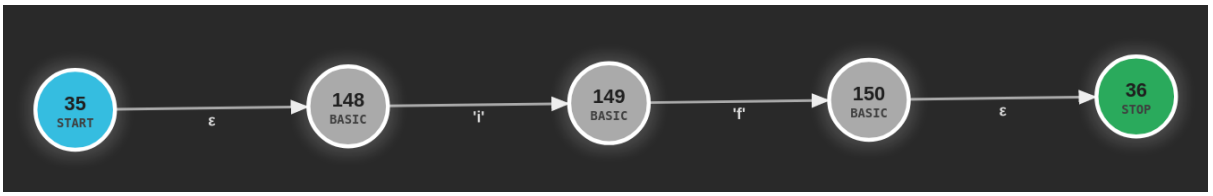
WHITESPACE	[\t\r\n]+ -> skip

3 Definições Regulares

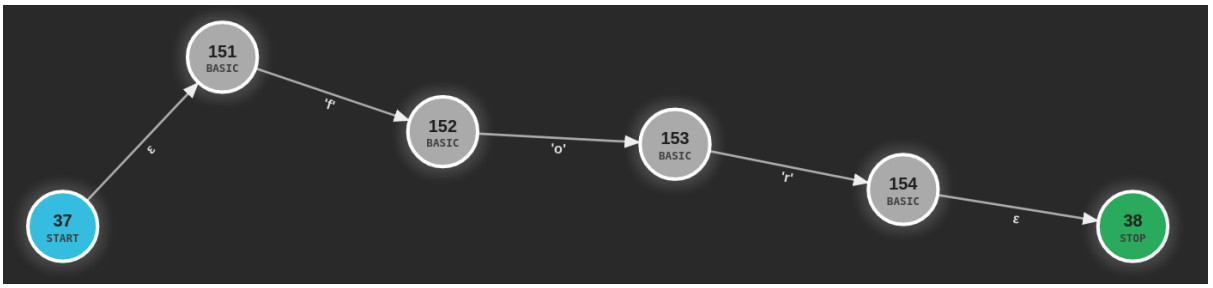
*letter_* -> [a-zA-Z_]	*lbrace* -> {
digit -> [0-9]	*rbrace* -> }
digits -> *digit*+	*lbrack* -> [
int_constant -> *digits*	*rbrack* ->]
string_constant -> “~[“]*”	*semi* -> ;
float_constant -> *digits* . *digits* | . *digits*	*comma* -> ,
identifier -> *letter_*(*letter_* | *digit*)*	*dot* -> .
white_space -> (| \n | \t | \r)	*colon* -> :
if -> if	*assign* -> =
for -> for	*add* -> +
else -> else	*sub* -> -
return -> return	*mul* -> *
read -> read	*div* -> /
print -> print	*mod* -> %
new -> new	*greater_than* -> >
break -> break	*less_than* -> <
def -> def	*equal* -> ==
string -> string	*less_equal* -> <=
int -> int	*greater_equal* -> >=
float -> float	*not_equal* -> !=
rparen ->)	*null* -> null
lparen -> (	

4 Diagramas de Transição

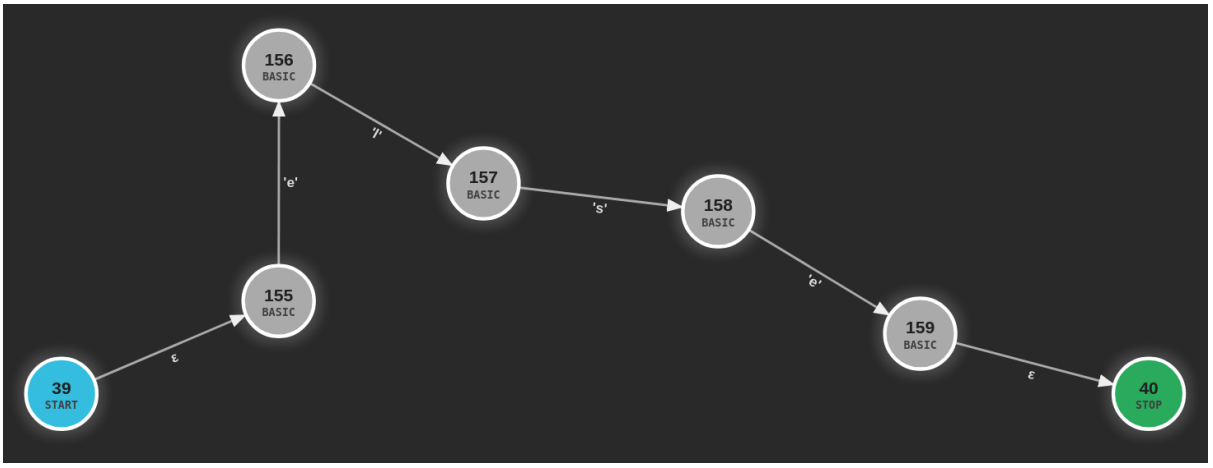
4.1 Token “if”



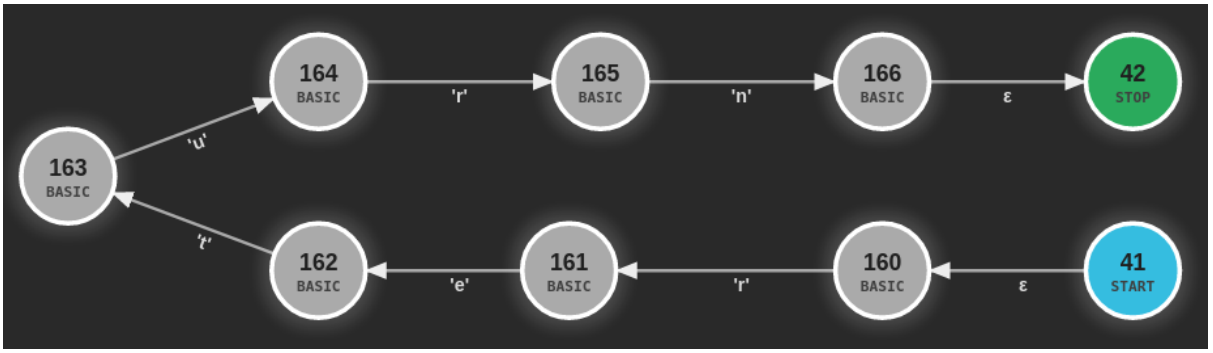
4.2 Token “for”



