Projeto de Compiladores

Projeto da Disciplina de Construção de Compiladores, cujo objetivo é implementar um frontend de compilador de uma linguagem, criando o analisador léxico com Codificação Direta e a Análise Sintática Preditiva Dirigida por Tabela.

Trabalho desenvolvido pelos alunos:

- Gabriel José Bueno Otsuka
- Marcos Felipe Belisário Costa

Especificação da linguagem (1a etapa)

- Estrutura principal:
  function nome_programa() bloco
- bloco:
  {
  declaração de variáveis
  sequência de comandos
  }
- Declaração de variáveis:
  tipo: lista_ids;
  - tipos: int, char, float
  - lista de ids (1 ou +):
    int: idade; float: nota; chat: c, letra, s;
- Comando de seleção:
  se (cond) entao
  comando ou bloco
  senao
  comando ou bloco
  - a estrutura senao deverá ser opcional
- Comando de repetição:

    - enquanto (cond) faca
      comando ou bloco
    - repita
      comando ou bloco
      ate (cond)

- Condições:
  Apenas operadores relacionais. ==, <>, <, >, <=, >=.
  Os operadores podem ser expressões aritméticas
- Comando de atribuição
  id = expresssao;
- Expressões aritméticas
  Permite parênteses para alterar ordem de precedência
  Expressões com operadores +, -, *, /, ^
  Operandos podem ser
    - char: ‘a’
    - int: 0 ≥ i ≥ 32767
    - float: ponto fixo (5.3) ou notação científica (0.1E-2)

Definição da Gramática Livre de Contexto

As regras da linguagem devem ser definidas por uma gramática livre de contexto, para que a definição sintática seja precisa. Desse modo, a primeira versão da GLC gerada foi a seguinte:

```
G1 = (
  {
    ini, bloco, decls_var, decl_var, lista_id, exp, termo, fator,
    cmds, cmd, cmd_atrib, cmd_cond, cond, cmd_rep, coment
  },
  {
    function, id, (, ), {, }, tipo, :, ;, ,, +, -, *, /, =, se,
    senao, op_rel, enquanto, faca, repita, ate, texto_comentario,
    num,
  }
  P,
  ini
);
```

```
P =
     ini ::= function id() { <bloco> }
     bloco ::= <decls_var> <cmds> | ε
     decls_var ::= <decl_var> <decl_vars> | <decl_var>
     decl_var ::= tipo: <lista_id>;
     lista_id ::= id | id, <lista_id>
     exp3 ::= <exp3> op_prec3 <exp2> | <exp2>
     exp2 ::= <exp2> op_prec2 <exp1> | <exp1>
     exp1 ::= <exp1> op_prec1 <fator> | <fator>
     fator ::= id | num | (<exp3>)
     cmds ::= <cmd> <cmds> | <cmd>
     cmd ::= <cmd_atr> | <cmd_cond> | cmd_rep | {<bloco>}
     cmd_atrib ::= id = <exp3> ;
     cmd_cond ::= se ( <cond> ) <cmd> [senao <cmd>]
     cond ::= <cond> relop <fator> | <fator>
     cmd_rep ::= enquanto (<cond>) faca <cmd> |
                          repita <cmd> ate (<cond>)
     coment ::= /*texto_comentario*/
```

Nota-se que a gramática não é definitiva para a construção do frontend do compilador, visto que o método preditivo dirigido por tabela exige que a gramática seja LL1, ou seja, sem recursões à esquerda, nem ambiguidades. Posteriormente, na etapa 3, técnicas para tornar a linguagem LL1 serão aplicados.

