Projeto de Construção de compiladores - Parte 1

Gabriel Kato Gomes

Leandro Fontellas Laurito

Vitor Henrique Oliveira Silva

Projeto da Linguagem

1. Definição da gramática livre de contexto (CLG) com as estruturas da linguagem especificada

   G = (T, V, P ,programa)

   T = {main, inicio, fim, ->, ;, ,, int, char, float, caso, entao, senao, enquanto, faca, repita, ate, =, (,), id, numero_int, numero_float, const_char, op_rel, op_add, op_mul, ε}

   V = {programa, bloco, decl_vars, lista_decl, decl_var, tipo, lista_ids, cont_lista_ids, seq_comandos, comando, corpo, atribuicao, selecao, repeticao, condicao, expr}

   P = {

   programa → main id () bloco
   bloco → inicio decl_vars seq_comandos fim
   decl_vars → lista_decl | ε
   lista_decl → decl_var lista_decl | decl_var
   decl_var → lista_ids -> tipo ;
   tipo → int | char | float
   lista_ids → id, lista_ids | id
   seq_comandos → comando seq_comandos | ε
   comando → atribuicao | selecao | repeticao
   atribuicao → id = expr ;
   selecao → caso (condicao) entao corpo
   | caso (condicao) entao corpo senao corpo
   repeticao → enquanto (condicao) faca corpo
   | repita corpo ate (condicao);
   corpo → comando | bloco
   condicao → expr op_rel expr
   expr → expr op_add expr
   | expr op_mul expr
   | expr ^ expr

| expr → id
| numero_int
| numero_float
| const_char
| (expr)

op_rel → == | != | < | > | <= | >=

op_add → + | -

op_mul → * | /

}

2. Identificação dos tokens usados na gramática

Token	**Nome do Token**	**Tipo do atributo**
main	main	-
Qualquer ID	ID	Nome do ID
inicio	inicio	-
fim	fim	-
char	tipo	char
int	tipo	int
float	tipo	float
->	pontuação	declaracao
;	pontuação	fimexp
,	pontuação	mulvars
caso	caso	-
entao	entao	-
senao	senao	-
enquanto	enquanto	-
faca	faca	-
repita	repita	-
ate	ate	-
=	atribuição	-
==	oprel	EQ
!=	oprel	NE

<	oprel	LT
>	oprel	GT
<=	oprel	LE
>=	oprel	GE
+	oparit	SUM
-	oparit	SUB
*	oparit	MUL
/	oparit	DIV
^	oparit	EXP
(	oparit	PARESQ
)	oparit	PARDIR
Qualquer número	numero	Retorna o número
comentarios	comentario	-
quaisquer ws	espaço	-

3. Definição dos padrões (expressões regulares) de cada token (inclusive os tokens especiais)

   digito: [0-9]
   digitos: digito digito*
   letra: [A-Z a-z]
   letra_: [A-Z a-z _]
   main: main
   inicio: inicio
   fim: fim
   int: int
   char: char
   float: float
   caso: caso
   entao: entao
   senao: senao
   enquanto: enquanto
   faca: faca
   repita: repita
   ate: ate
   ID: letra_ (letra_ | digito)*

```
oprel: [== != < > <= >=]
oparit: [+ - * / ^]
atribuicao: =
abre_parenteses: \(
fecha_parenteses: \)
pontuacao: [, ;]
->: ->
ws : (" " | \t | \n)
comentario: \/*[^*]*\*/
```

Análise Léxica

Diagrama de transição para cada token;

Diagrama para oprel:

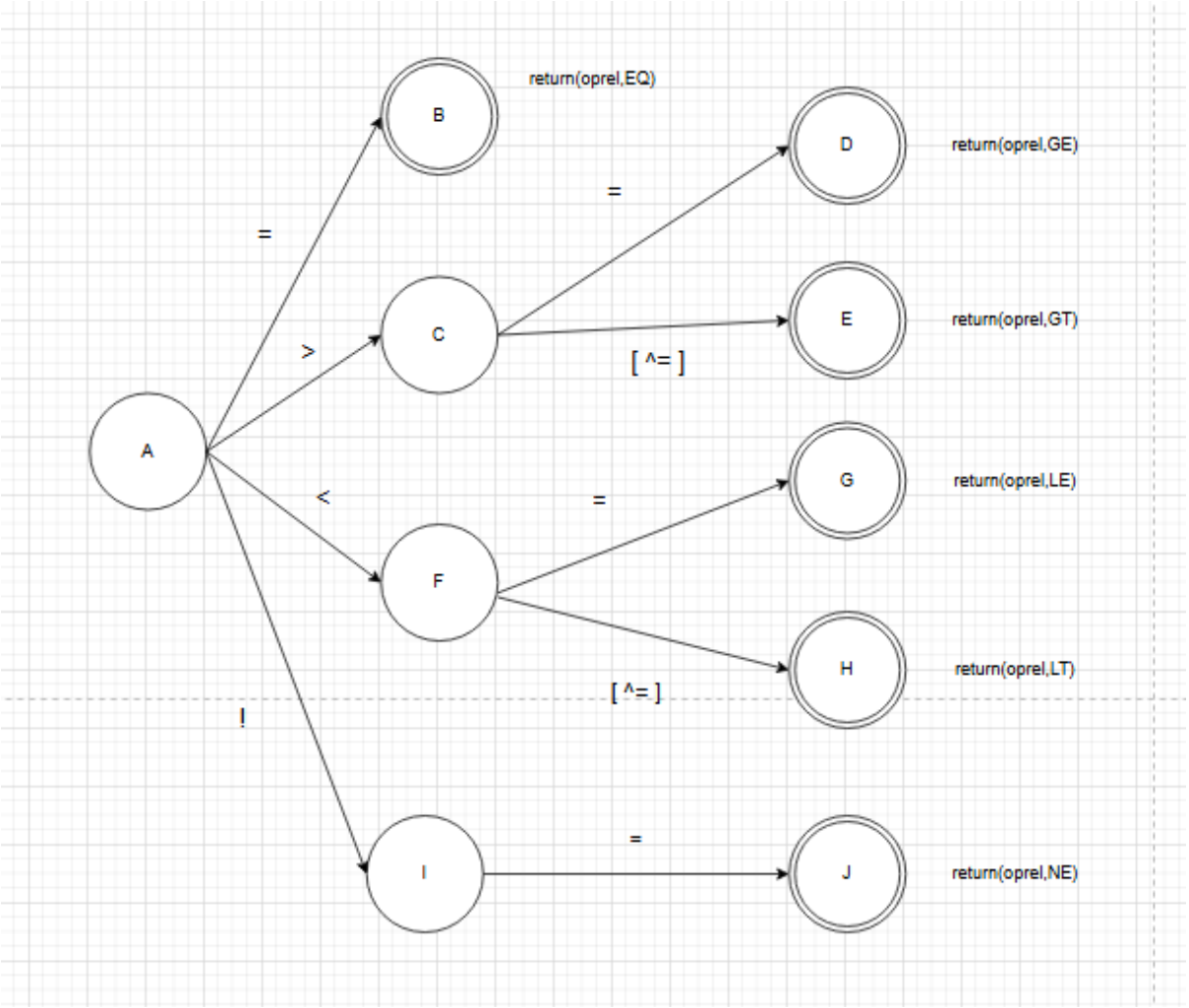


Diagrama para operações aritméticas:

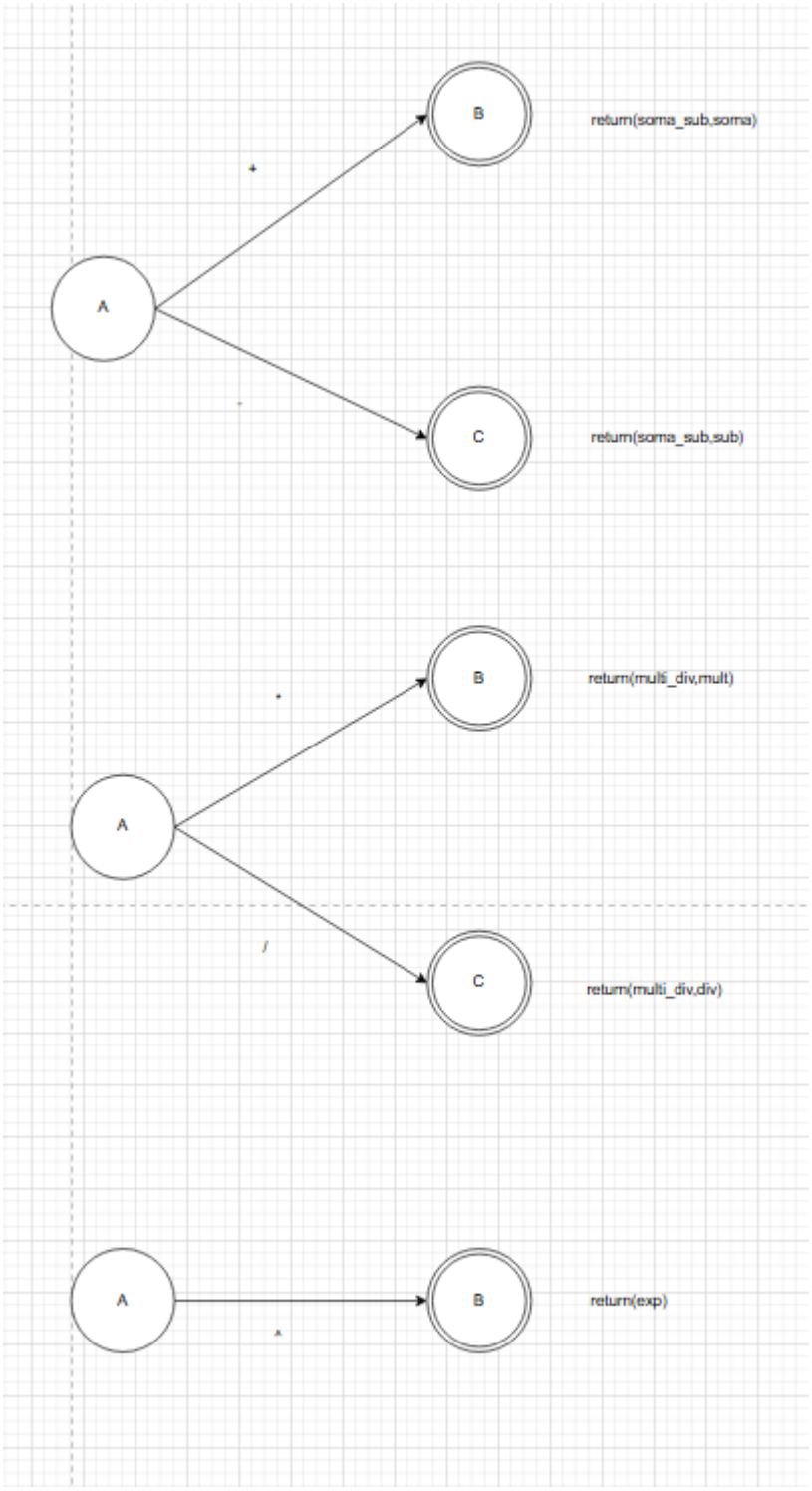


Diagrama para id e caractere:

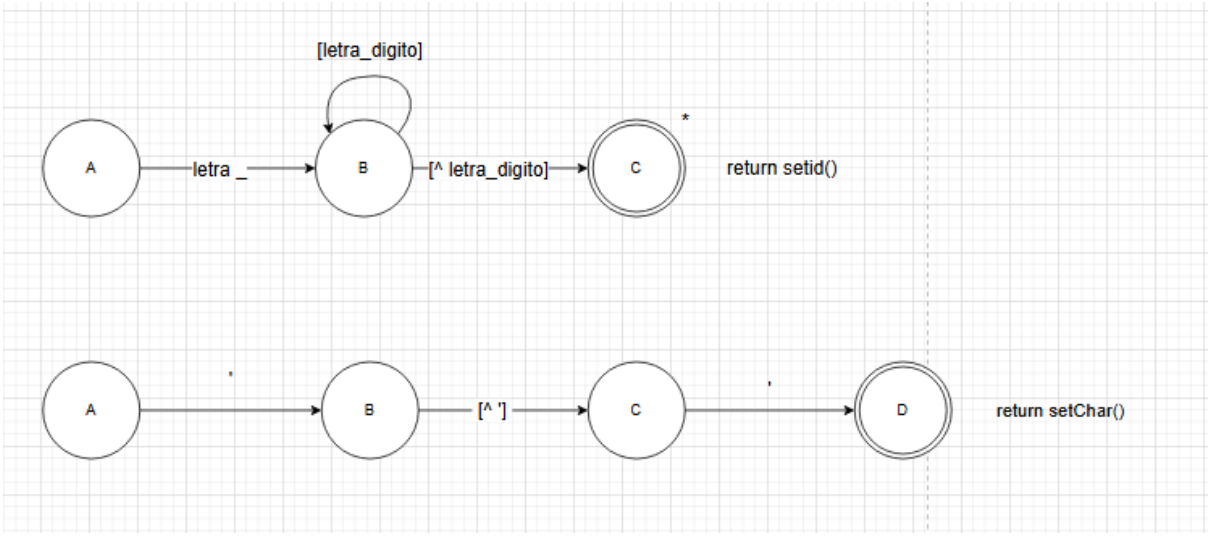


Diagrama para parênteses:

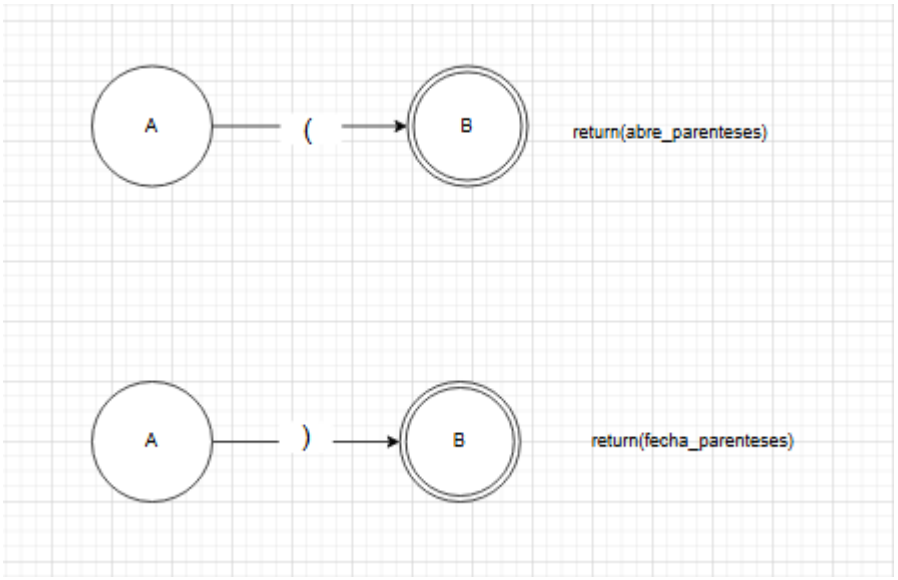


Diagrama para pontuação e atribuição:

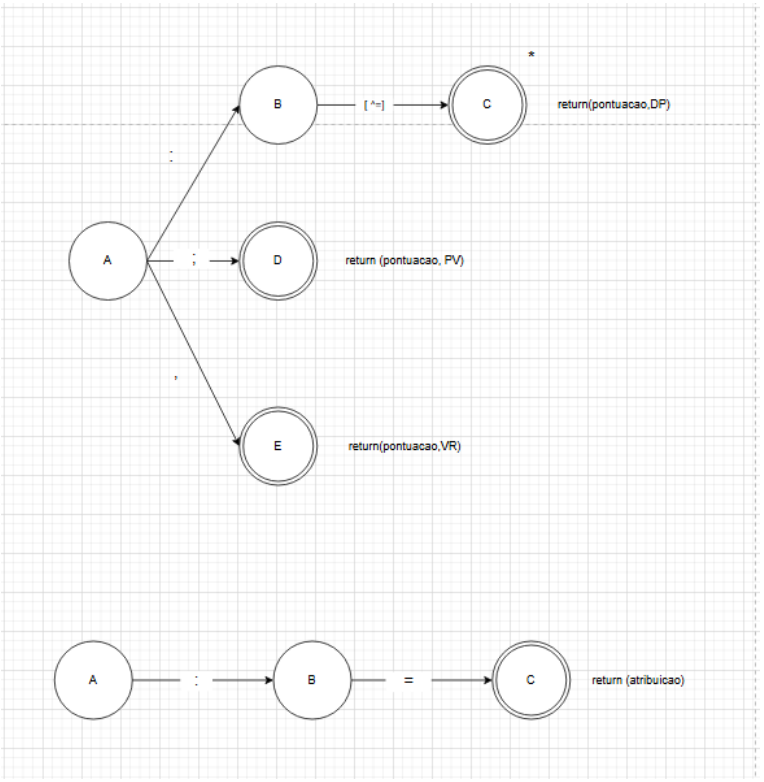


Diagrama para número e comentário:

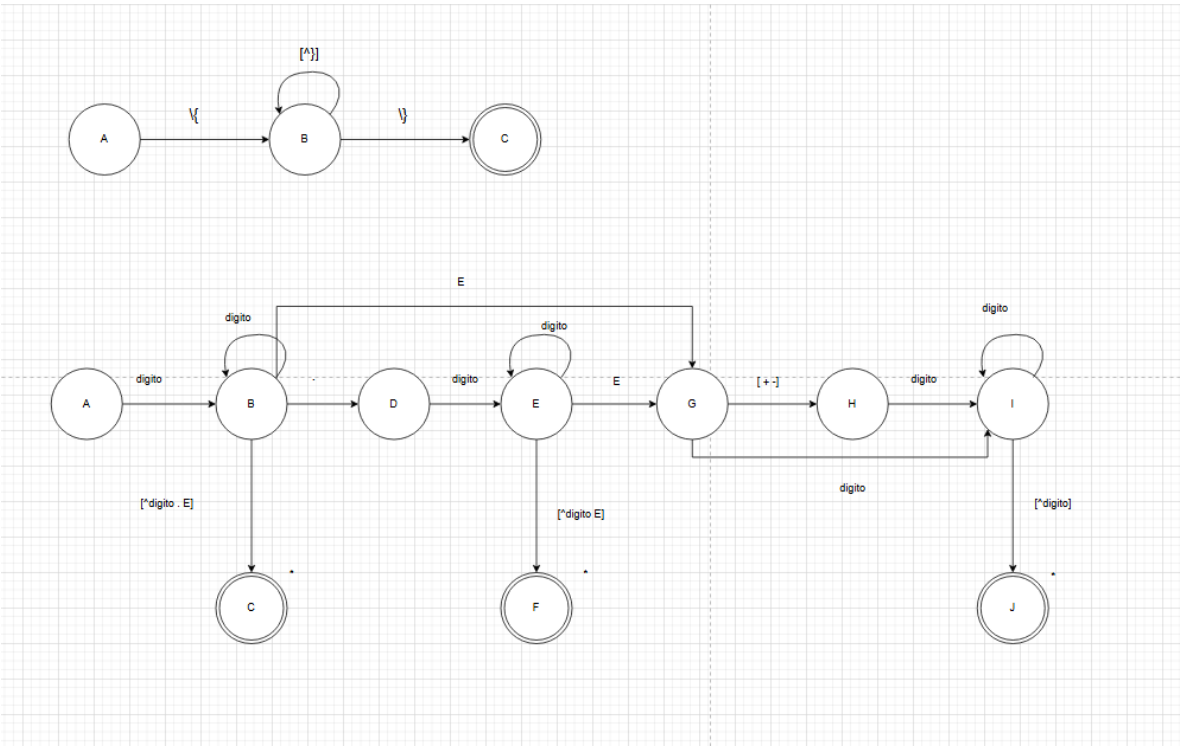


Diagrama unificado não determinístico:

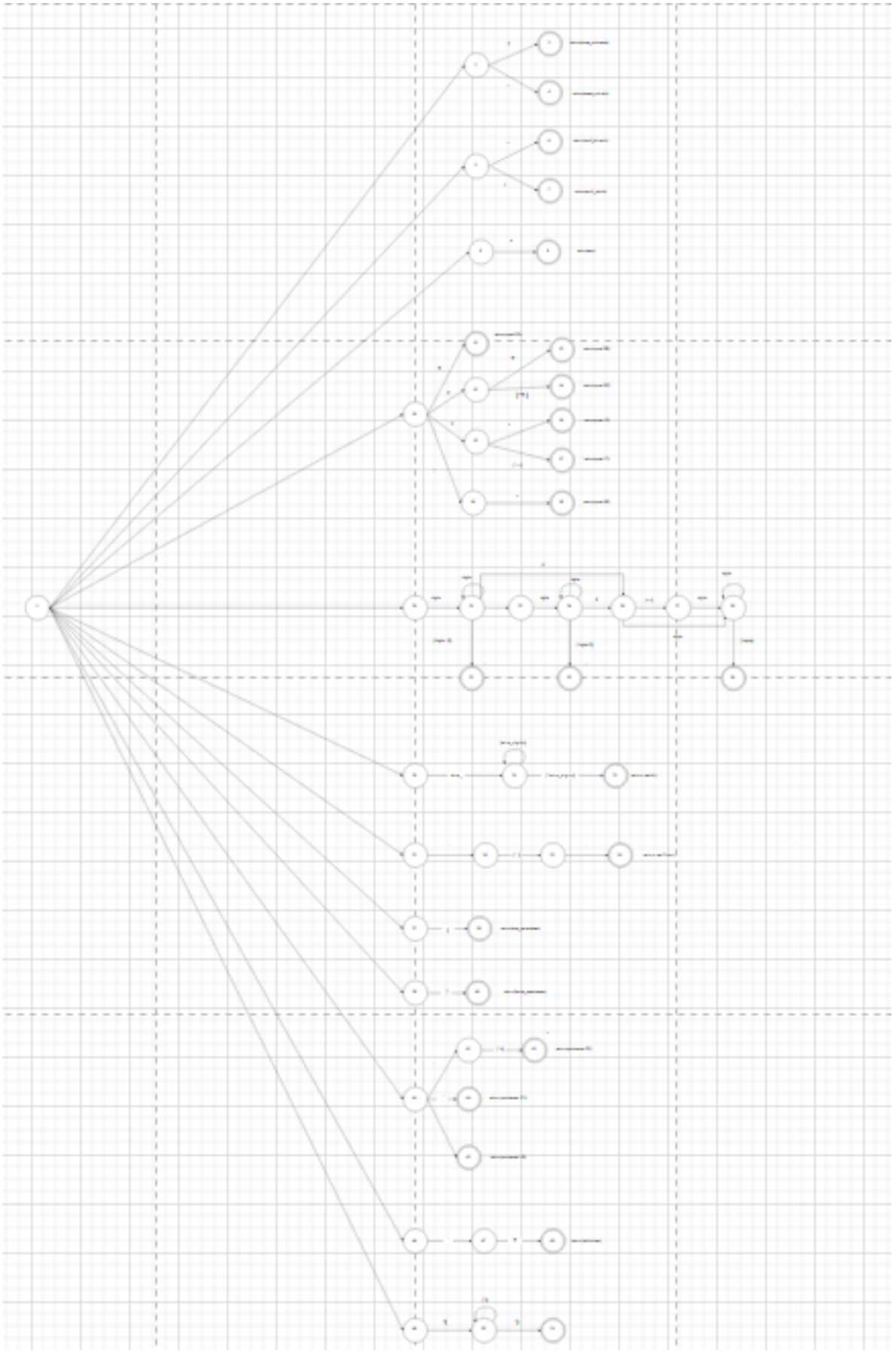
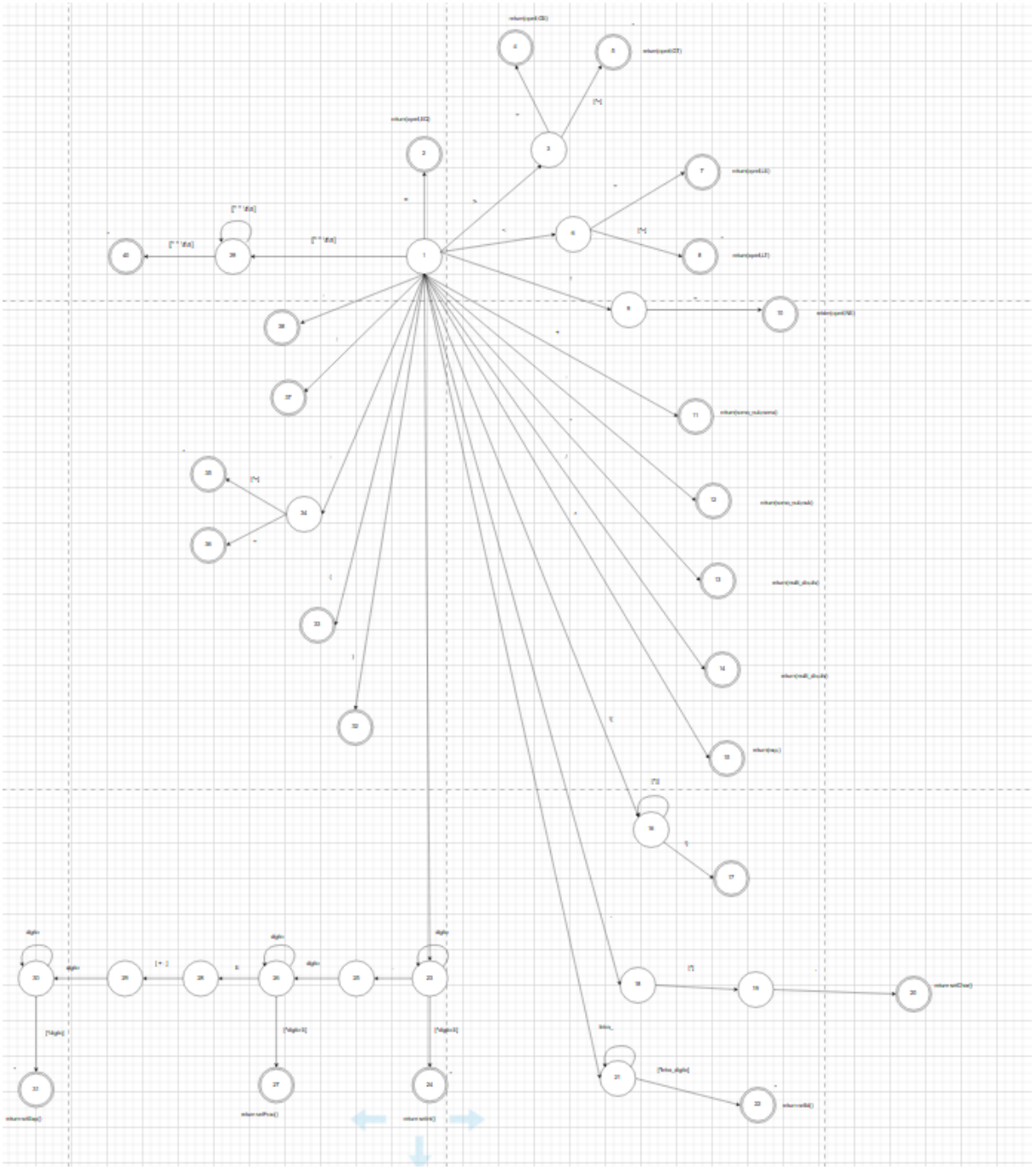


Diagrama de transição unificado determinìstico:



Análise Sintática - Etapa 3

Ajustes necessários para que a GLC da linguagem seja LL(1):
G = (T, V, P ,programa)

T = {main, inicio, fim, ->, ;, ,, int, char, float, caso, entao, senao, enquanto, faca, repita, ate, =, (,), id, numero_int, numero_float, const_char, op_rel, op_add, op_mul, ε}

V = {programa, bloco, decl_vars, lista_decl, decl_var, tipo, lista_ids, cont_lista_ids, seq_comandos, comando, corpo, atribuicao, selecao, senao_opc, repeticao, condicao, expr, termo, fator, unario}

P = {
programa → main id () bloco
bloco → inicio decl_vars seq_comandos fim
decl_vars → lista_decl | ε
lista_decl → decl_var lista_decl | decl_var
decl_var → lista_ids -> tipo ;
tipo → int | char | float
lista_ids → id cont_lista_ids
cont_lista_ids → , lista_ids | ε
seq_comandos → comando seq_comandos | ε
comando → atribuicao | selecao | repeticao
atribuicao → id = expr ;
selecao → caso (condicao) entao corpo senao_opc
senao_opc → senao corpo | ε
repeticao → enquanto (condicao) faca corpo
| repita corpo ate (condicao) ;
corpo → comando | bloco
condicao → expr op_rel expr
expr → termo expr'
expr'→ op_add termo expr' | ε
termo → fator termo'
termo'→ op_mul fator termo' | ε
fator → unario fator'
fator'→ ^ unario fator' | ε
unario → id
| numero_int
| numero_float
| const_char
| (expr)
op_rel → == | != | < | > | <= | >=
op_add → + | -
op_mul → * | /
}

