Integrantes:
Arthur Resende Santos
Daniel Dias Barbosa
Flávio Vezono Filho

Projeto da Linguagem:

GLC

S -> PROGRAM id() bloco

bloco -> BEGIN declaração comandos END
bloco -> BEGIN comandos END

declaração -> tipo : lista_id ;
declaração -> tipo : lista_id ; declaração ;

tipo -> int
tipo -> char
tipo -> float

lista_id -> id
lista_id -> id , lista_id

comandos -> comando
comandos -> comando comandos

comando -> seleção
comando -> atribuição
comando -> repetição

seleção -> if condição then instruções
seleção -> if condição then instruções else instruções

instruções -> bloco
instruções -> comando

atribuição -> id := expressão ;

repetição -> while condição instruções
repetição -> repeat instruções until condição ;

condição -> elemento relop elemento

elemento -> id
elemento -> constante

relop -> =
relop -> !=
relop -> >=
relop -> <=

expressão -> (expressão)
expressão -> elemento
expressão -> expressão ariop expressão

ariop -> +
ariop -> -
ariop -> *
ariop -> /
ariop -> ^

constante -> caracteres
constante -> numeroInt
constante -> numeroFloat

Tabela de tokens:

Comentário, tabulação, quebra de linha e espaço
Palavra-chave para a função principal do programa
Palavra-chave para início de bloco
Palavra-chave para término de bloco
Tipo inteiro
Tipo ponto flutuante
Tipo caractere
Identificador, nome das variáveis
Primeira palavra-chave para condição
Segunda palavra-chave para condição
Palavra-chave para fluxo alternativo da condição
Palavra-chave para tipo 1 de repetição
Primeira palavra-chave para tipo 2 de repetição
Segunda palavra-chave para tipo 2 de repetição
Abre parênteses, para preferência
Fecha parênteses
Número Inteiro
Número Float
Qualquer caractere no formato 'c'
Dois pontos
Ponto e vírgula
Atribuição
Operador relacional de menor que
Operador relacional de menor e igual à
Operador relacional de igual à
Operador relacional de diferente de
Operador relacional de maior que
Operador relacional de maior e igual à
Operador aritmético de soma
Operador aritmético de subtração
Operador aritmético de multiplicação
Operador aritmético de divisão
Operador aritmético de exponenciação

Tokens	
Nome do Token	Valor do Atributo
ws	-
program	-
begin	-
end	-
int	-
float	-
char	-
id	Posição na tabela de símbolo
If	-
then	-
else	-
while	-
repeat	-
until	-
(	-
)	-
numeroInt	Posição na tabela de símbolos
numeroIFloat	Posição na tabela de símbolos
caracteres	Posição na tabela de símbolos
:	-
;	-
:=	-
relop	LT
relop	LE
relop	EQ
relop	NE
relop	GT
relop	GE
+	-
-	-
*	-
/	-
^	-

Expressões regulares:

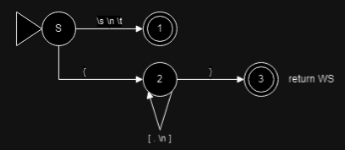
ws -> [\b \n \t] | {[. \n]*}

program -> program
begin -> begin
end -> end
int -> int
float -> float
char -> char
id -> [A – Z a – z _][[A – Z a – z _] [0 – 9]] *
if -> if
then -> then
else -> else
while -> while
repeat -> repeat
until -> until
(-> \(
) -> \)
numeroInt -> -? [0-9]+
numeroFloat -> -? [0-9]+ \. [0-9]+ [E [0-9]+]?
caracteres -> '[. \n]'
: -> :
; -> ;
:= -> :=
= -> =
LT -> <
LE -> <=
EQ -> ==
NE -> !=
GT -> <
GE -> >=
+ -> +
- -> -
*** -> ***
/ -> /
^ -> ^