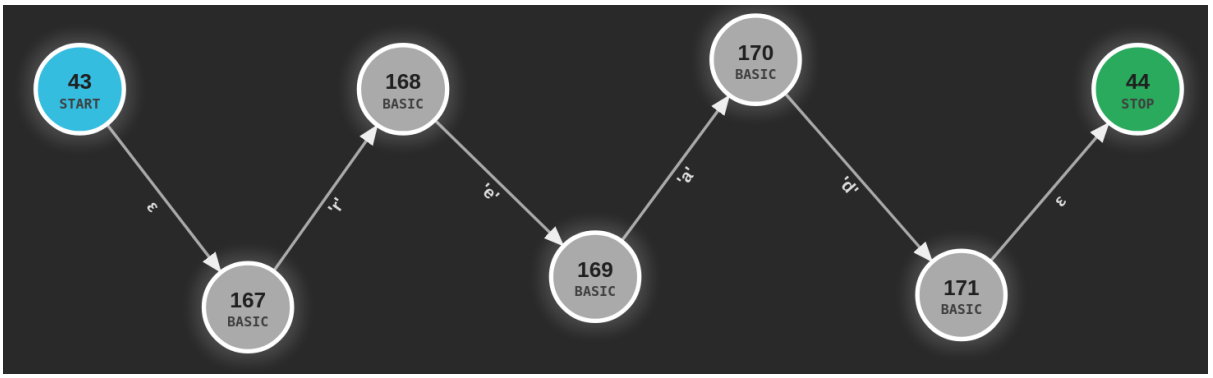
4.3 Token “else”



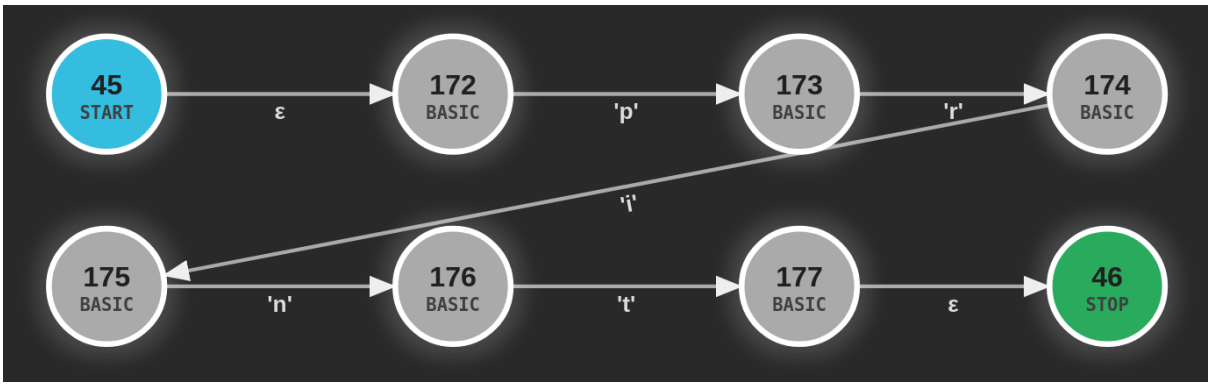
4.4 Token “return”



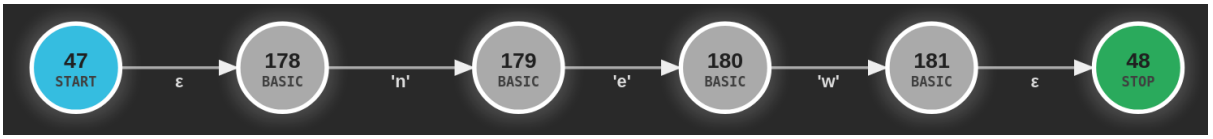
4.5 Token “read”



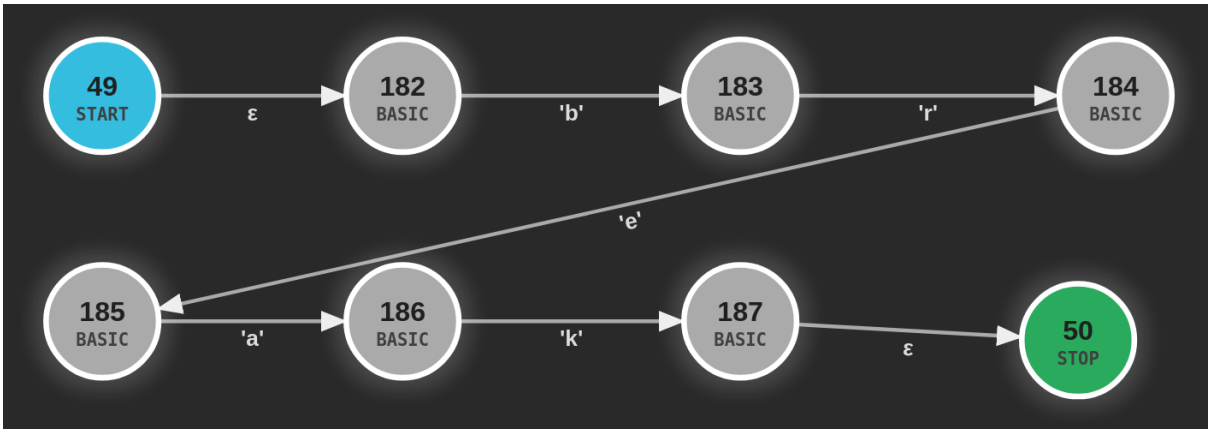
4.6 Token “print”



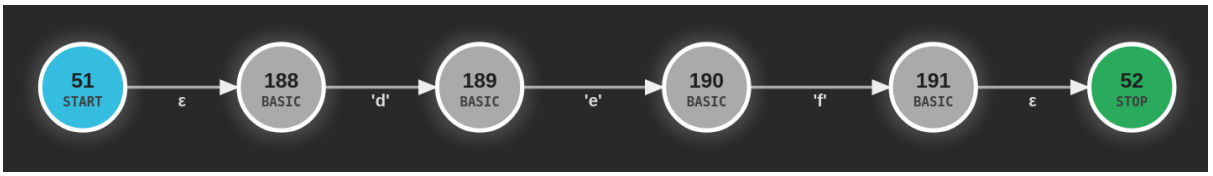
4.7 Token “new”