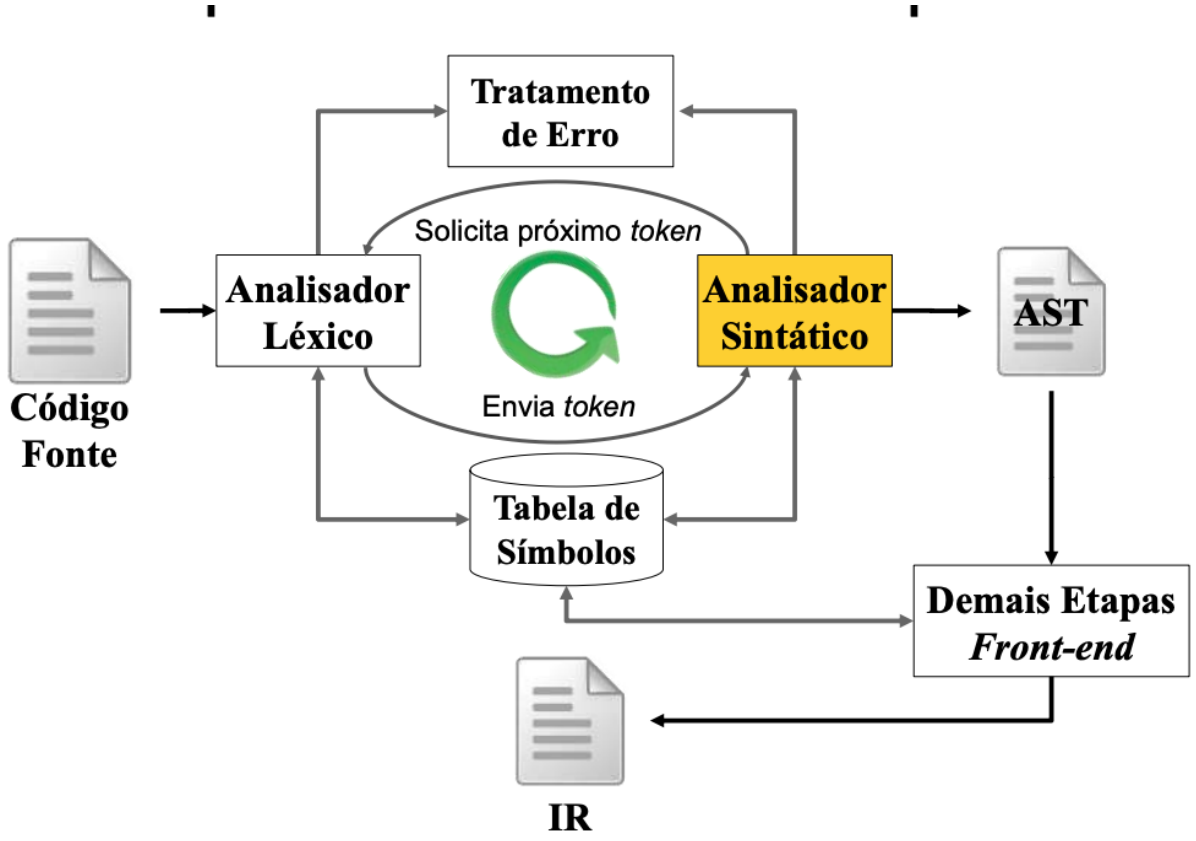
Identificação dos Tokens

Nome	Atributo	Expressão Regular
function	-	function
se	-	se
entao	-	entao
senao	-	senao
enquanto	-	enquanto
faca	-	faca
repita	-	repita
ate	-	ate
dois_pontos	-	:
ponto_virgula	-	;
virgula	-	,

Nome	Atributo	Expressão Regular
atribuicao	-	=
tipo	O tipo de variável reconhecido	char | int | float
abre_parenteses		(
fecha_parenteses		)
abre_chaves		{
fecha_chaves		}
op_prec1	-	^
op_prec2	Operador reconhecido (*, /)	[*/]
op_prec3	Operador reconhecido (+, -)	[+-]
relop	Operador reconhecido (>=, <=, >, <, ==, <>)	< | > | <= | >= | == | <>
texto_comentario	**ESSE TOKEN NÃO É RETORNADO**	**/***[^(*/)]****/**
ws	**ESSE TOKEN NÃO É RETORNADO**	(“ “ | \t | \n)
const_char	Valor reconhecido	**‘**[^’]**’**
const_int	Valor reconhecido	*digitos*
const_float	Valor reconhecido	*digitos*.*digitos*(E [+-]? *digitos*)?
id	Posição na tabela de símbolos	*letra* (*letra* | *digito* | _)*

Análise léxica (2a etapa)

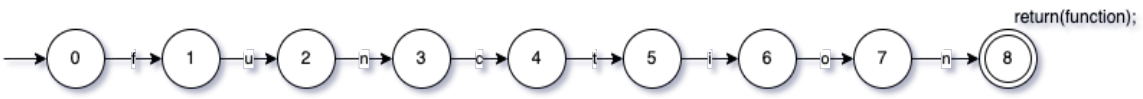
Para que a análise do código fonte seja feita, é necessário que haja um analisador léxico, responsável por quebrar o arquivo de entrada nos tokens cobertos pela gramática e retorná-los para o Analisador sintático.