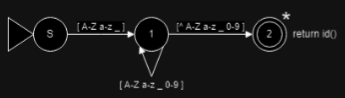
Análise Léxica:

Diagramas de transição:

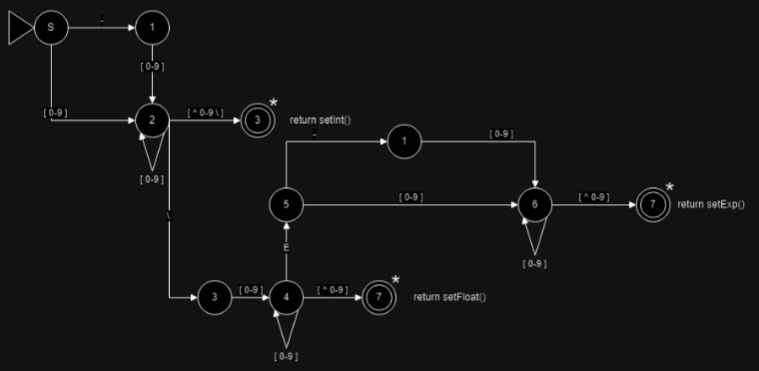
WS



ID

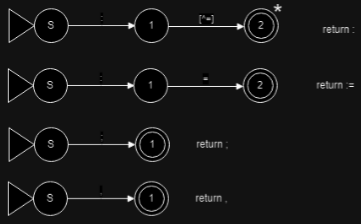


numero

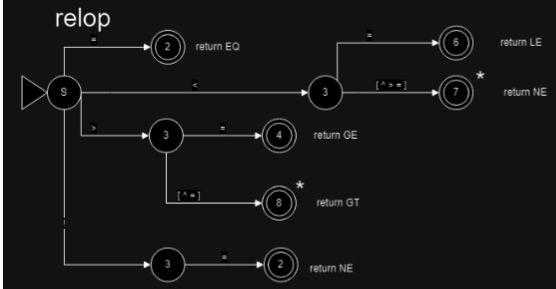


caracteres

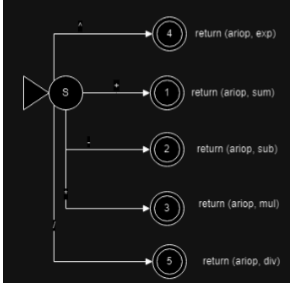
: := ;