Calculo dos FIRST para os simbolos da gramatica:

FIRST(programa) = { main };

FIRST(bloco) = { inicio };

FIRST(decl_vars) = FIRST(decl_var) U { ε } = FIRST(lista_ids) U { ε } = { id, ε };

FIRST(lista_decl) = FIRST(decl_var) = { id, ε };

FIRST(tipo) = { int, char, float };

FIRST(lista_ids) = { id };

FIRST(cont_lista_ids) = { ,, ε };

FIRST(seq_comandos) = FIRST(comando) U { ε } = FIRST(atribuicao) U FIRST(selecao) U FIRST(repeticao) U FIRST(bloco) U { ε } = { id, caso, enquanto, repita, inicio, ε };

FIRST(comando) = FIRST(atribuicao) U FIRST(selecao) U FIRST(repeticao) = { id, caso, enquanto, repita };

FIRST(atribuicao) = { id };

FIRST(selecao) = { caso };

FIRST(repeticao) = { enquanto, repita };

FIRST(bloco) = { inicio };

FIRST(senao_opc) = { senao };

FIRST(corpo) = FIRST(comando) U FIRST(bloco) = { id, caso, equanto, repita, inicio };

FIRST(condicao) = FIRST(expr) = FIRST(termo) = FIRST(fator) = FIRST(unario) = FIRST(id) U FIRST(numero_int) U FIRST(numero_float) U FIRST(const_char) U FIRST((expr)) = { id, numero_int, numero_float, const_char, (};

FIRST(expr) = FIRST(termo) = FIRST(fator) = FIRST(unario) = { id, numero_int, numero_float, const_char, (};

FIRST(expr') = FIRST(op_add) U { ε } = FIRST(+) U FIRST(-) U { ε } = { +, -, ε };

FIRST(termo) = FIRST(fator) = FIRST(unario) = { id, numero_int, numero_float, const_char, (};

FIRST(termo') = FIRST(op_mul) U { ε } = FIRST(*) U FIRST (/) U { ε } = { *, /, ε };

FIRST(fator) = FIRST(unario) = { id, numero_int, numero_float, const_char, (};

FIRST(fator') = FIRST(^) U { ε } = { ^, ε };

FIRST(unario) = { id, numero_int, numero_float, const_char, (};

FIRST(op_rel) = FIRST(==) U FIRST(!=) U FIRST(<) U FIRST(>) U FIRST(<=) U FIRST(>=) = { ==, !=, <, >, <=, >= };

FIRST(op_add) = FIRST(+) U FIRST(-) = { +, - };

FIRST(op_mul) = FIRST(*) U FIRST(/) = { *, / };

Calculo dos FOLLOW para os simbolos da gramatica:

FOLLOW(programa) = { $ };

FOLLOW(main) = { id };

FOLLOW(id) = FIRST(cont_lista_ids) U FOLLOW(unario) U { (, = } = { „, ^, *, /, +, -, ;,), (, = };

FOLLOW(unario) = FIRST(fator') U FOLLOW(fator) = { ^, *, /, +, -, ;,)};

FOLLOW(fator) = FIRST(termo') U FOLLOW(termo) = { *, /, +, -, ;,)};

FOLLOW(termo) = FIRST(expr') U FOLLOW(expr) = { +, -, ;,) };

FOLLOW(expr) = FIRST(op_rel) U FOLLOW(condicao) U { ; } = { ;,) };

FOLLOW(condicao) = {) };

FOLLOW(( ) = FIRST(condicao) U FIRST(expr) U {) } = { id, numero_int, numero_float, const_char, (,) };

FOLLOW( )) = FIRST(bloco) U { entao, faca, ; } U FOLLOW(unario) = { inicio, entao, faca, ^, *, /, +, -, ;,)};

FOLLOW(bloco) = FOLLOW(programa) U FOLLOW(corpo) = { $, senao, id, caso, enquanto, repita, inicio, ate };

FOLLOW(corpo) = FIRST(senao_opc) U FOLLOW(selecao) U FOLLOW(repeticao) U { ate } = { senao, id, caso, enquanto, repita, inicio, ate };

FOLLOW(selecao) = FOLLOW(comando) = {id, caso, enquanto, repita, inicio };

FOLLOW(comando) = FIRST(seq_comandos) - { ε } = {id, caso, enquanto, repita, inicio };

FOLLOW(repeticao) = FOLLOW(comando) = {id, caso, enquanto, repita, inicio };

FOLLOW(inicio) = FIRST(decl_vars) - { ε } = { id };

FOLLOW(decl_vars) = FIRST(seq_comandos) - { ε } = { id, caso, enquanto, repita, inicio };
FOLLOW(seq_comandos) = { fim };

FOLLOW(fim) = FOLLOW(bloco) = { $, senao, id, caso, enquanto, repita, inicio, ate };

FOLLOW(lista_decl) = FOLLOW(decl_vars) = { id, caso, enquanto, repita, inicio };

FOLLOW(decl_var) = FIRST(lista_decl) U FOLLOW(lista_decl) = { id, caso, enquanto, repita, inicio };

FOLLOW(lista_ids) = { -> };

FOLLOW(cont_lista_ids) = FOLLOW(lista_ids) = { -> };

FOLLOW(->) = FIRST(tipo) = { int, char, float };

FOLLOW(tipo) = { ; };

FOLLOW(;) = FOLLOW(decl_var) U FOLLOW(atribuicao) U FOLLOW(repeticao) = { id, caso, enquanto, repita, inicio };

FOLLOW(atribuicao) = FOLLOW(comando) = { id, caso, enquanto, repita, inicio };

FOLLOW(int) = FOLLOW(tipo) = { ; };