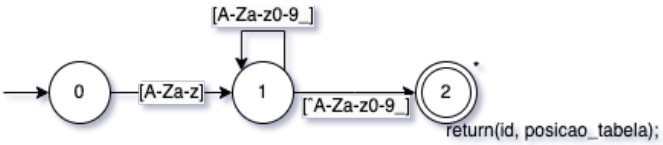
Isso será feito via codificação direta, algoritmo no qual, para cada caractere lido da cadeia de entrada, se transitará para um estado que representa sua leitura e, eventualmente, poderá retornar qual o lexema identificado, assim como a posição e possíveis atributos relevantes para o analisador sintático.

Diagrama de transição

- Para cada token:
  - function

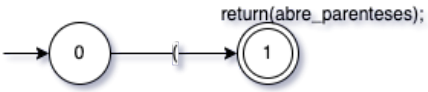


  - id

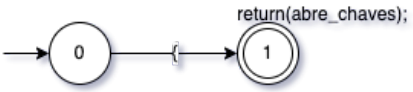


- fecha_parenteses

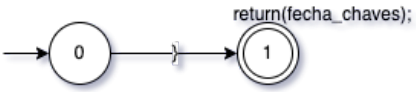
- abre_parenteses



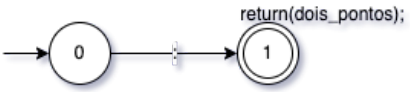
