**Relatório de Projeto**

Silvano Junior (12011BCC042)

Thais Damasceno (11721BCC007)

Vitor Yuji (11921BCC021)

08.09.2024

Bacharelado Ciência da Computação - Construção de Compiladores

**Etapa 1 - Projeto da Linguagem**

**Especificação da linguagem**

**– Definição da gramática livre de contexto (GLC) com as estruturas da linguagem especificada**

- Estrutura principal;

- Componentes Básicos;

- Comandos e Estruturas de Controle;

- Condições e Expressões;

- Identificadores e Comentários.

A gramática livre do contexto G é definida como:

G = (V, T, P, S)

Onde:

V (Variáveis/Não-terminais)

T (Terminais)

P (Regras de Produção)

S (Símbolo inicial)

V =

Programa, Bloco, Declaracoes, DeclaracaoVar, Tipo, ListaIds, Comandos, Comando, Atribuicao, ComandoSelecao, ComandoElse, ComandoRepeticao, Condicao, OperadorRelacional, Expressao, Termo, Fator

T =

main, begin, end, int, char, float, if, then, else, while, do, repeat, until, :=, :, ;, ,, ==, !=, <, >, <=, >=, +, -, \*, /, \*\*, (, ), ID, INT\_CONST, FLOAT\_CONST, CHAR\_CONST, {Comentario}

P = {

Programa → main ID Bloco

Bloco → begin Declaracoes Comandos end

Declaracoes → DeclaracaoVar Declaracoes | ε

DeclaracaoVar → Tipo : ListaIds ;

Tipo → int | char | float

ListaIds → tipo -> ID , ListaIds | tipo -> ID

Comandos → Comando Comandos | ε

Comando → Atribuicao | ComandoSelecao | ComandoRepeticao | Bloco | {Comentario}

Atribuicao → ID := Expressao ;

ComandoSelecao → if ( Condicao ) then Comando ComandoElse

ComandoElse → else Comando | ε

ComandoRepeticao → while( Condicao ) do Comando | repeat Comando until ( Condicao ) ;

Condicao → Expressao OperadorRelacional Expressao

OperadorRelacional → == | != | < | > | <= | >=

Expressao → Expressao + Termo | Expressao - Termo | Termo

Termo → Termo \* Fator | Termo / Fator | Termo \*\* Fator | Fator

Fator → ( Expressao ) | ID | INT\_CONST | FLOAT\_CONST | CHAR\_CONST

}

S Programa

**– Identificação dos tokens usados na gramática**

**● Apresentar uma tabela com o nome e o tipo de atributo que será retornado (quando aplicável) para cada token**

| LEXEMA | NOME DO TOKEN | valor do atributo |
| --- | --- | --- |
| main | main | - |
| begin | begin | - |
| end | end | - |
| int | tipo | INT |
| char | tipo | CHAR |
| float | tipo | FLOAT |
| if | if | - |
| then | then | - |
| else | else | - |
| while | while | - |
| do | do | - |
| repeat | repeat | - |
| until | until | - |
| : | pont | COLON |
| ; | pont | SEMICOLON |
| , | pont | COMMA |
| := | pont | ASSIGN |
| () | pont | PAREN |
| -> | associacao | ARR |

| == | relop | EQ |
| --- | --- | --- |
| != | relop | NEQ |
| < | relop | LT |
| > | relop | GT |
| <= | relop | LE |
| >= | relop | GE |
| + | ariop | ADD |
| - | ariop | SUB |
| \* | ariop | MUL |
| / | ariop | DIV |
| \*\* | ariop | EXP |
| Qualquer id | lista\_ids | Posição na tabela de símbolos |
| qualquer número | INT\_CONST | Posição na tabela de símbolos |
| qualquer constante float | FLOAT\_CONST | Posição na tabela de símbolos |
| Qualquer constante char | CHAR\_CONST | Posição na tabela de símbolos |
| Qualquer ws | - | - |
| Qualquer comentário | - | - |

**– Definição dos padrões (expressões regulares) de cada token (inclusive os tokens especiais)**

ID → [A-Za-z\_][A-Za-z0-9\_]\*

relop → == | != | < | > | <= | >=

ariop → + | - | \* | / | \*\*

pont → := | : | ; | , | ( | )

tipo → INT | CHAR | FLOAT

INT\_CONST → -?([0-9]{1,5})

FLOAT\_CONST → [0-9]+(.[0-9]+)?([Ee][-]?[0-9]+)?

CHAR\_CONST →'[ID]'