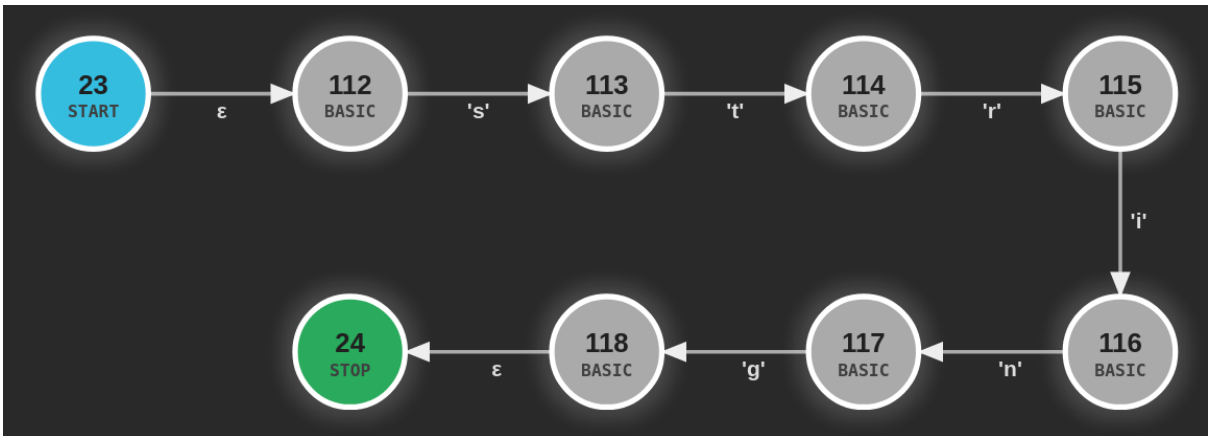
4.8 Token “break”



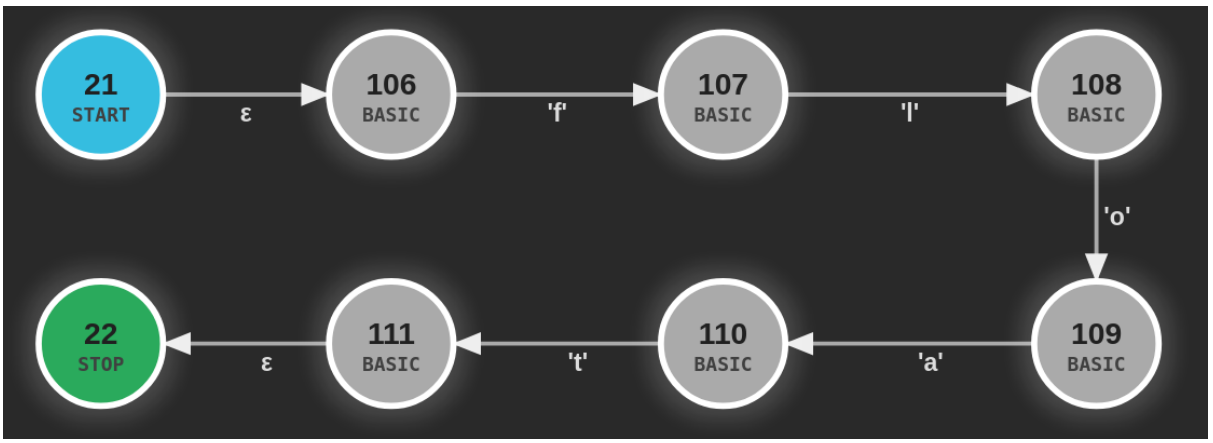
4.9 Token “def”



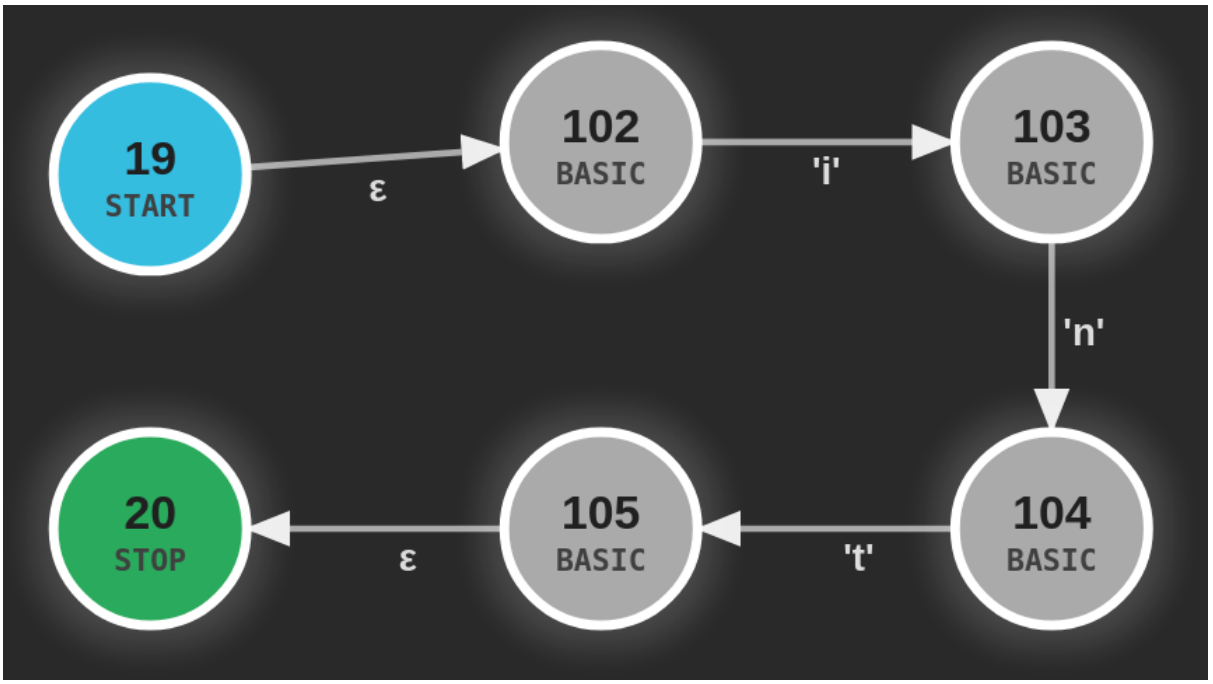
4.10 Token “string”



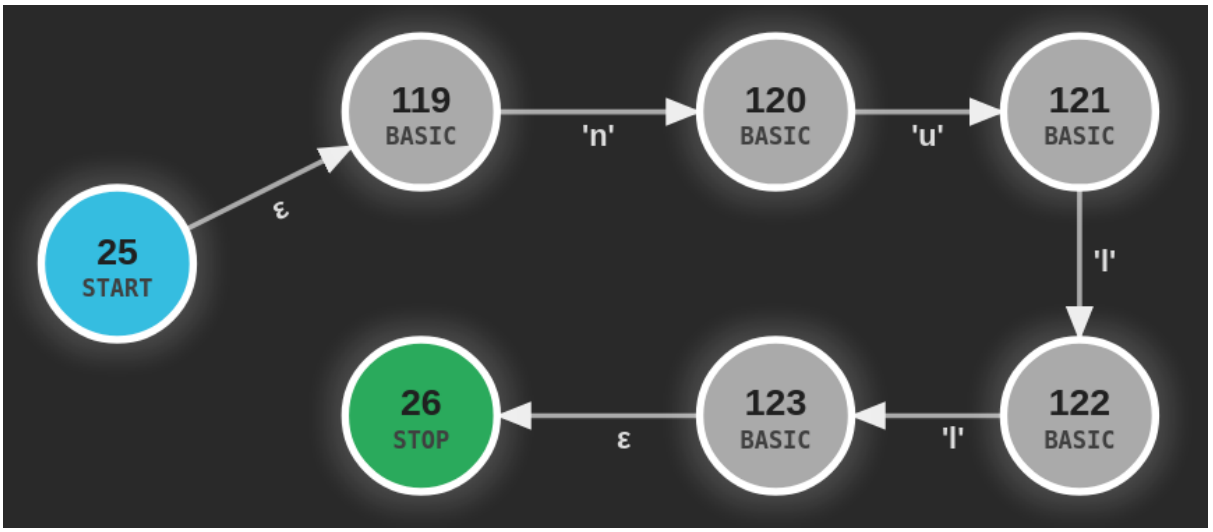
4.11 Token “float”