FOLLOW(char) = FOLLOW(tipo) = { ; };

FOLLOW(float) = FOLLOW(tipo) = { ; };

FOLLOW(,) = FIRST(lista_ids) = { id };

FOLLOW(=) = FIRST(expr) = { id, numero_int, numero_float, const_char, (};

FOLLOW(caso) = { (};

FOLLOW(entao) = FIRST(corpo) = { id, caso, equanto, repita, inicio };

FOLLOW(senao_opc) = FOLLOW(selecao) = {id, caso, enquanto, repita, inicio };

FOLLOW(enquanto) = { (};

FOLLOW(faca) = FIRST(corpo) = { id, caso, equanto, repita, inicio };

FOLLOW(repita) = FIRST(corpo) = { id, caso, equanto, repita, inicio };

FOLLOW(ate) = { (};

FOLLOW(op_rel) = FIRST(expr) = { id, numero_int, numero_float, const_char, (};

FOLLOW(expr') = FOLLOW(expr) = { ;,) };

FOLLOW(termo') = FOLLOW(termo) = { **+**, -, ;,) };

FOLLOW(fator') = FOLLOW(fator) = { *, /, +, -, ;,)};

FOLLOW(unario') = FOLLOW(unario) = { ^, *, /, +, -, ;,)};

FOLLOW(^) = FIRST(unario) = { id, numero_int, numero_float, const_char, (};

FOLLOW(numero_int) = FOLLOW(unario) = { ^, *, /, +, -, ;,)};

FOLLOW(numero_float) = FOLLOW(unario) = { ^, *, /, +, -, ;,)};

FOLLOW(const_char) = FOLLOW(unario) = { ^, *, /, +, -, ;,)};

FOLLOW(== | != | < | > | <= | >=) = FOLLOW(op_rel) = { id, numero_int, numero_float, const_char, (};

FOLLOW(op_add) = FIRST(termo) = { id, numero_int, numero_float, const_char, (};

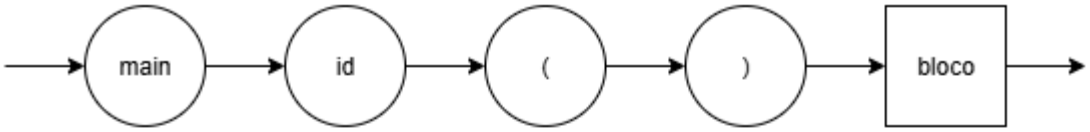
FOLLOW(op_mul) = FIRST(fator) = { id, numero_int, numero_float, const_char, (};

FOLLOW(+ | -) = FOLLOW(op_add);

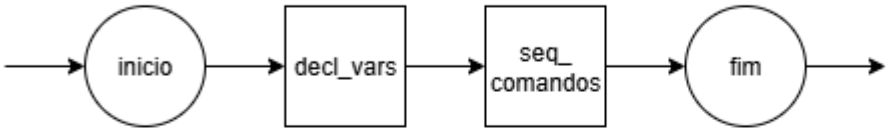
FOLLOW(* | /) = FOLLOW(op_mul);

Grafo sintático (link)

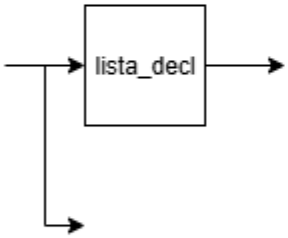
programa → main id () bloco



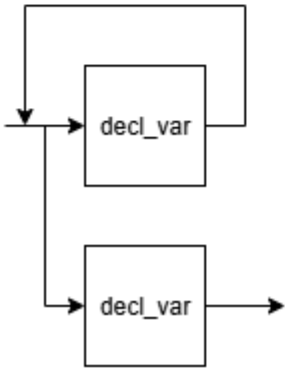
bloco → inicio decl_vars seq_comandos fim



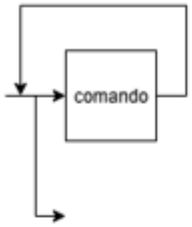
decl_vars → lista_decl | ε



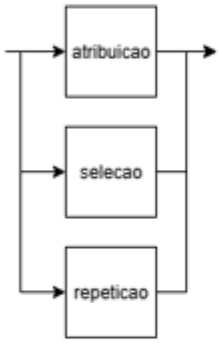
lista_decl → decl_var lista_decl | decl_var



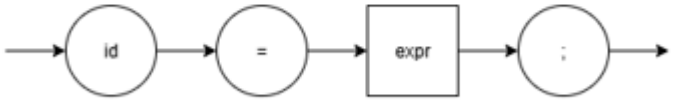
seq_comandos → comando seq_comandos | ε



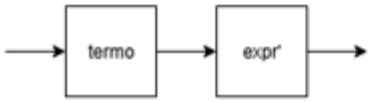
comando → atribuicao | selecao | repeticao



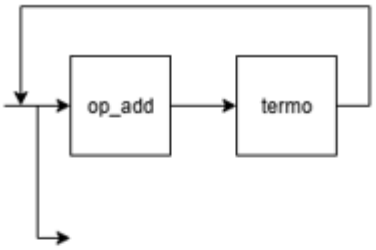
atribuicao → id = expr ;



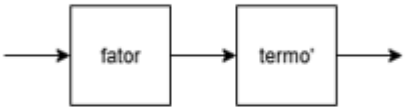
expr → termo expr'



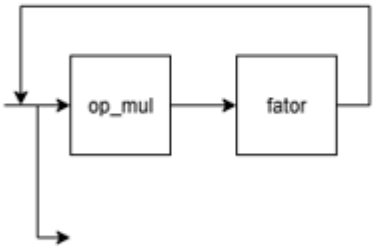
expr' → op_add termo expr' | ε



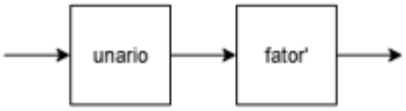
termo → fator termo'