associacao → ->

lista\_ids → tipo->ID (,ID)\*

MAIN → main

BEGIN → begin

END → end

IF → if

THEN → then

ELSE → else

WHILE → while

DO → do

REPEAT → repeat

UNTIL → until

COMMENT → \{[^}]\*\}

ws → (‘ ‘ | \t | \n)\*

**Etapa 2 - Análise Léxica**

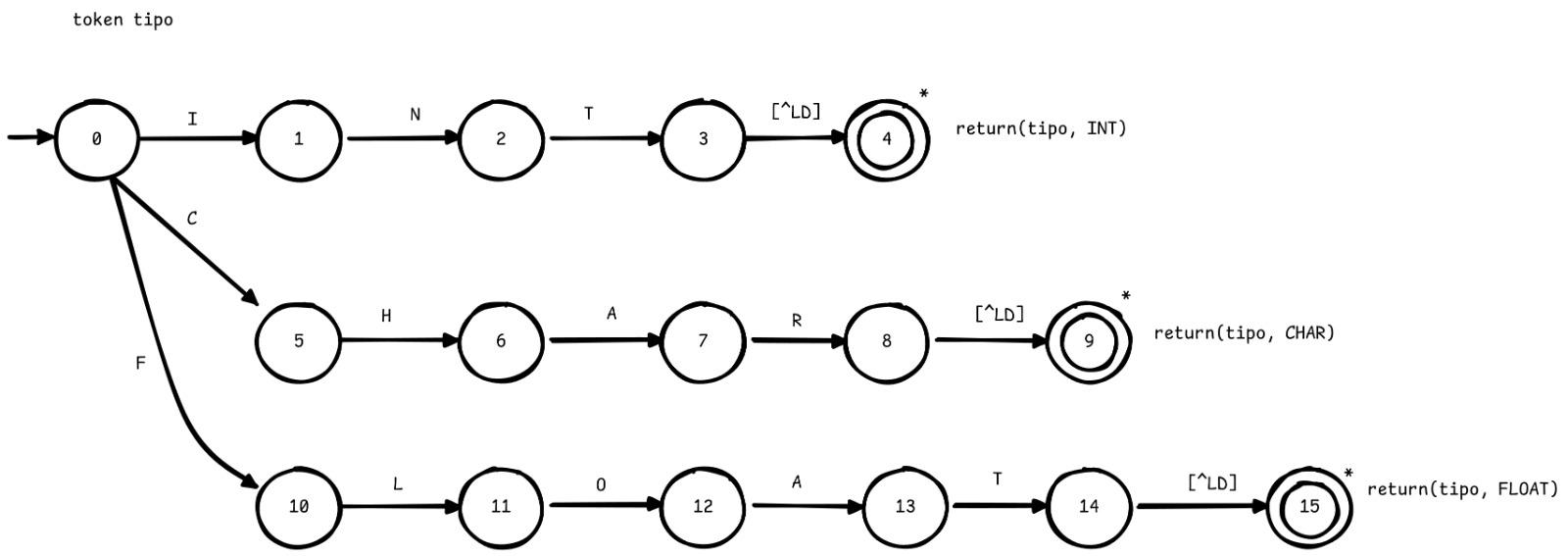
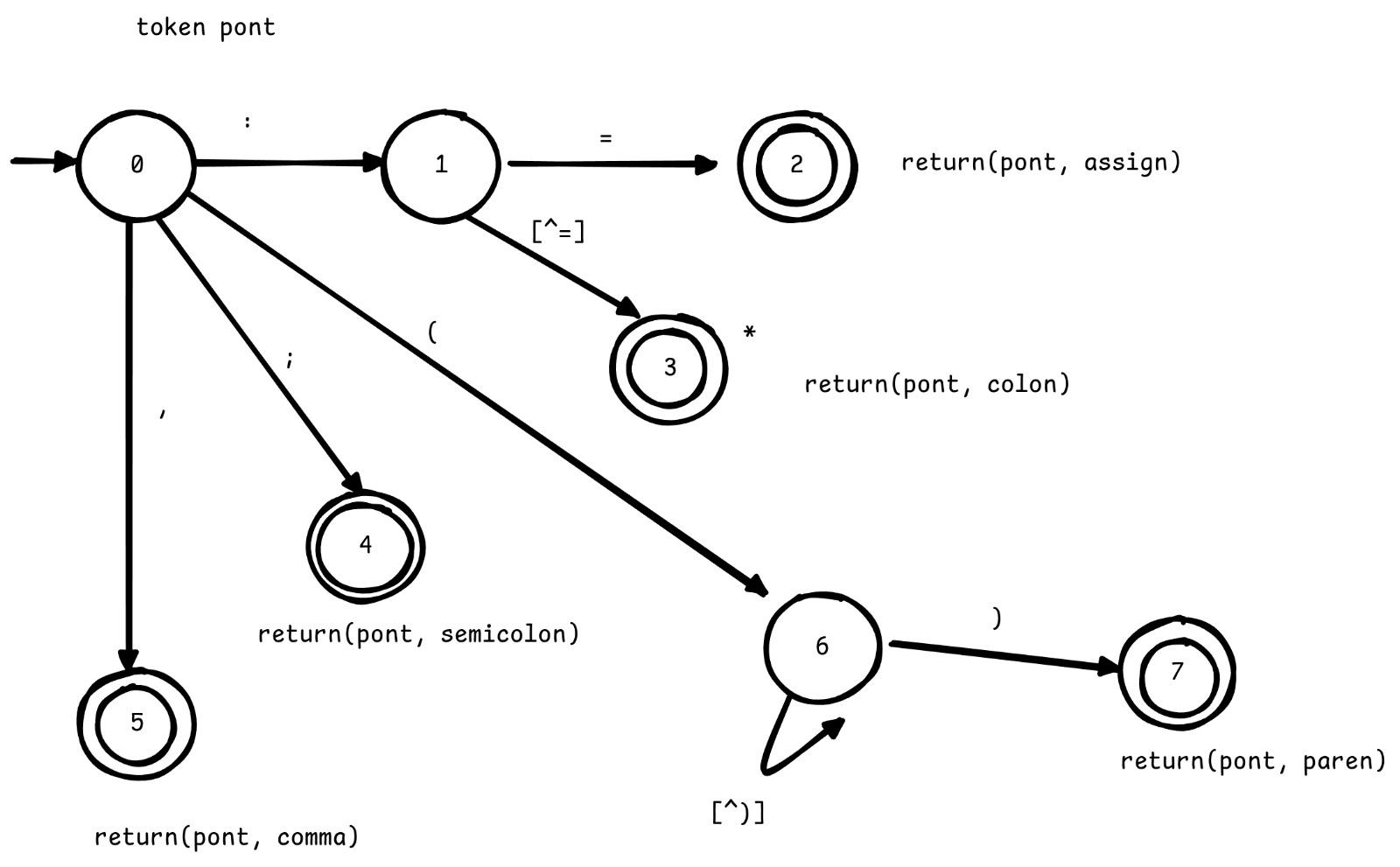