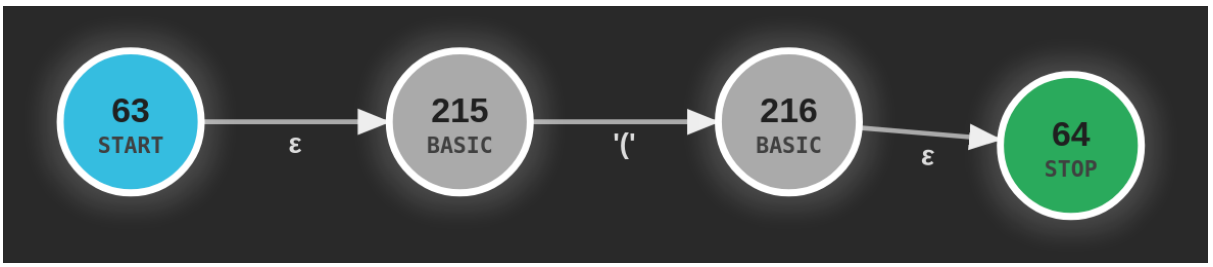
4.12 Token “int”



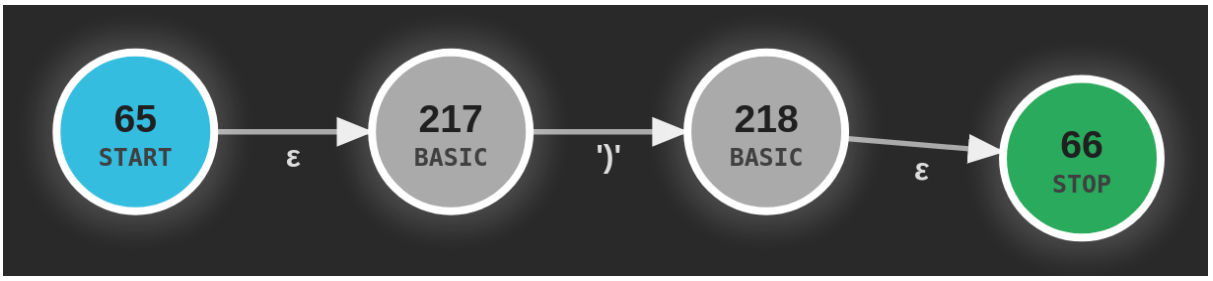
4.13 Token “null”



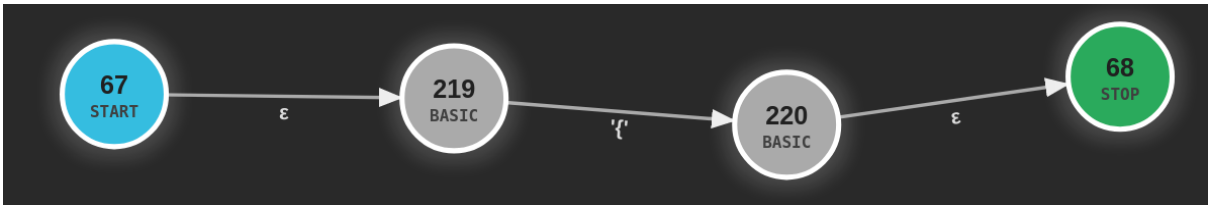
4.14 Token “Lparen”



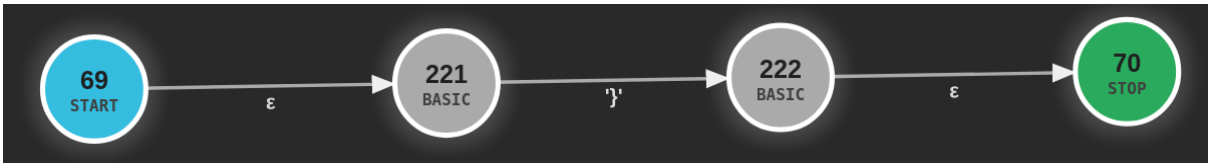
4.15 Token “Rparen”



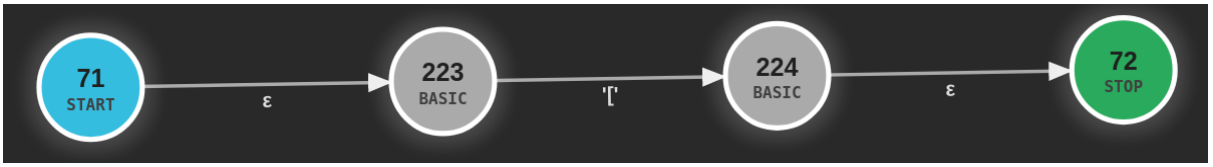
4.16 Token “Lbrace”



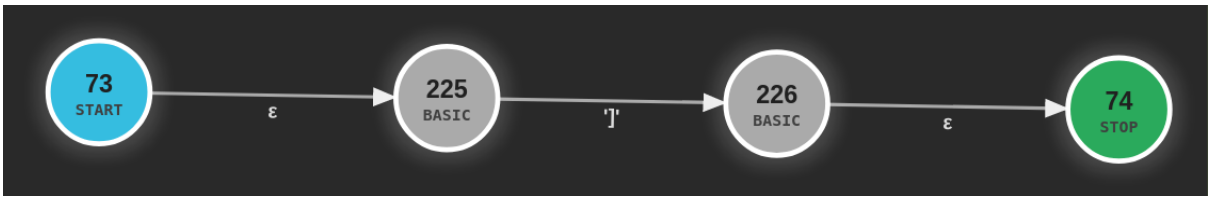
4.17 Token “Rbrace”