- abre_chaves



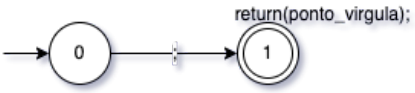
- fecha_chaves



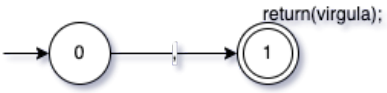
- dois_pontos



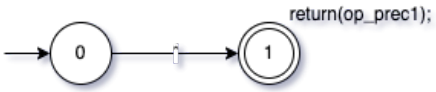
- ponto_virgula



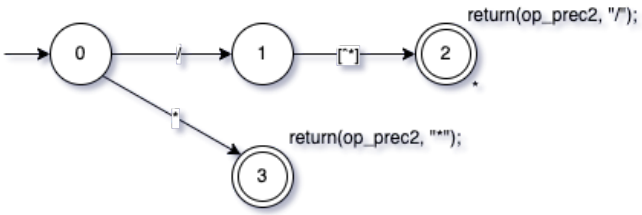
- virgula



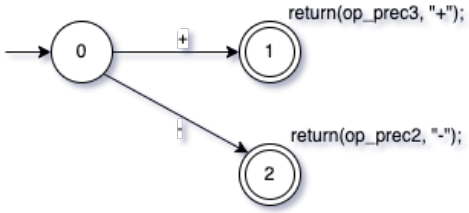
- op_prec1



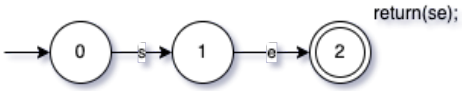
- op_prec2



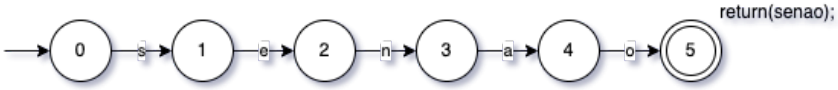
- op_prec3



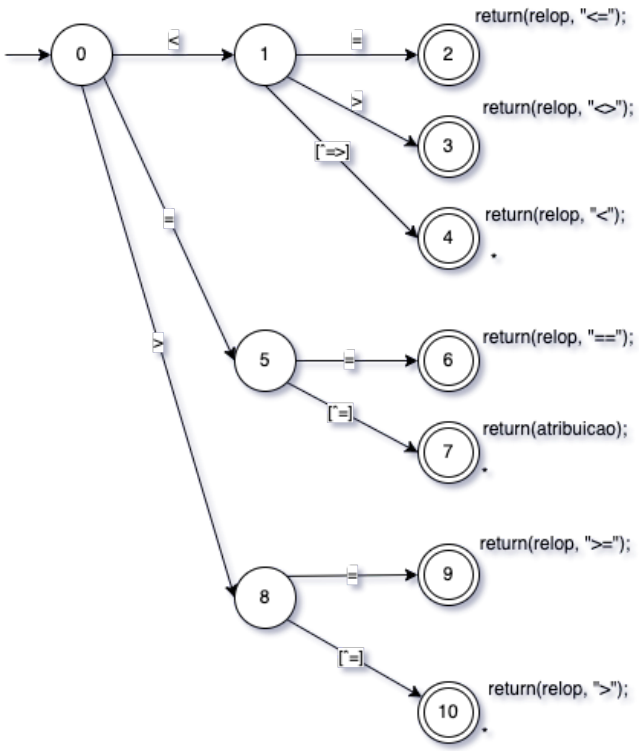
- se



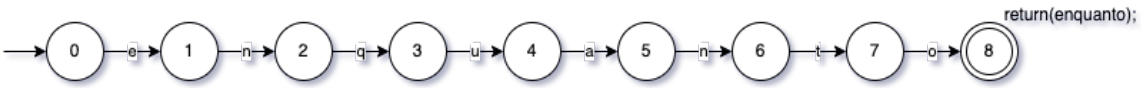
- senao



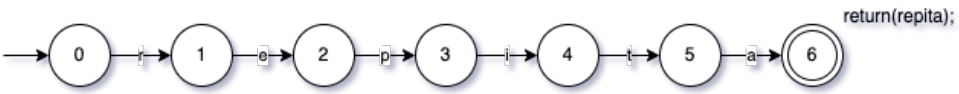
- relop



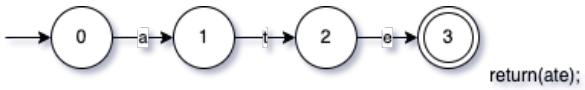