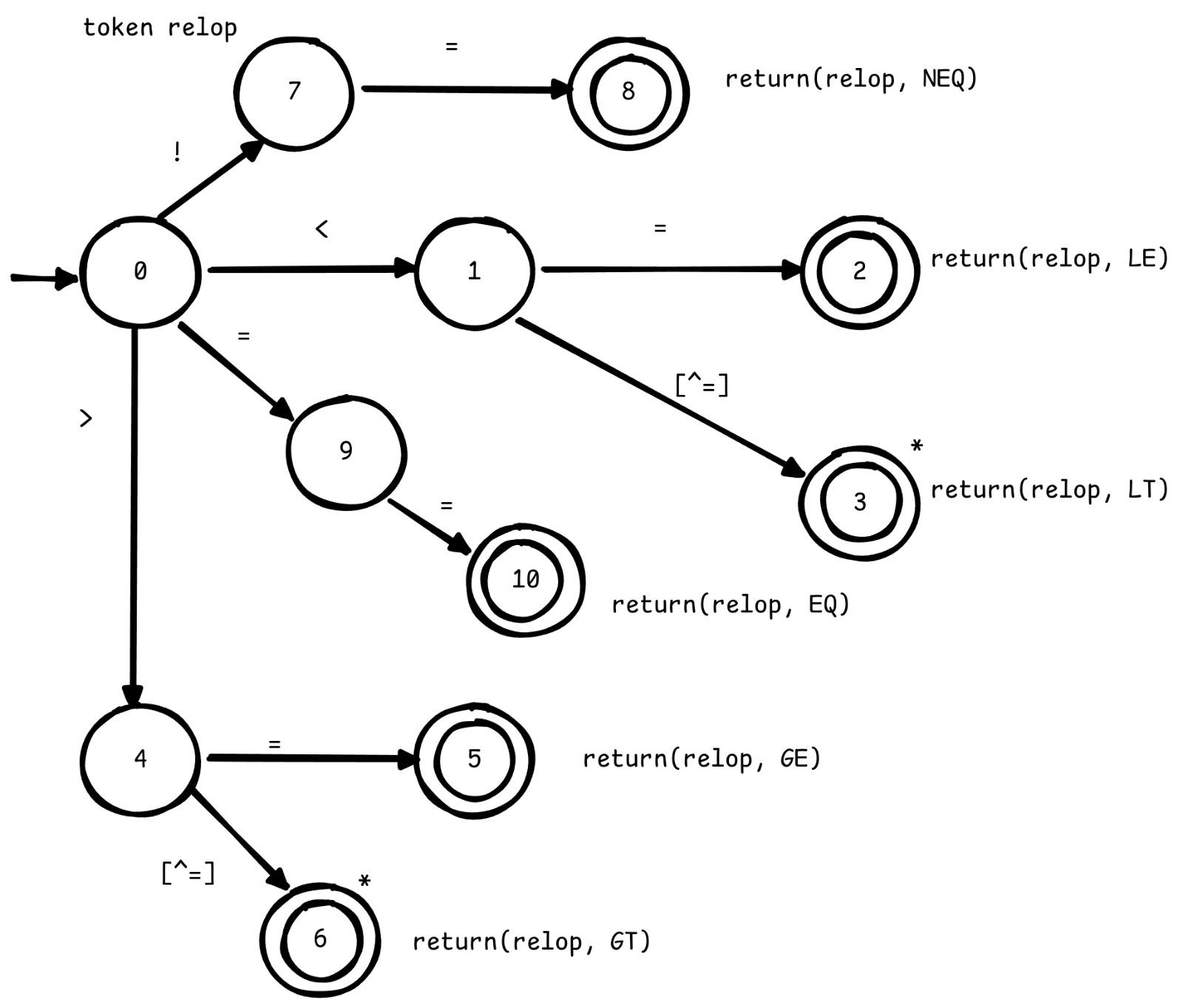
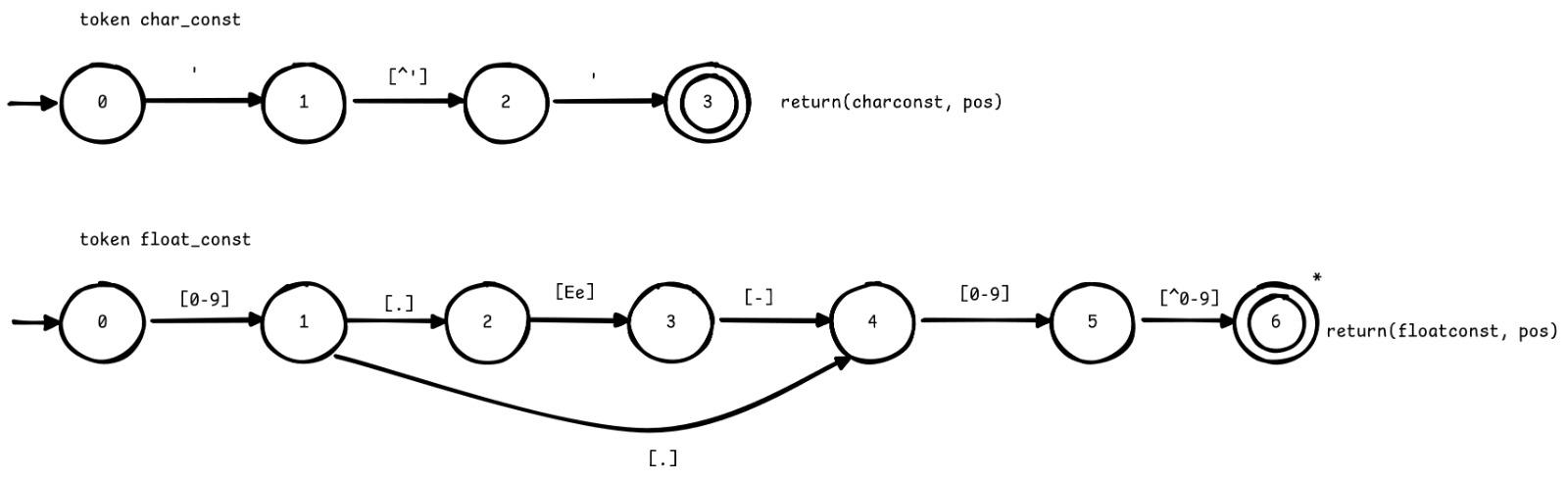
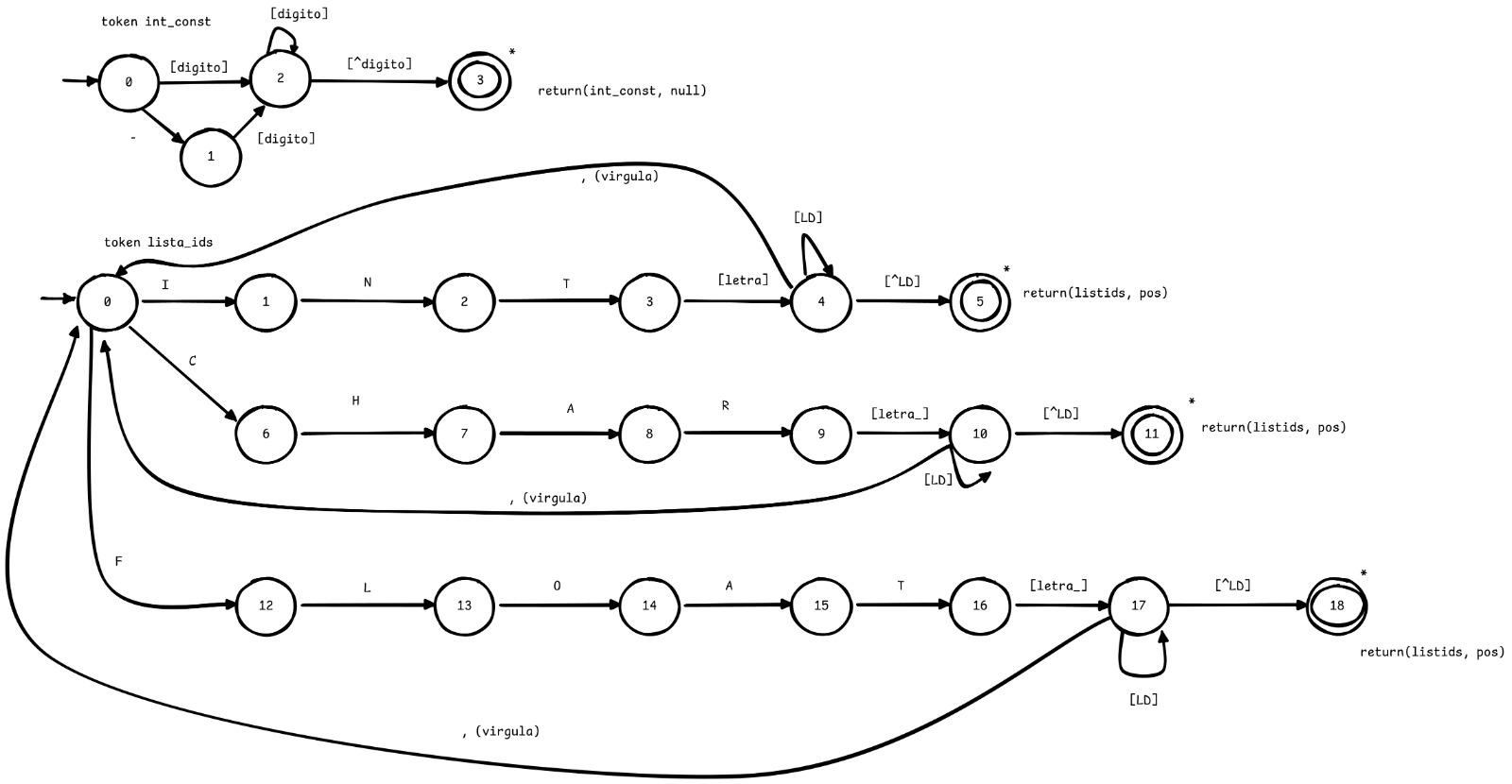
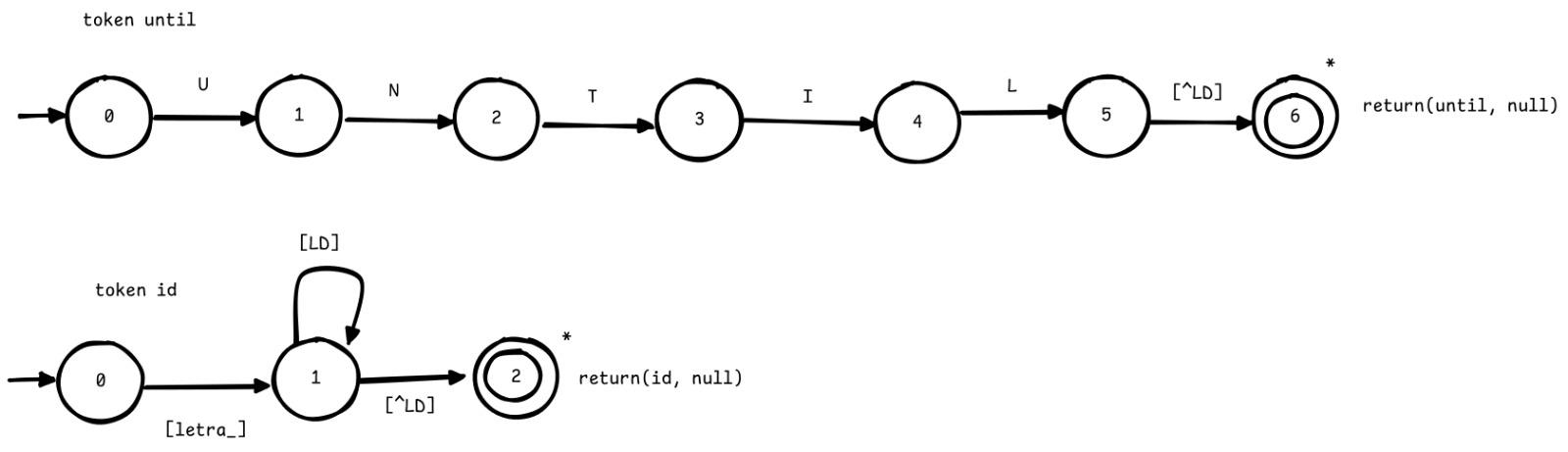