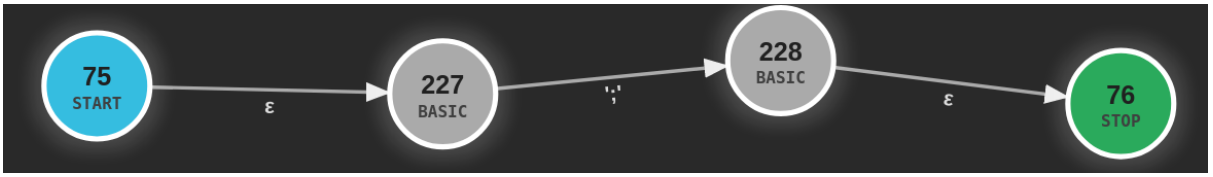
4.18 Token “Lbrack”



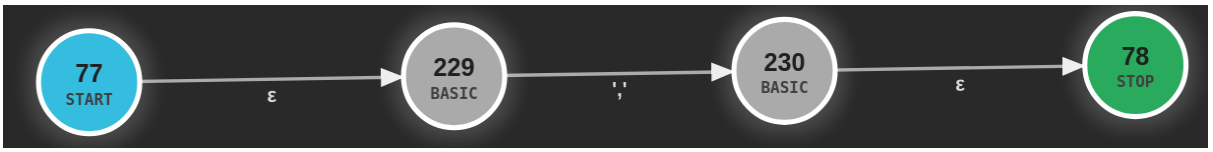
4.19 Token “Rbrack”



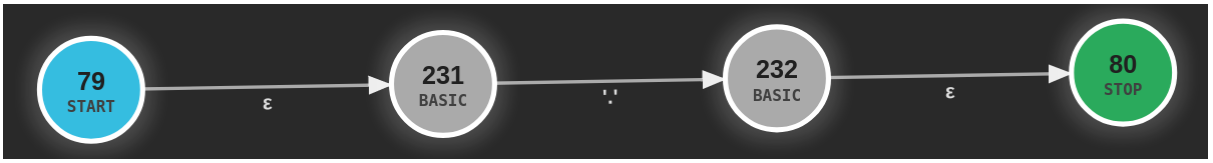
4.20 Token “semi”



4.21 Token “comma”

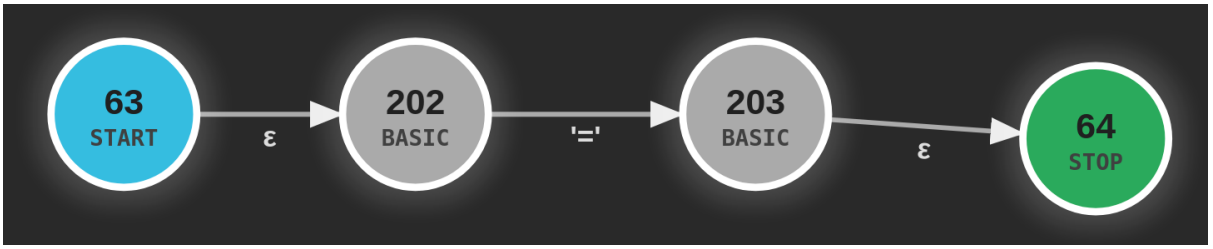


4.22 Token “dot”

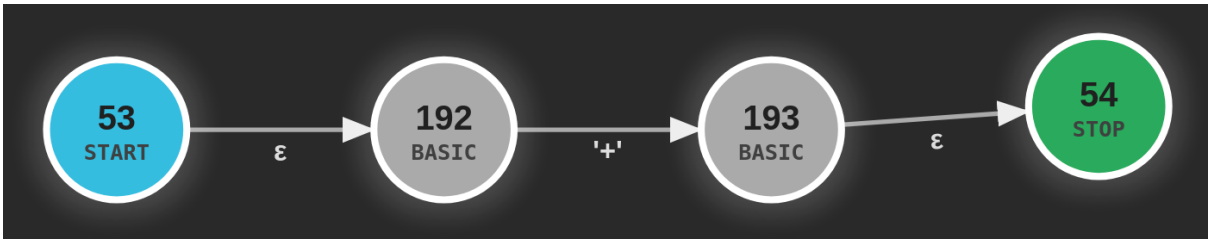


4.23 Token “colon”

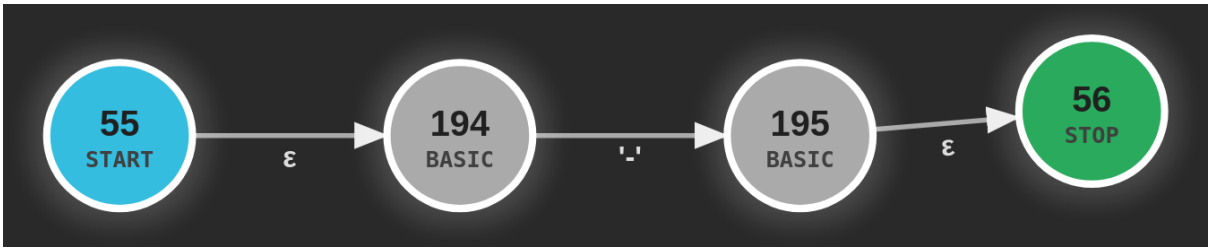
4.24 Token “assign”



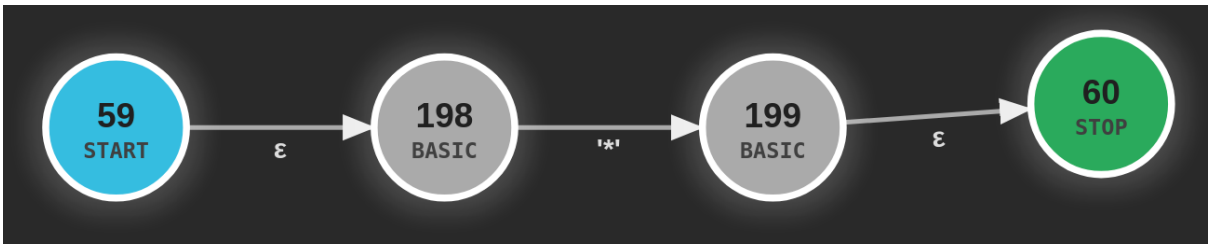
4.25 Token “add”



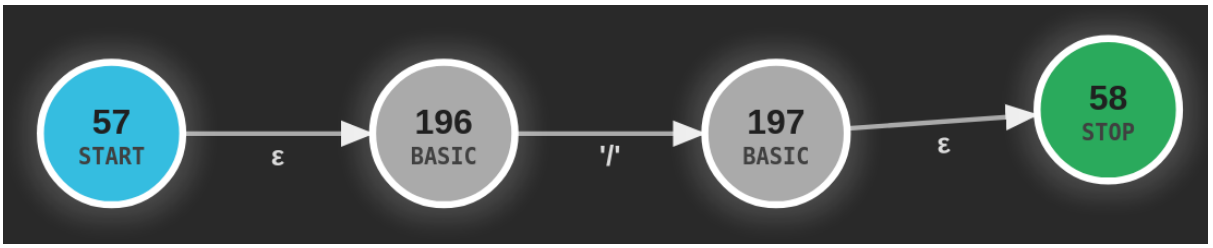
4.26 Token “sub”