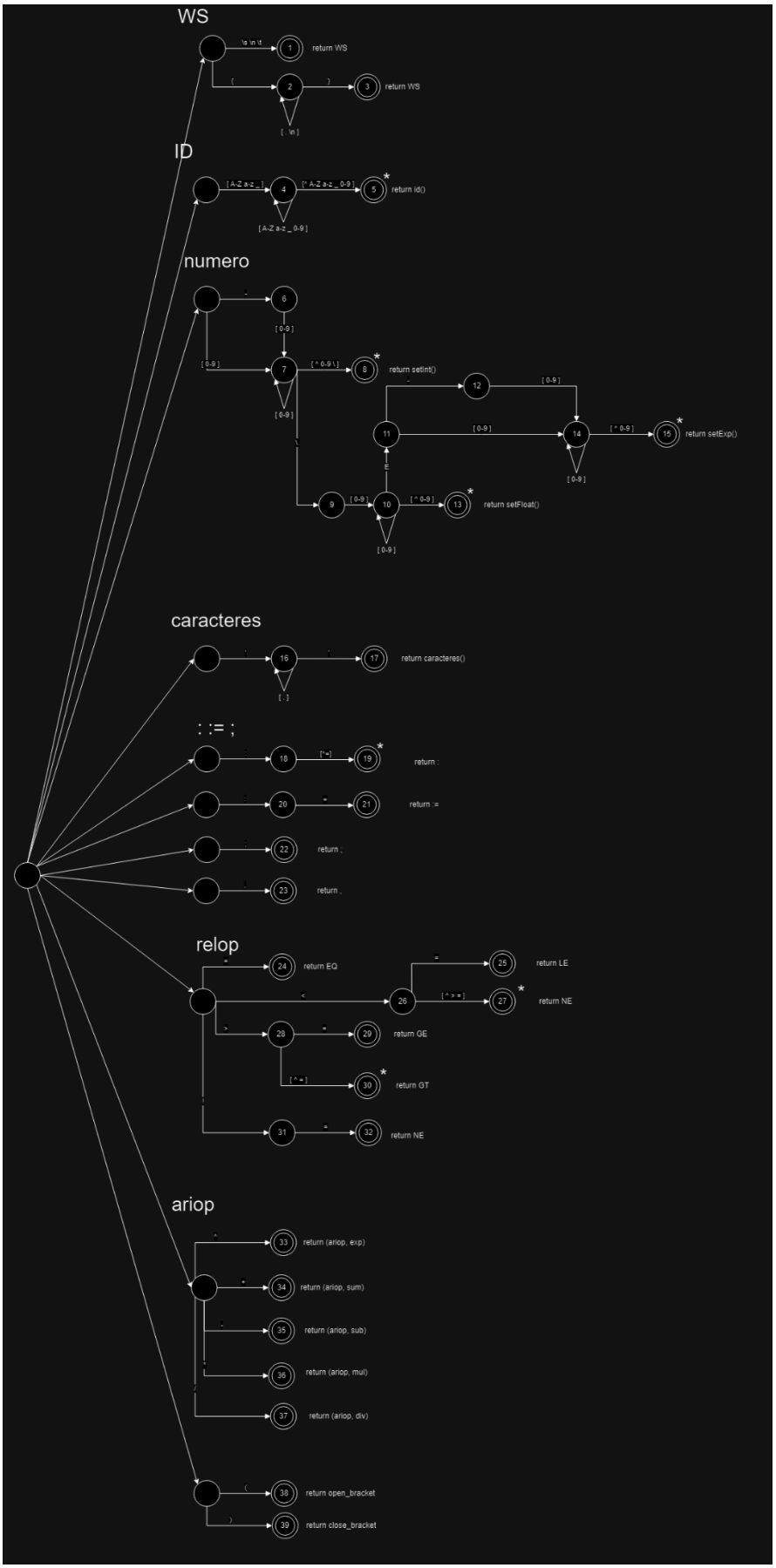
relop



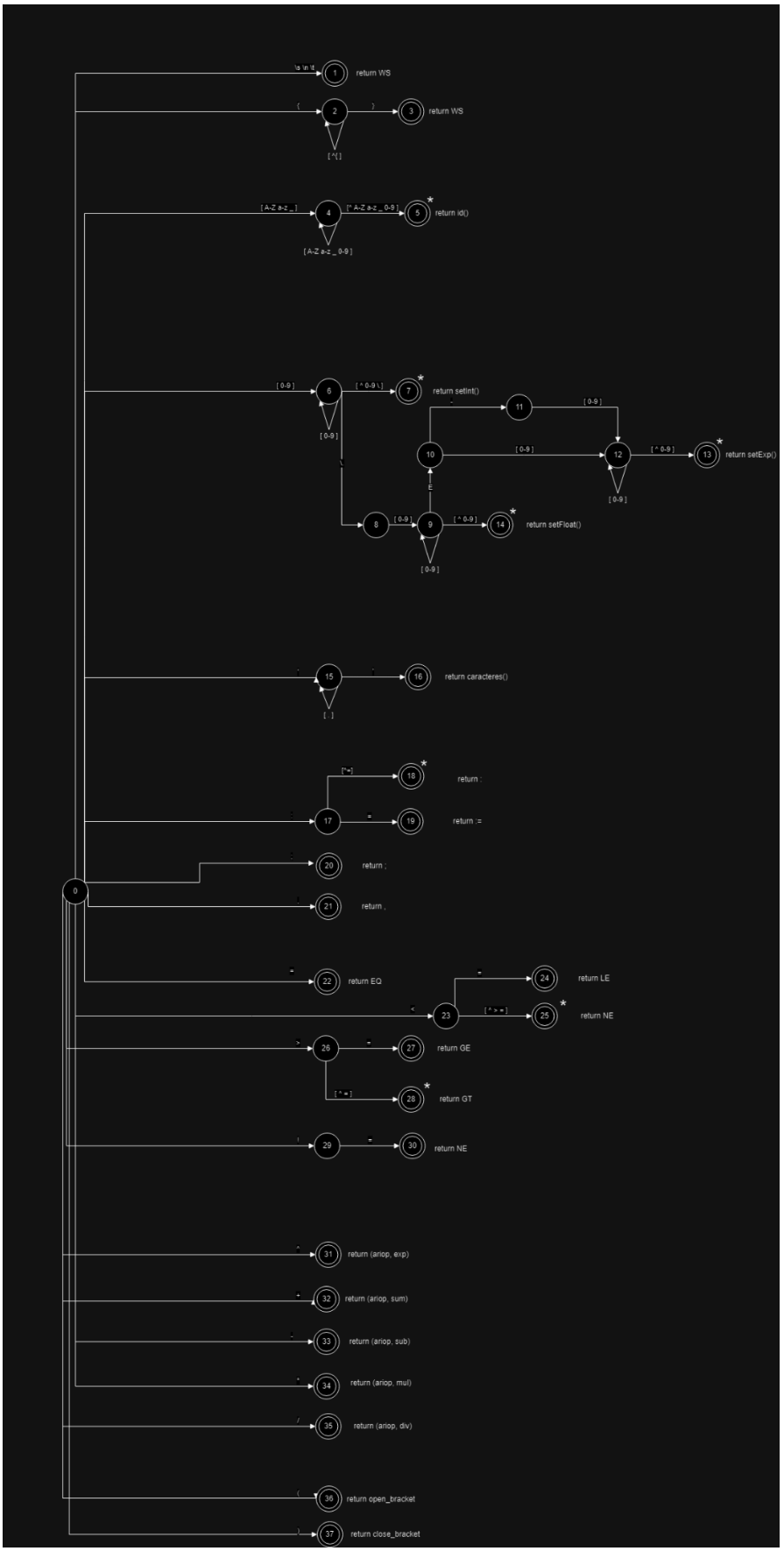
ariop



AFND único:


WS
\s \n \t
1
return WS
{
2
}
3
return WS
[\n]
ID
[A-Z a-z _]
4
[^ A-Z a-z _ 0-9]
5
return id()
[A-Z a-z _ 0-9]
numero
-
6
[0-9]
[0-9]
7
[^ 0-9 \ ]
8
return setInt()
[0-9]
\
12
[0-9]
11
[0-9]
14
[^ 0-9]
15
return setExp()
E
[0-9]
9
[0-9]
10
[^ 0-9]
13
return setFloat()
[0-9]
caracteres
'
16
'
17
return caracteres()
[.]
: := ;
:
18
[^=]
19
return :
:
20
=
21
return :=
;
22
return ;
,
23
return ,
relop
=
24
return EQ
<
26
=
25
return LE
[^ > =]
27
return NE
>
28
=
29
return GE
[^ =]
30
return GT
!
31
=
32
return NE
ariop
*
33
return (ariop, exp)
+
34
return (ariop, sum)
-
35
return (ariop, sub)
36
return (ariop, mul)
/
37
return (ariop, div)
(
38
return open_bracket
)
39
return close_bracket

AFD:


0
1
return WS
2
3
return WS
[^{]
4
[A-Z a-z _]
[A-Z a-z _ 0-9]
5
[^ A-Z a-z _ 0-9]
return id()
6
[0-9]
[0-9]
7
[^ 0-9 \ ]
return setInt()
8
[0-9]
9
[0-9]
10
11
[0-9]
12
[0-9]
13
[^ 0-9]
return setExp()
14
[^ 0-9]
return setFloat()
15
[']
16
return caracteres()
17
[^=]
18
return :
19
=
return :=
20
return ;
21
return ,
22
return EQ
23
24
return LE
25
[^ > =]
return NE
26
27
return GE
[^ =]
28
return GT
29
30
return NE
31
return (ariop, exp)
32
return (ariop, sum)
33
return (ariop, sub)
34
return (ariop, mul)
35
return (ariop, div)
36
return open_bracket
37
return close_bracket

Análise Sintática:

Remoção de recursão a esquerda

S -> PROGRAM id() bloco

bloco -> BEGIN inicio_blocos
inicio_blocos -> declaração comandos END
inicio_blocos -> comandos END

declaração -> tipo : lista_id ; declaração'
declaração' -> declaração
declaração' -> ε

tipo -> int
tipo -> char
tipo -> float

lista_id -> id
lista_id -> id , lista_id

comandos -> comando comando'
comando' -> comandos
comando' -> ε

comando -> seleção
comando -> atribuição
comando -> repetição

seleção -> if condição then instruções seleção'
seleção' -> else instruções
seleção' -> ε

instruções -> bloco
instruções -> comando

atribuição -> id := expressão ;

repetição -> while condição instruções
repetição -> repeat instruções until condição ;

condição -> elemento relop elemento

elemento -> id
elemento -> constante

relop -> =
relop -> !=
relop -> >=
relop -> <=

expressão -> elemento expressão'
expressão -> (expressão) expressão'
expressão' -> ariop expressão expressão'
expressão' -> ε