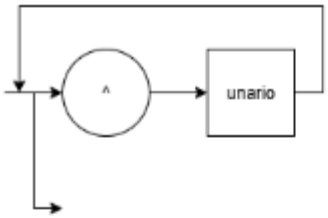
termo' → op_mul fator termo' | ε



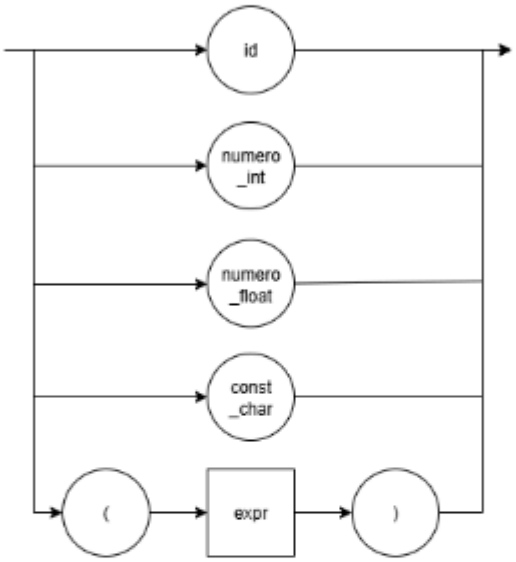
fator → unario fator'



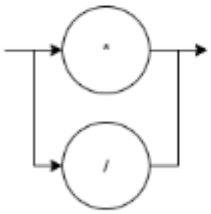
fator' → ^ unario fator' | ε



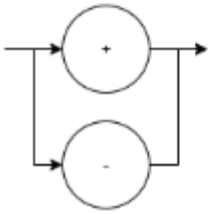
unario → id | numero_int | numero_float | const_char | (expr)



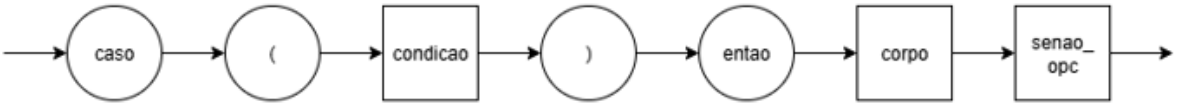
op_mul → * | /



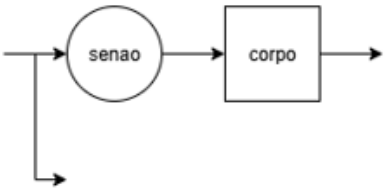
op_add → + | -



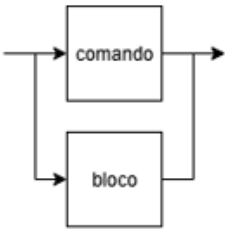
selecao → caso (condicao) entao corpo senao_opc



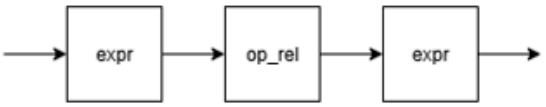
senao_opc → senao corpo | ε



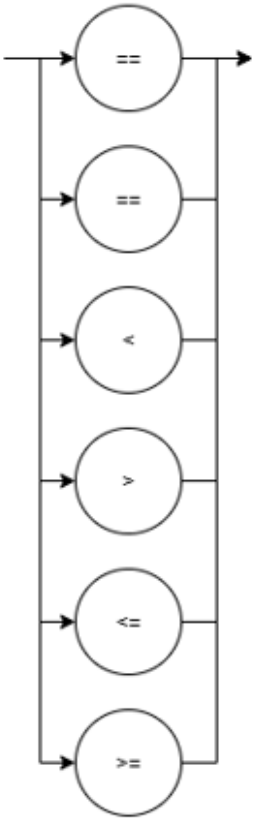
corpo → comando | bloco



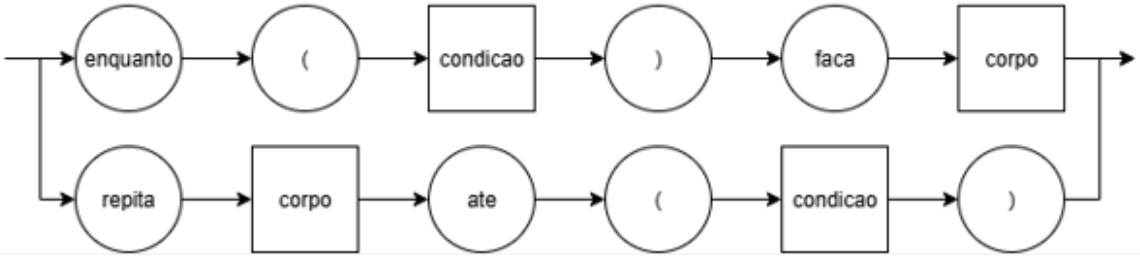
condicao → expr op_rel expr



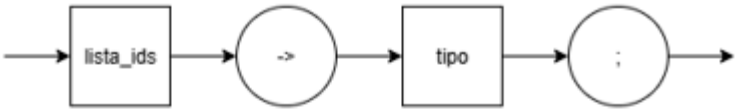
op_rel ⟶ == | != | < | > | <= | >=



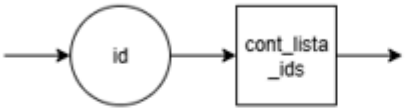
repeticao ⟶ enquanto (condicao) faca corpo | repita corpo ate (condicao) ;



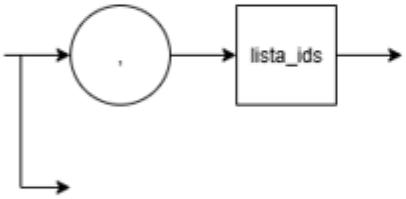
decl_var → lista_ids -> tipo ;



lista_ids → id cont_lista_ids



cont_lista_ids → , lista_ids | ε



tipo → int | char | float