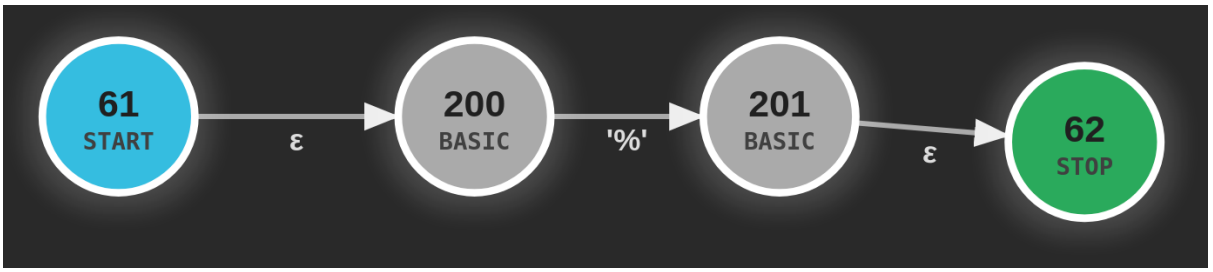
4.27 Token “mul”



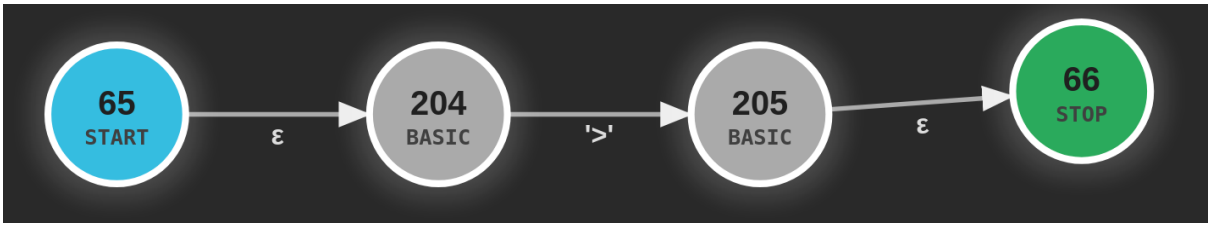
4.28 Token “div”



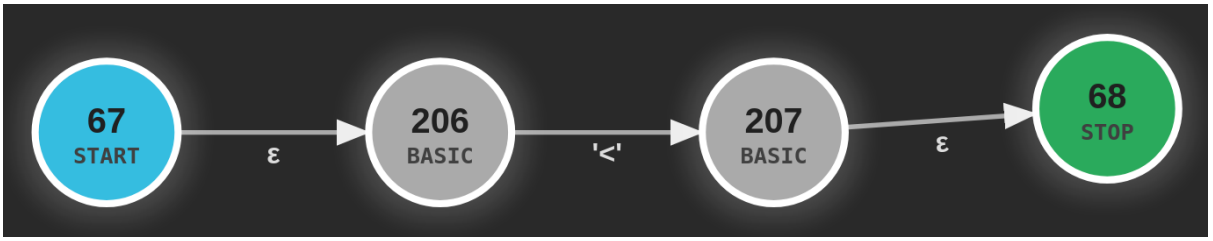
4.29 Token “mod”



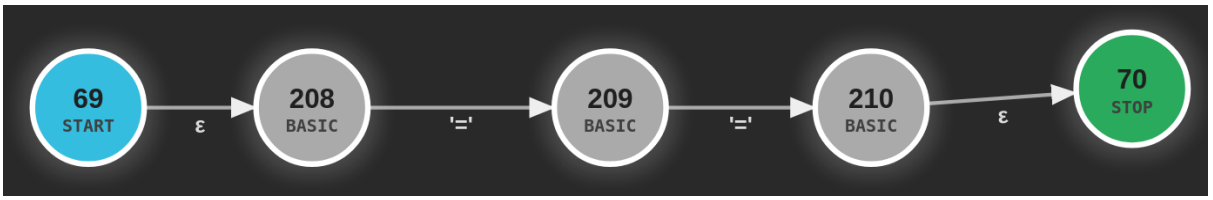
4.30 Token “greater_than”



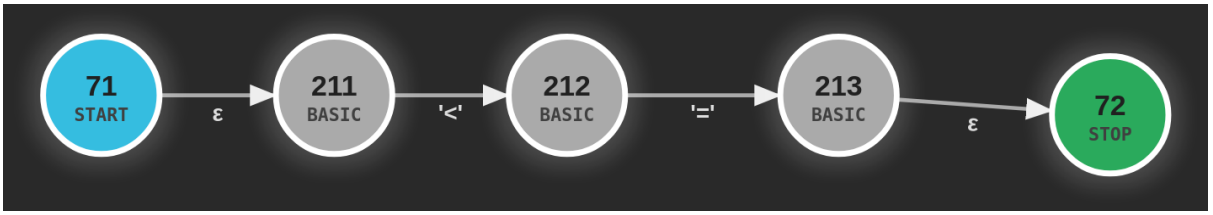
4.31 Token “less_than”



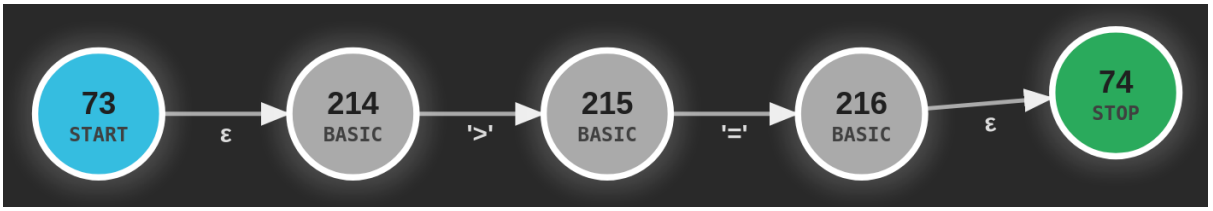
4.32 Token “equal”



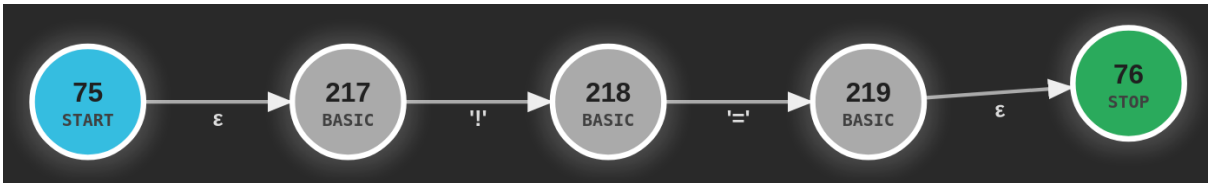
4.33 Token “less_equal”