ariop -> +
ariop -> -
ariop -> *
ariop -> /
ariop -> ^

constante -> caracteres
constante -> numeroInt
constante -> numeroFloat

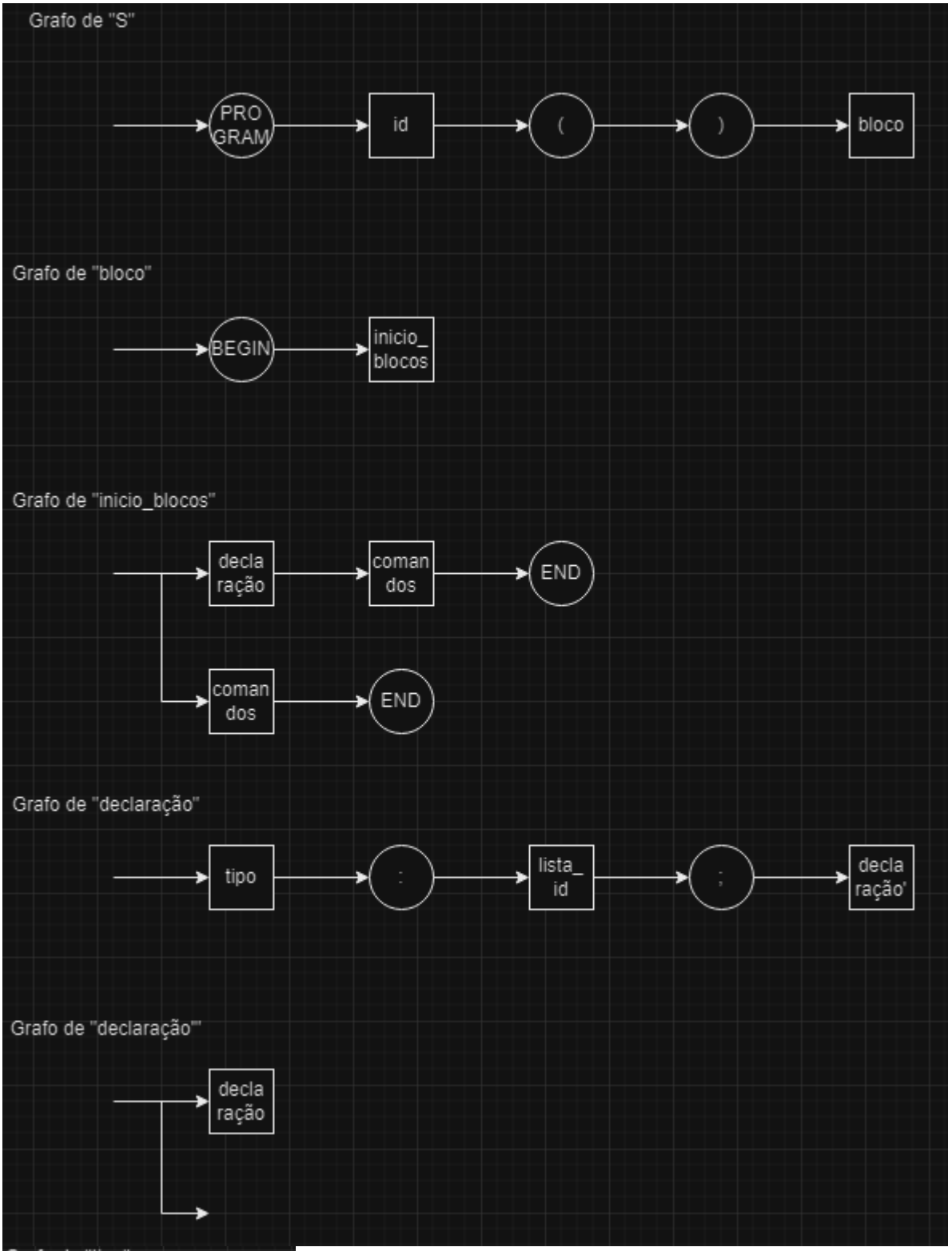
FIRST
EXPRESSAO

FIRST(S) = {program}
FIRST(bloco) = {BEGIN}
FIRST(inicio_blocos) = {int, char, float, id, if, while, repeat}
FIRST(declaração) = {int, char, float,ε}
FIRST(tipo) = {int, char, float}
FIRST(lista_id) = {id}
FIRST(comandos) = {id, if, while, repeat}
FIRST(comando) = {id, if, while, repeat}
FIRST(seleção) = {if}
FIRST(instruções) = {BEGIN, id, if, while, repeat}
FIRST(atribuição) = {id}
FIRST(repetição) = {while, repeat}
FIRST(condição) = {id, constante}
FIRST(elemento) = {id, caracteres, numeroInt, numeroFloat}
FIRST(relop) = {=, !=, >=, <=}
FIRST(expressão) = {(, id, caracteres, numeroInt, numeroFloat}
FIRST(ariop) = {+, -, *, /, ^}
FIRST(constante) = {caracteres, numeroInt, numeroFloat}