- enquanto



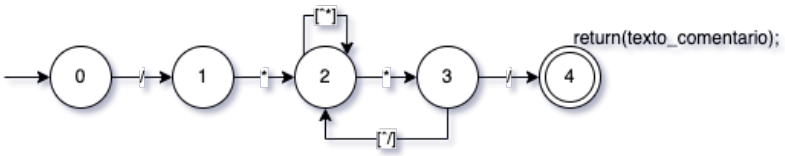
- faca
- repita



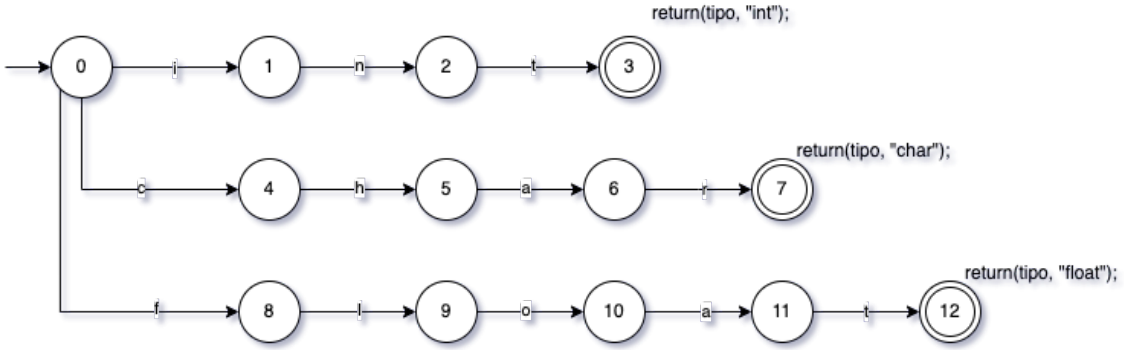
- ate



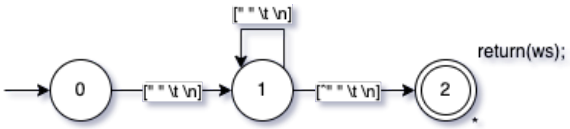
- texto_comentario



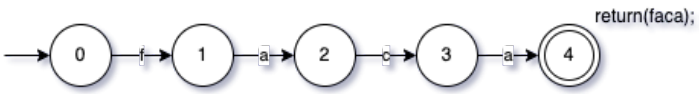
- tipo



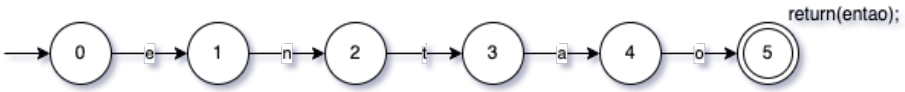
- ws



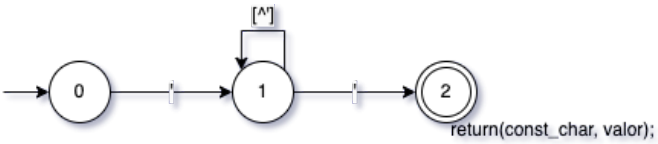
- faca



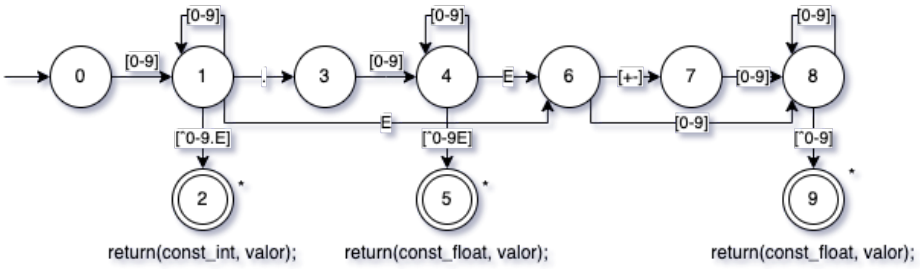
- entao



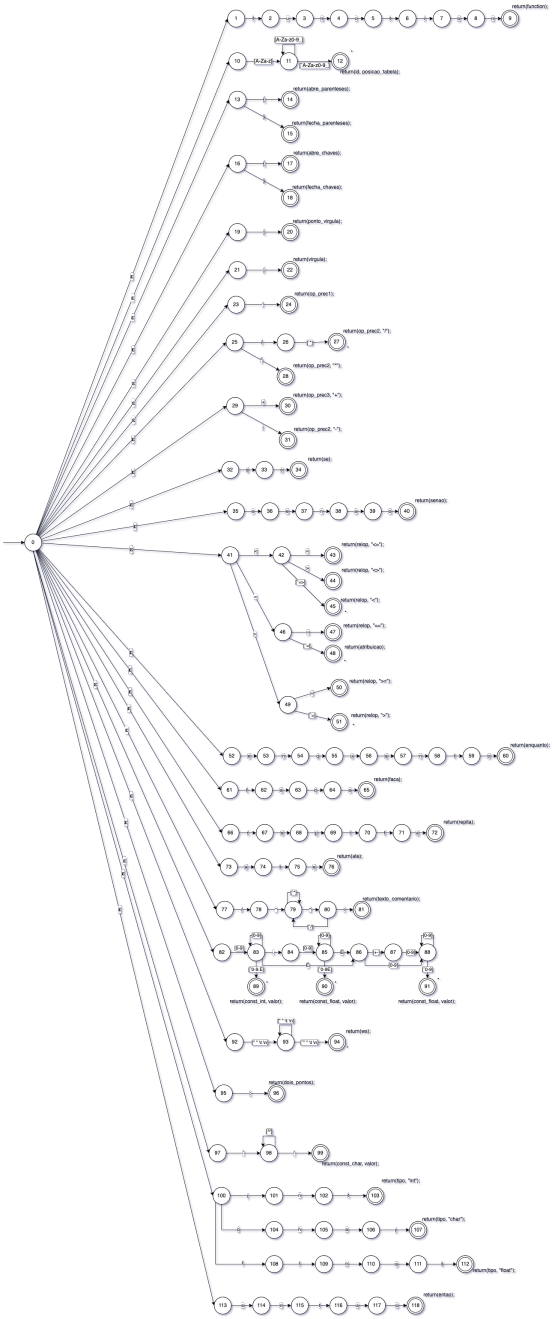
- const_char



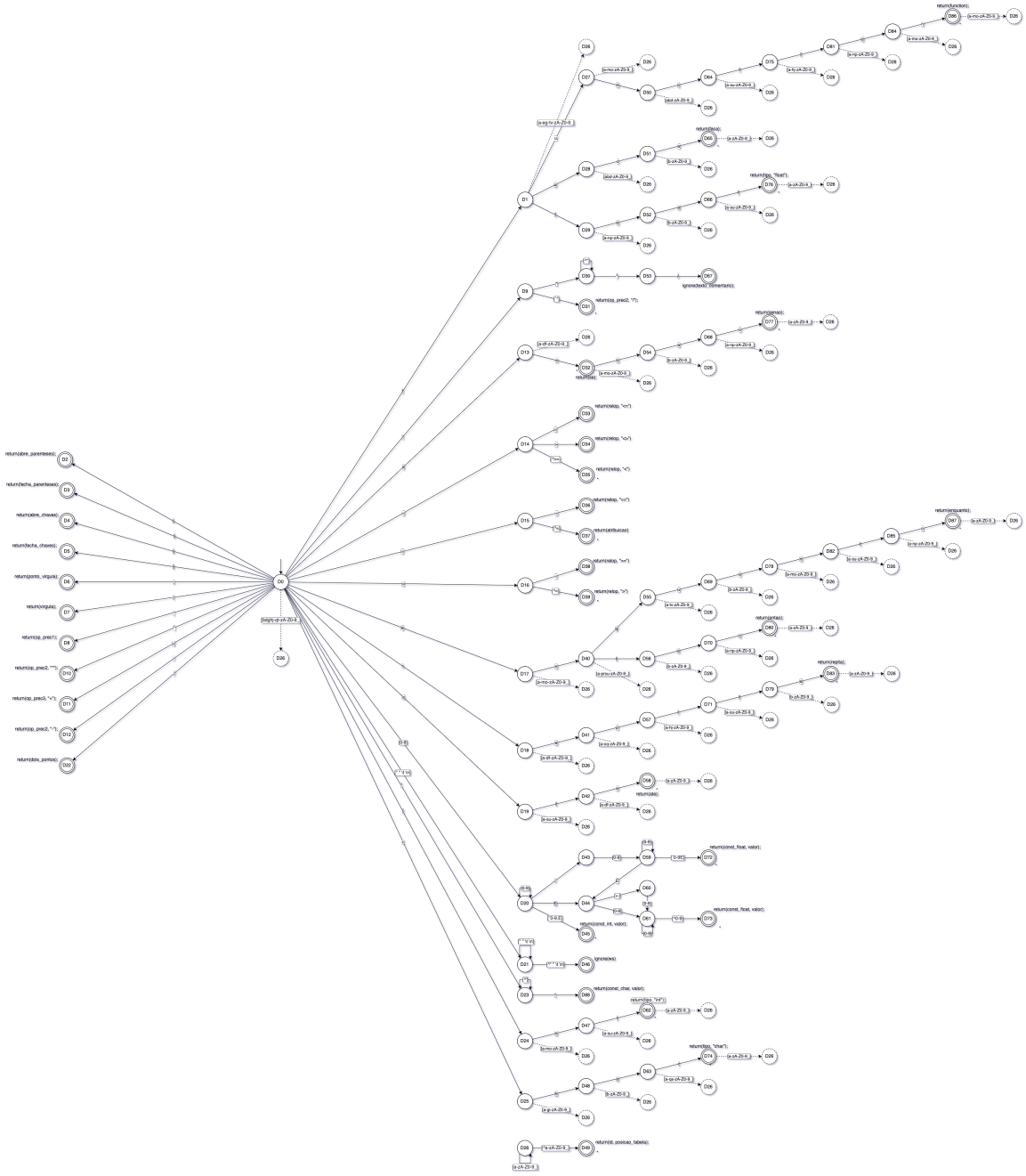
- const_float e const_int



- Unificação em um AFN-ε



- Para conversão do AFN-ε e implementação do analisador léxico, criou-se uma tabela para auxílio, antes da concepção do desenho. A tabela pode ser encontrada no link a seguir https://docs.google.com/spreadsheets/d/1N0R90HMKUptBUz6IxSfdqHFuMEfERSNWB6NGf6c1cbY/edit#gid=0
- O diagrama final, determinístico



Análise Sintática (3a etapa)

Correções da linguagem

Para que a análise sintática dirigida por tabela seja eficiente, é preciso ter uma gramática LL1, que por sua vez necessita que não haja ambiguidade. As ambiguidades antes presentes foram removidas por fatoração, postergando a decisão da produção a ser considerada até que se tenha certeza de qual produção adotar.