4.34 Token “greater_equal”

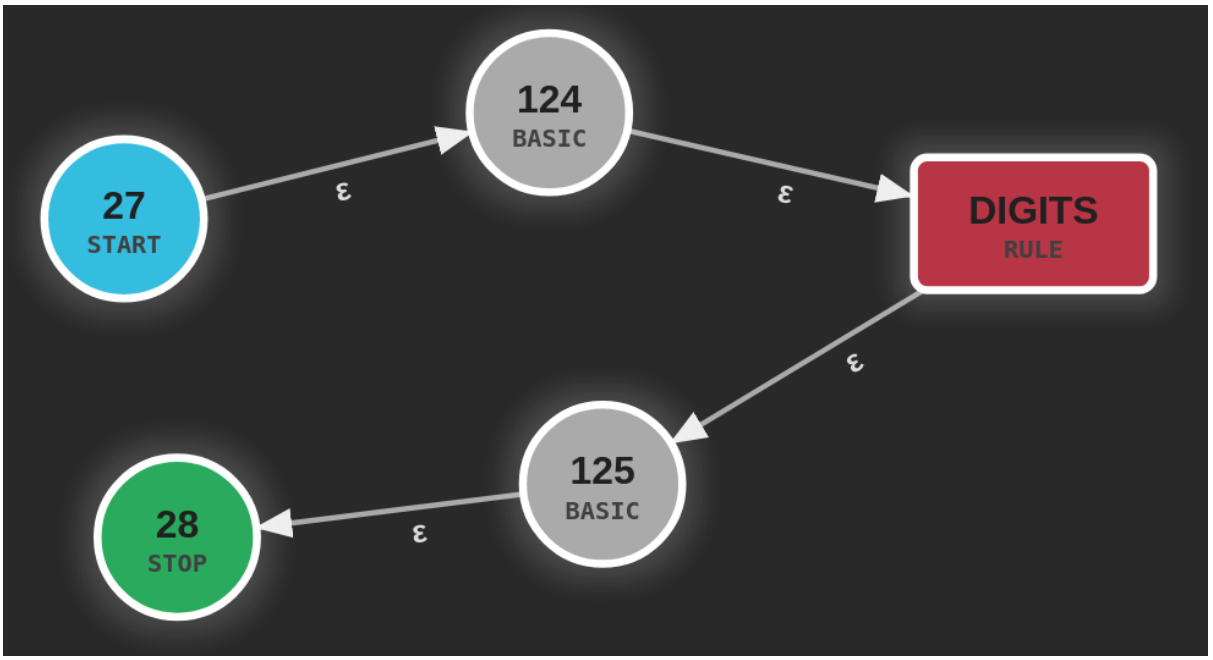


4.35 Token “not_equal”

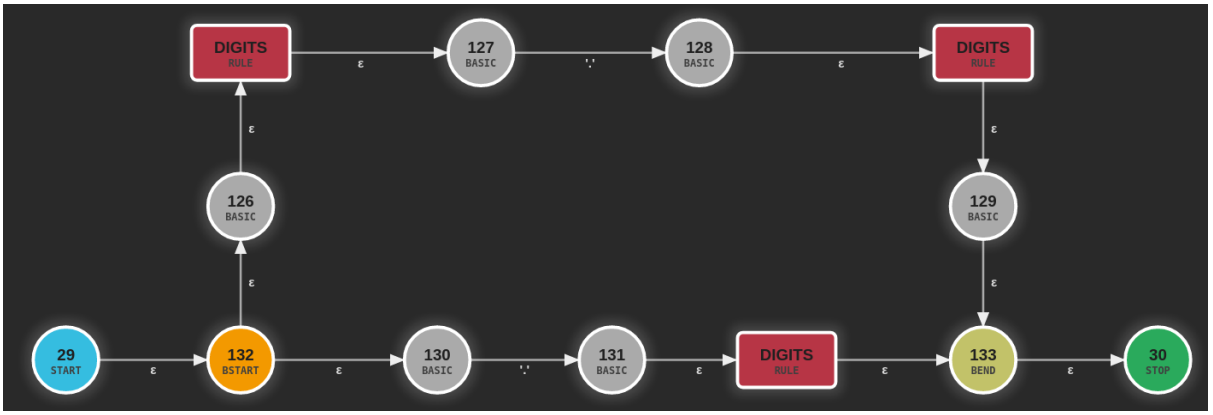


4.36 Token não-triviais

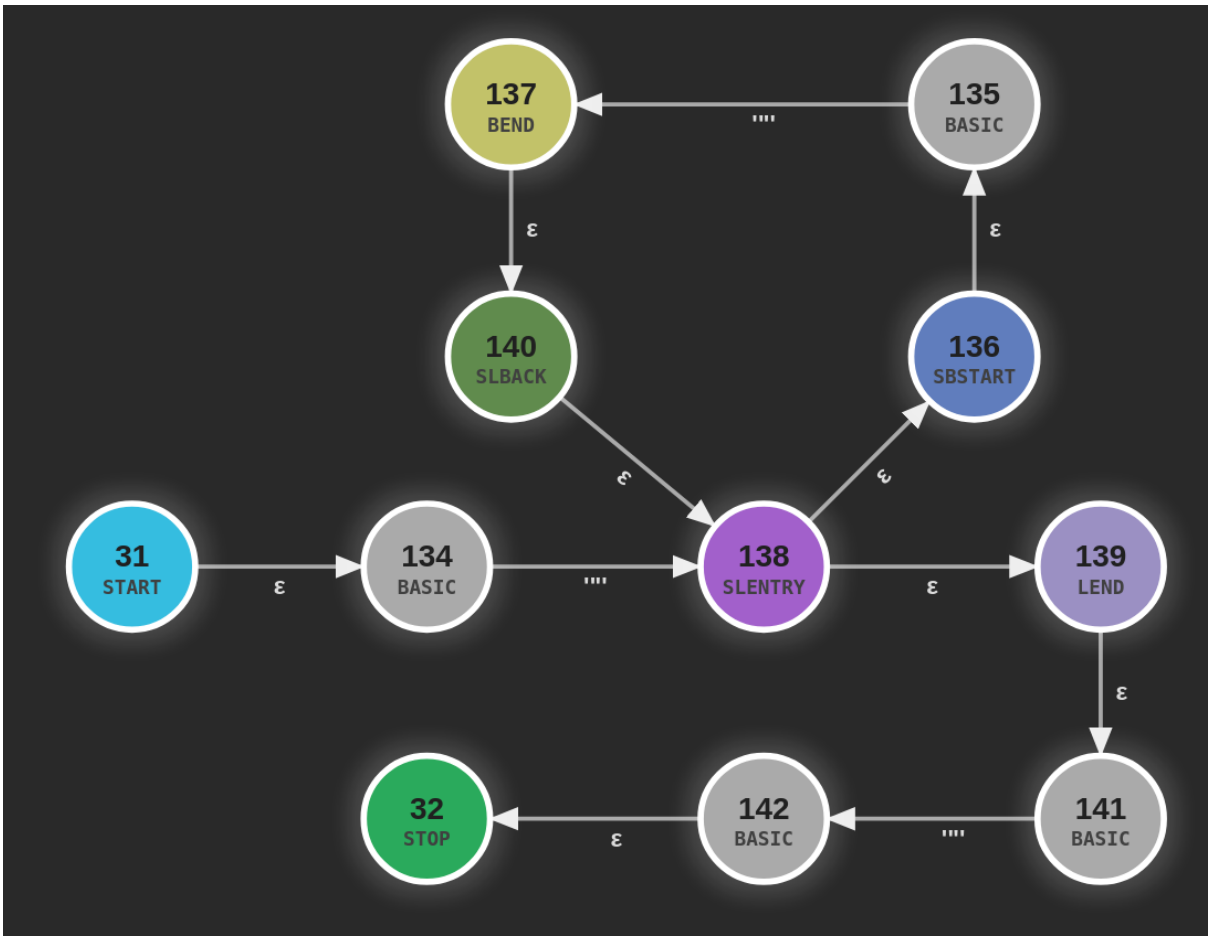
4.36.1 Token “int_constant”