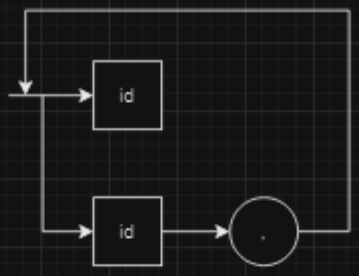
FOLLOW

FOLLOW(S) = {$}
FOLLOW(bloco) = {$}
FOLLOW(inicio_blocos) = {$}
FOLLOW(declaração) = {id, if, while, repeat}
FOLLOW(declaração') = {ìd, if, while, repeat}
FOLLOW(tipo) = {:}
FOLLOW(lista_id) = {;}
FOLLOW(comandos) = {END }
FOLLOW(comando') = {END}
FOLLOW(comando) = {END}
FOLLOW(seleção) = {END}
FOLLOW(seleção') = {END}
FOLLOW(instruções) = {END, until }
FOLLOW(atribuição) = {END}
FOLLOW(repetição) = {END}
FOLLOW(condição) = {then, BEGIN, id, if, while, repeat, ;}
FOLLOW(elemento) = {=, !=, >=, <=, then, BEGIN, id, if, while, repeat, ;}
FOLLOW(relop) = {id, caracteres, numeroInt, numeroFloat}
FOLLOW(expressão) = {;}
FOLLOW(expressão') = {;}
FOLLOW(ariop) = {;}
FOLLOW(constante) = {=, !=, >=, <=, then, BEGIN, id, if, while, repeat, ;}

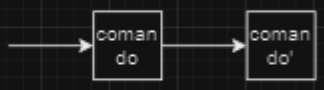
GRAFOS SINTÁTICOS



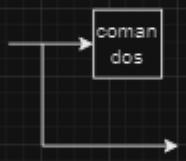
Grafo de "lista_id"



Grafo de "comandos"

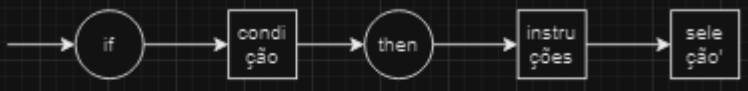


Grafo de "comando'"

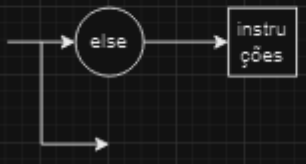


Grafo de "comando"

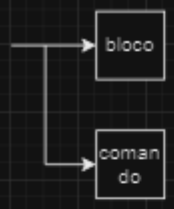
Grafo de "seleção"



Grafo de "seleção'"

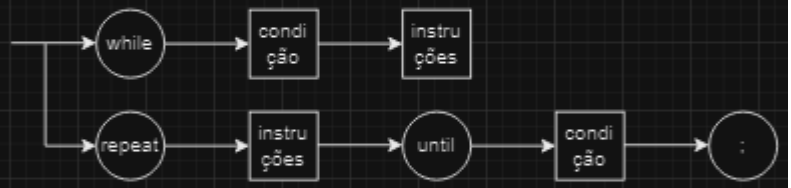


Grafo de "instruções"

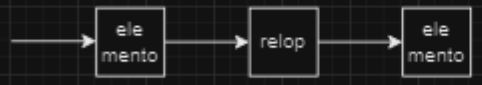


Grafo de "atribuição"

Grafo de "repetição"

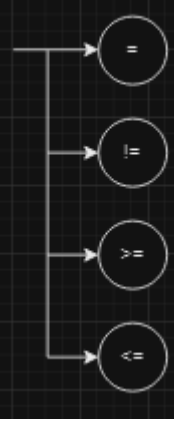


Grafo de "condição"

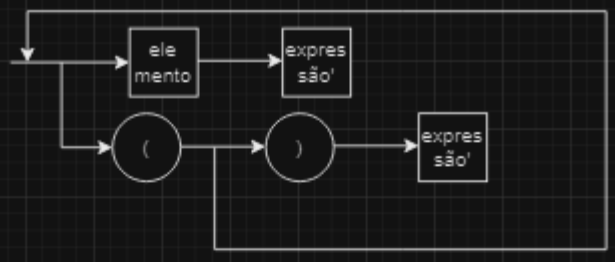


Grafo de "elemento"

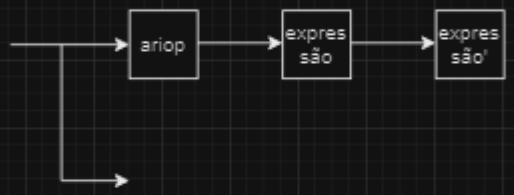
Grafo de "relop"



Grafo de "expressão"



Grafo de "expressão'"



Grafo de "ariop"

Grafo de "constante"