P =

ini ::= function id() <bloco>
bloco ::= { <bloco_aux> }
bloco_aux ::= <decl_vars> <cmds> | <cmds>
decl_vars ::= <decl_var> <decl_vars> | <decl_var>
decl_var ::= tipo: <lista_id>;
lista_id ::= id | id, <lista_id>
cmds ::= <cmd> <cmds> | <cmd>
cmd ::= <cmd_atr> | <cmd_cond> | <cmd_rep>
cmd_atrib ::= id = <arit3> ;
arit3 ::= <arit3> op_arit_prec3 <arit2> | <arit2>
arit2 ::= <arit2> op_arit_prec2 <arit1> | <arit1>
arit1 ::= <arit1> op_arit_prec1 <arit_fator> | <arit_fator>
arit_fator ::= id | num | const_char | (<arit3>)
cmd_cond ::= se (<cond>) <cmd_bloco> <senao>
cond ::= <arit3> relop <arit3> | (cond)
cmd_bloco::= <cmd> | <bloco>
senao ::= senao <cmd_bloco> | ε
cmd_rep ::= enquanto (<cond>) faca <cmd_bloco> |
repita <cmd_bloco> ate (<cond>)

Remoção da Recursão à Esquerda e Fatoração

Além da fatoração, o método exige que não haja recursividade à esquerda na gramática, nem direta, nem indireta. Desse modo, a remoção de recursividade foi aplicada nas produções que representam expressões aritméticas.

P =

ini ::= function id() <bloco>
bloco ::= { <Vbloco_aux> }
bloco_aux ::= <decl_vars> <cmds> | <cmds>
decl_vars ::= <decl_var><decl_vars_fat>
decl_vars_fat ::= <decl_vars> | ε
decl_var ::= tipo: <lista_id>;
lista_id ::= id <lista_id_fat>
lista_id_fat ::= , <lista_id> | ε
cmds ::= <cmd><cmds_fat>
cmds_fat ::= <cmds> | ε
cmd ::= <cmd_atrib> | <cmd_cond> | <cmd_rep>
cmd_atrib ::= id = <arit3> ;
arit3 ::= <arit2><arit3'>
arit3' ::= op_arit_prec3 <arit2> <arit3'> | ε
arit2 ::= <arit1><arit2'>
arit2' ::= op_arit_prec2 <arit1> <arit2'> | ε
arit1 ::= <arit_fator><arit1'>
arit1' ::= op_arit_prec1 <arit_fator> <arit1'> | ε
arit_fator ::= id | const_int | const_float | const_char | (<arit3>)
cmd_cond ::= se (<cond>) entao <cmd_bloco> <senao_fat>
cond ::= <arit3> relop <arit3>
cmd_bloco::= <cmd> | <bloco>
senao_fat ::= senao <cmd_bloco> | ε
cmd_rep ::= enquanto (<cond>) faca <cmd_bloco> |
repita <cmd_bloco> ate (<cond>)

Cálculo dos First() e Follows()

Valores auxiliares para construção da tabela preditiva.

Símbolo	First()	Follow()
ini	{ function }	{ $ }
bloco	{ { }	{ { , id, se, enquanto, repita, ate }
bloco_aux	{ tipo, id, se, enquanto, repita }	{ } }

Símbolo	First()	Follow()
decl_vars	{ tipo }	{ id, se, enquanto, repita }
decl_vars_fat	{ tipo, ε }	{ id, se, enquanto, repita }
decl_var	{ tipo }	{ tipo, id, se, enquanto, repita }
lista_id	{ id }	{ ; }
lista_id_fat	{, , ε}	{ ; }
cmds	{ id, se, enquanto, repita }	{ } }
cmds_fat	{ id, se, enquanto, repita, **ε**}	{ } }
cmd	{ id, se, enquanto, repita }	{ { , id, se, enquanto, repita, ate }
cmd_atrib	{ id }	{ { , id, se, enquanto, repita, ate }
arit3	{ id, const_int, const_char, const_float, (}	{ ; ,) , relop }
arit3'	{ op_arit_prec3, ε}	{ ; ,) , relop}
arit2	{ id, const_int, const_char, const_float, (}	{ op_arit_prec3, ; ,) , relop}
arit2'	{ op_arit_prec2, ε}	{ op_arit_prec3, ; ,) , relop}
arit1	{ id, const_int, const_char, const_float, (}	{ op_arit_prec2, op_arit_prec3, ; ,) , relop }
arit1'	{ op_arit_prec1, ε}	{ op_arit_prec2, op_arit_prec3, ; ,) , relop }
arit_fator	{ id, const_int, const_char, const_float, (}	{ op_arit_prec1, op_arit_prec2, op_arit_prec3, ; ,) , relop}
cmd_cond	{ se }	{ { , id, se, enquanto, repita, ate }
cond	{ id, const_int, const_char, const_float, (}	{) }
cmd_bloco	{ { , id, se, enquanto, repita }	{ { , id, se, enquanto, repita, ate }
senao_fat	{ senao, **ε**}	{ { , id, se, enquanto, repita, ate }
cmd_rep	{ enquanto, repita}	{ { , id, se, enquanto, repita, ate }