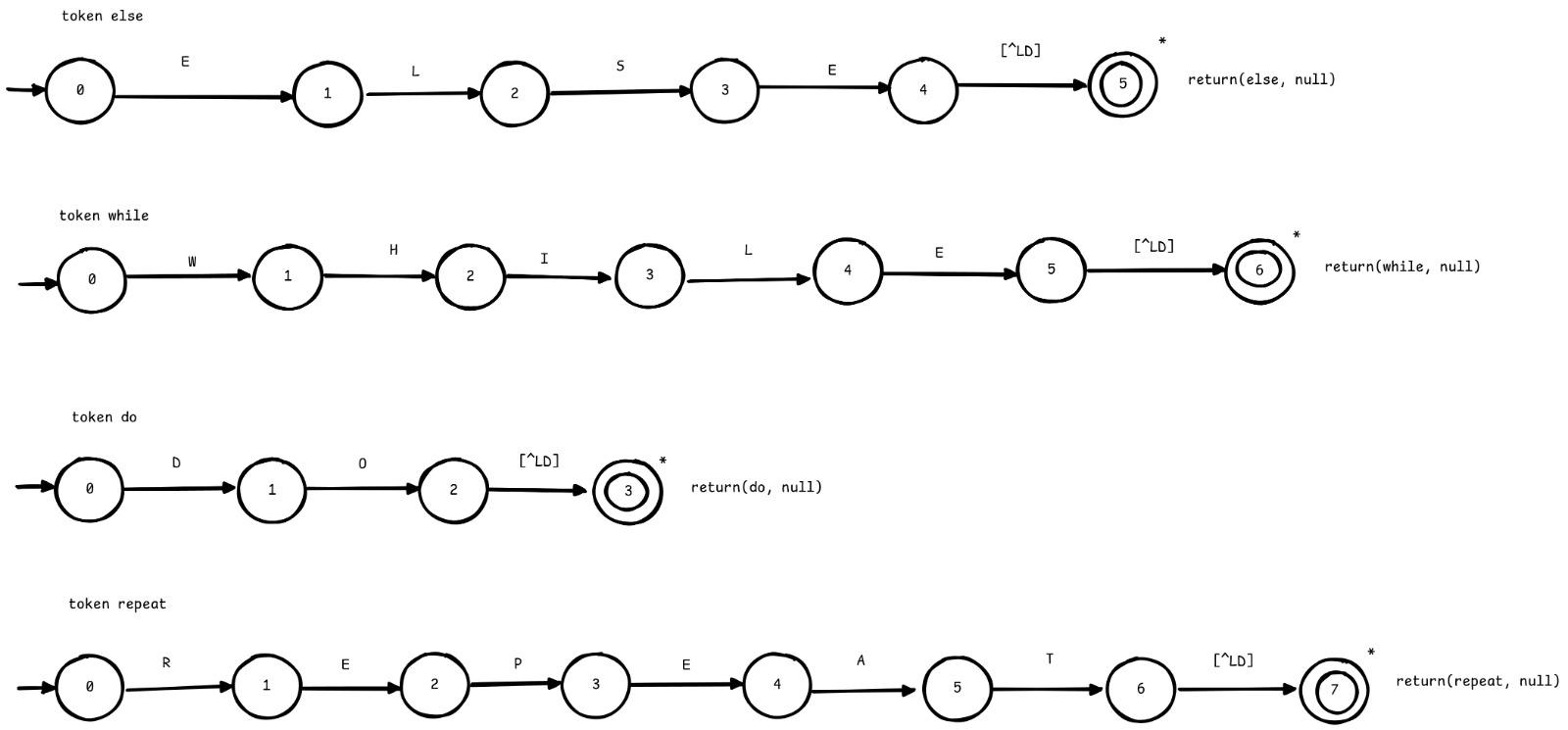
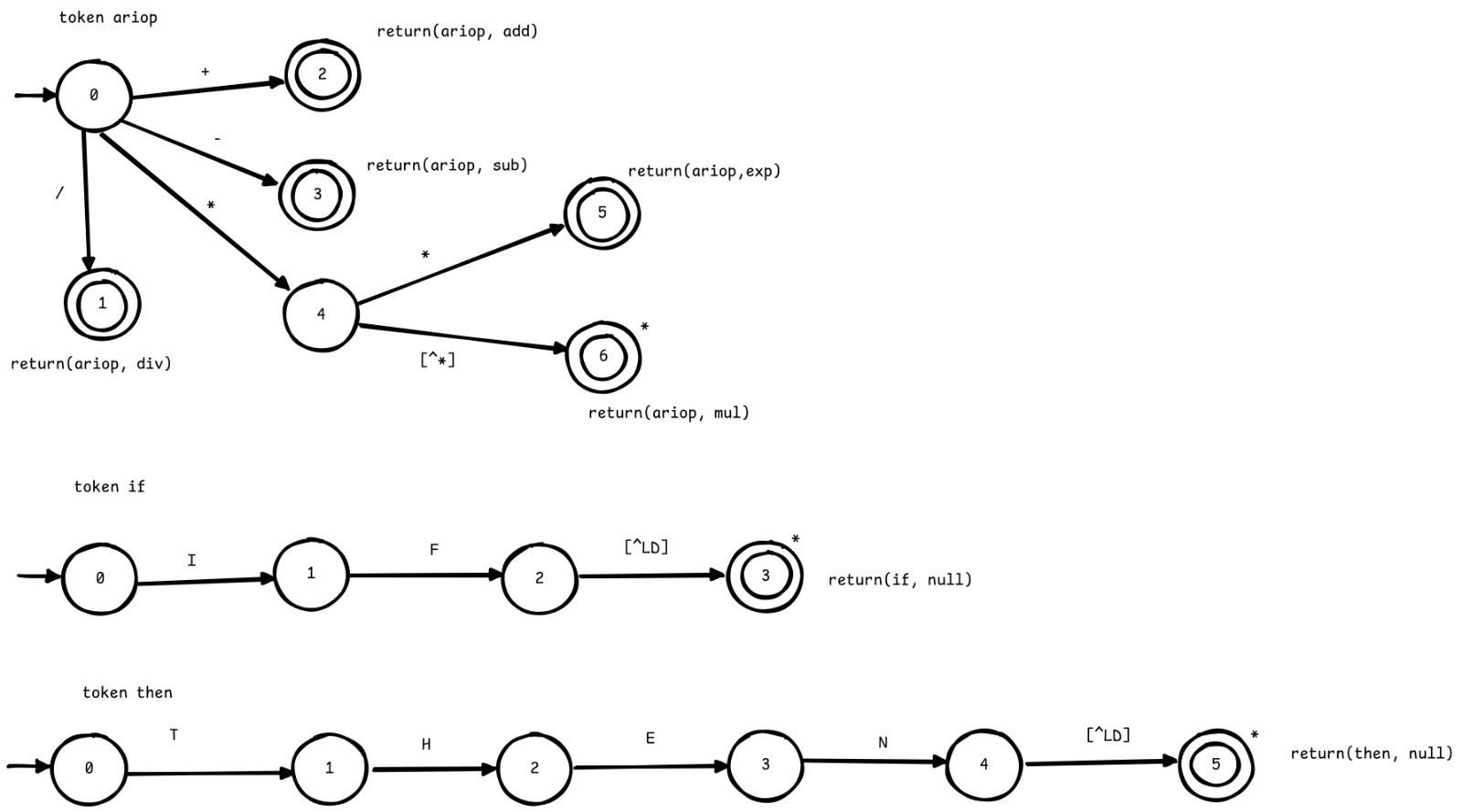
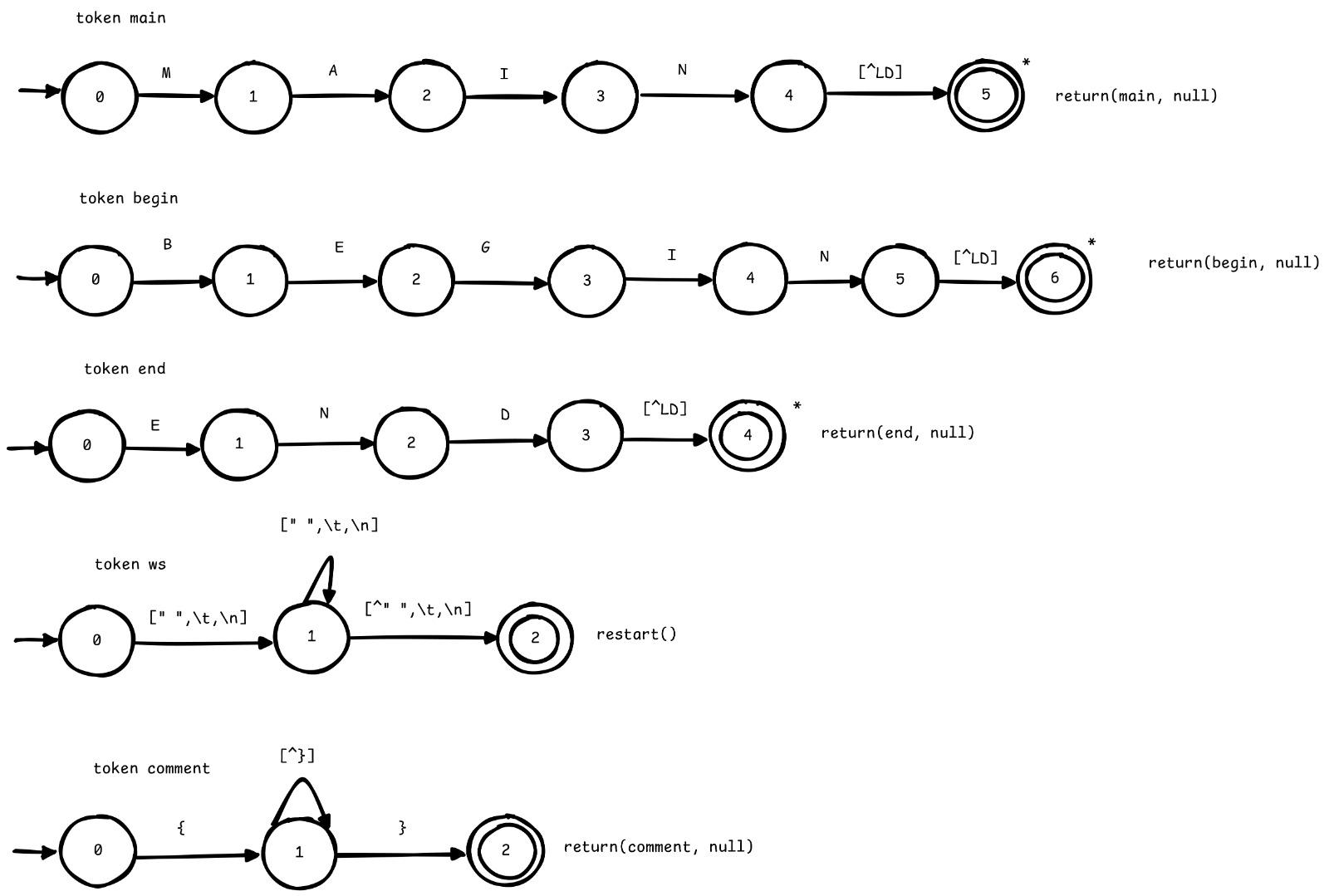
* **Gerar um diagrama de transição para cada token**