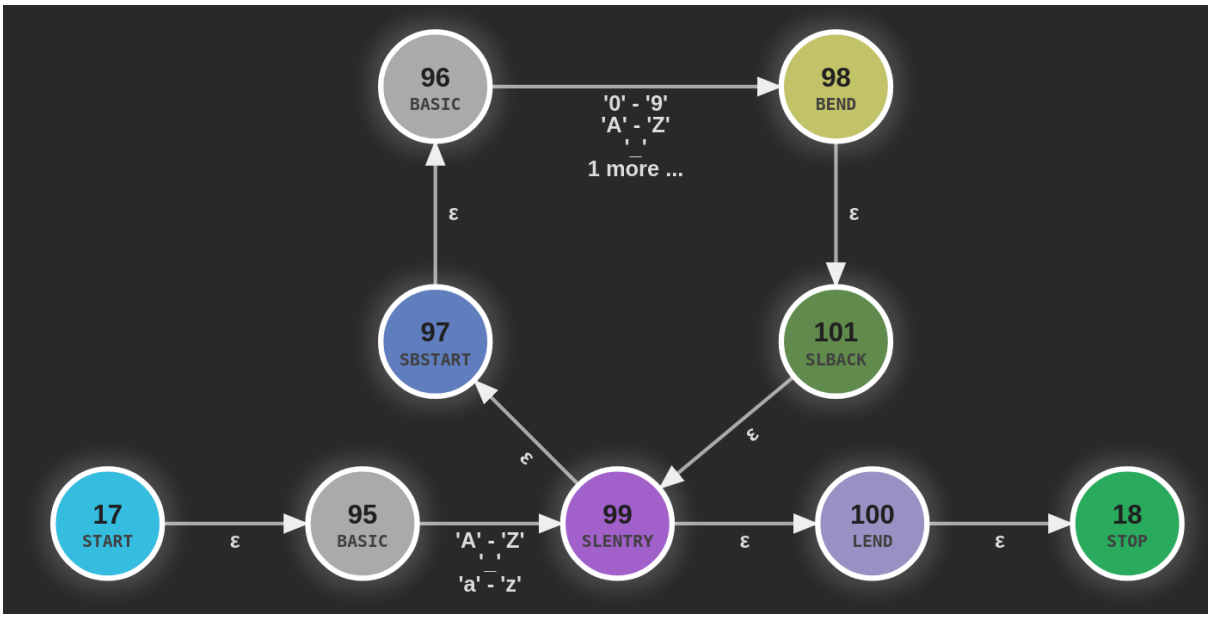
4.36.2 Token “float_constant”



4.36.3 Token “string_constant”



4.36.4 Token “ident”



5 Tabela de Símbolos

O mesmo foi gerado a partir da criação de um Set filtrando apenas por lexemas do tipo IDENT (de identificador). E nesta tabela existe um contador de símbolos do tipo IDENT com os respectivos lexemas. Por se tratar de um Set ele não terá repetições, mostrando então apenas lexemas únicos do tipo identificador.

O mesmo será gerado na pasta *saida_lexico/tabela_simbolos* com um arquivo de mesmo nome do de entrada para rodar o analisador.

6 Tabela de Tokens

A tabela de Tokens exibe o lexema, seu tipo, a linha do código em que se encontra e o índice inicial e final do mesmo na linha em que se encontra.

Essa tabela é gerada e armazenada na pasta *saida_lexico/tabela_tokens* com um arquivo de mesmo nome do de entrada para rodar o analisador.

7 A Ferramenta

A implementação do Analisador Léxico, como já comentado, foi feita através de um ferramenta de geração de código, chamada ANTLR (versão 4).

ANTLR possui integração com JAVA, que foi a nossa linguagem de programação escolhida, portanto isso é um ponto que influenciou na nossa escolha. Outro fator, é a familiaridade de alguns membros da equipe com a ferramenta e principalmente a linguagem Java.

Na parte dos diagramas, também foi possível fazer uso de ferramentas de automatização, não sendo necessária a construção de cada um dos diagramas de forma manual.

7.1 Descrição da entrada da ferramenta

ANTLR espera que o usuário entregue um arquivo de extensão “.g4” como entrada. Esse arquivo é o responsável por definir todas as regras da gramática a ser utilizada. A sintaxe é bem simples, sendo basicamente a definição do nome da regra, seguida pela definição e um ponto-e-vírgula identificando o fim daquela regra. Exemplo:

NOMEREGRA: <definicao>;

Nos arquivos enviados junto com este relatório, é possível visualizar a aplicação da sintaxe na prática. O mesmo se encontra na pasta src/antlr4/CC20212.g4.

O mesmo após a criação do arquivo gera diversos outros arquivos dentro da pasta target, o qual são utilizados posteriormente na aplicação para fazer o parser, identificação, geração da lista de tokens e verificação de erro léxico.

7.2 Descrição da saída da ferramenta

Ao executar o ANTLR, com os arquivos exigidos (no caso, o arquivo .g4), temos como saída uma classe (gerada automaticamente) que irá se parecer com “*<NomeArquivo.g4>Lexer.java*”. Nesta classe, estão os métodos que definem o analisador léxico.

Com um arquivo de código-fonte da linguagem representada pelo arquivo .g4, é possível fazer a identificação dos tokens presentes nesse arquivo. É possível, também, a identificação de erros que podem existir no código-fonte lido como já mencionado anteriormente.