Tabela de produções com os identificadores utilizados em código

id	produção	com ids
0	**ini** ::= function id() <bloco>	0 3 11 19 24

id	produção	com ids
1	**bloco** ::= { <bloco_aux> }	1 25 17
2	**bloco_aux** ::= <decl_vars> <cmds>	26 31
3	**bloco_aux** ::= <cmds>	31
4	**decl_vars** ::= <decl_var><decl_vars_fat>	28 27
5	**decl_vars_fat** ::= <decl_vars>	26
6	**decl_vars_fat** ::= ε	X
7	**decl_var** ::= tipo: <lista_id>;	2 23 30 18
8	**lista_id** ::= id <lista_id_fat>	3 31
9	**lista_id_fat** ::= , <lista_id>	7 30
10	**lista_id_fat** ::= ε	X
11	**cmds** ::= <cmd><cmds_fat>	34 33
12	**cmds_fat** ::= <cmds>	32
13	**cmds_fat** ::= ε	X
14	**cmd** ::= <cmd_atrib>	35
15	**cmd** ::= <cmd_cond>	43
16	**cmd** ::= <cmd_rep>	47
17	**cmd_atrib** ::= id = <arit3> ;	3 24 37 18
18	**arit3** ::= <arit2><arit3'>	39 38
19	**arit3'** ::= op_arit_prec3 <arit2> <arit3'>	12 39 38
20	**arit3'** ::= ε	X
21	**arit2** ::= <arit1><arit2'>	41 40
22	**arit2'** ::= op_arit_prec2 <arit1> <arit2'>	13 41 40
23	**arit2'** ::= ε	X
24	**arit1** ::= <arit_fator><arit1'>	43 42
25	**arit1'** ::= op_arit_prec1 <arit_fator> <arit1'>	14 43 42
26	**arit1'** ::= ε	X
27	**arit_fator** ::= id	3
28	**arit_fator** ::= const_int	8

id	produção	com ids
29	**arit_fator** ::= const_char	9
30	**arit_fator** ::= const_float	10
31	**arit_fator** ::= (<arit3>)	11 37 19
32	**cmd_cond ::**= se (<cond>) entao <cmd_bloco> <senao_fat>	4 11 45 19 22 44 47
33	**cond** ::= <arit3> relop <arit3>	37 20 37
34	**cmd_bloco**::= <cmd>	35
35	**cmd_bloco**::= <bloco>	26
36	**senao_fat** ::= senao <cmd_bloco>	15 46
37	**senao_fat** ::= ε	X
38	**cmd_rep** ::= enquanto (<cond>) faca <cmd_bloco>	5 11 45 19 21 46
39	**cmd_rep** ::= repita <cmd_bloco> ate (<cond>)	6 46 16 11 45 19

	function	{	tipo	id	se	enquanto	repita	,	const_int	const_char	const_float	(	op_arit_prec3	op_arit_prec2	op_arit_prec1	senao	ate	}	;	)	relop	faca	entao	:	=
25 ini OK	0																								
26 bloco OK		1																							
27 bloco_aux OK			2	3	3	3	3																		
28 decl_vars			4																						
29 decl_vars_fat			5	6	6	6	6																		
30 decl_var			7																						
31 lista_id				8																					
32 lista_id_fat								9											10						
33 cmds				11	11	11	11																		
34 cmds_fat				12	12	12	12											13							
35 cmd				14	15	16	16																		
36 cmd_atrib				17																					
37 arit3				18					18	18	18	18													
38 arit3'													19						20	20	20				
39 arit2				21					21	21	21	21													
40 arit2'													23	22					23	23	23				
41 arit1				24					24	24	24	24													
42 arit1'													26	26	25				26	26	26				
43 arit_fator				27					28	29	30	31													
44 cmd_cond					32																				
45 cond				33					33	33	33	33													
46 cmd_bloco		35		34	34	34	34																		
47 senao_fat		37		37	37	37	37									36	37								?
48 cmd_rep						38	39																		