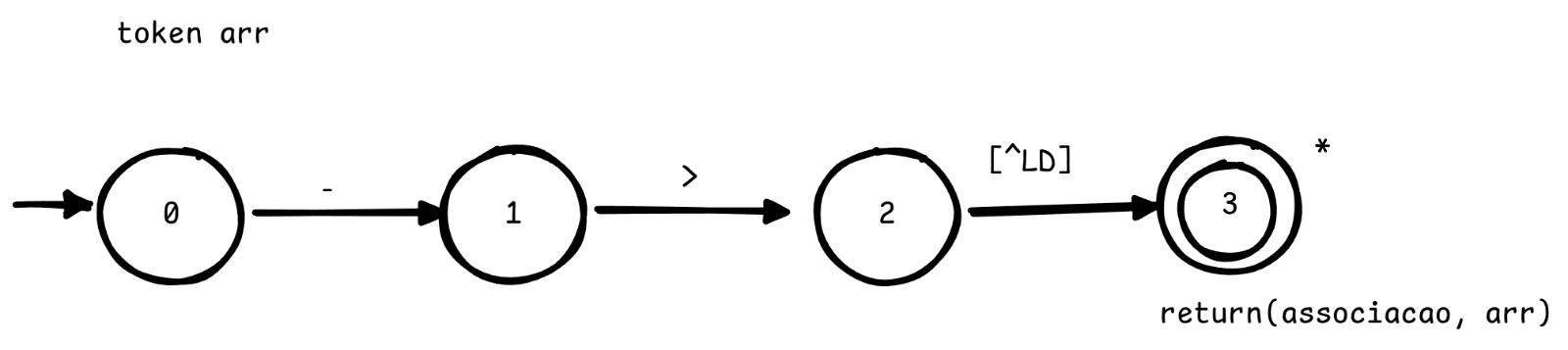
letra\_ → [ A – Z a – z \_ ]

digito → [ 0 – 9 ]

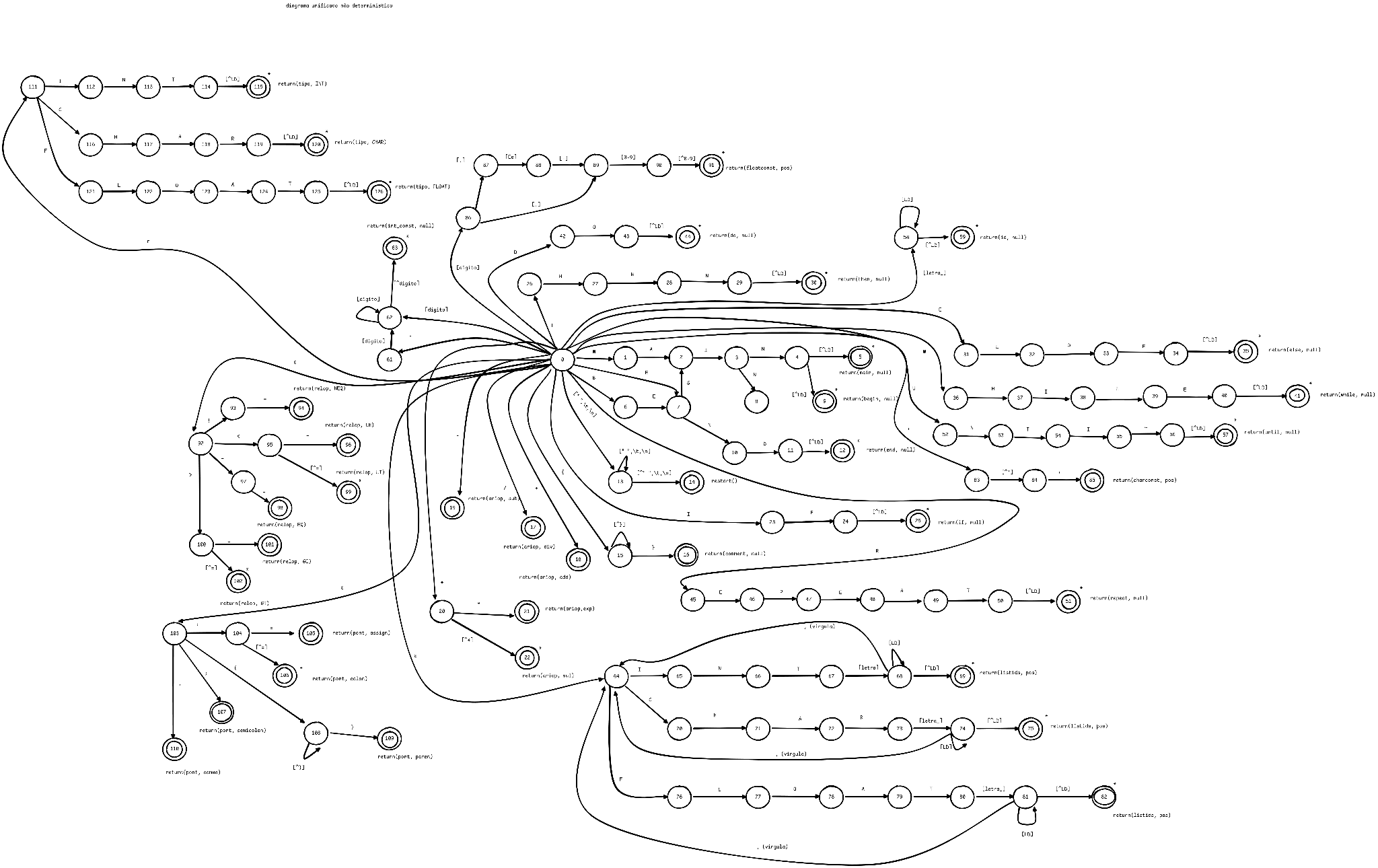
[^LD] = [^letra\_digito]

[D] = digito

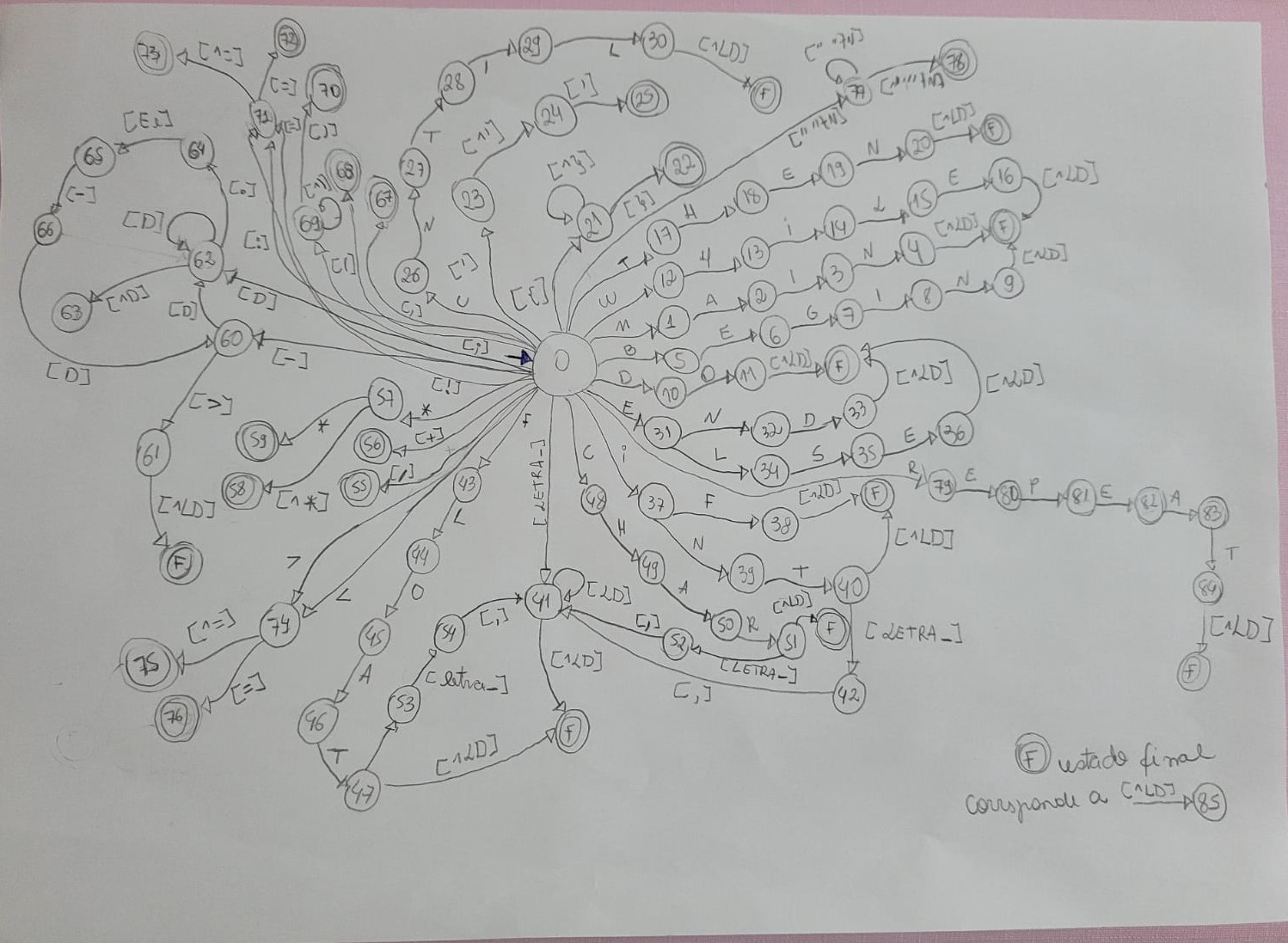


****

* **Unificá-los em um diagrama não determinístico**

****

* **Convertê-lo em um diagrama de transição determinístico**

****

A partir da entrada [-] é possível ligar os diagramas ariop, arr, id e int\_const

A partir dos tipos int, char e float do é possível ligar os diagramas float\_const, int\_const, id e listaIds.

A partir de ‘e’ end e else

A partir de ‘i’ if e int (de tipo)

A partir da entrada [=] é possível ligar os diagramas relop e pont