Prova de que a linguagem modificada é LL (1)

Considera que para uma linguagem ser LL (1), deve satisfazer às seguintes condições:

- Não deve ser recursiva à esquerda;
- A regra de produção que deve ser escolhida ao desenvolver um não-terminal deve ser determinada por esse não-terminal e pelo (no máximo) próximo token na entrada.

Para isto a gramática precisa:

1. Não ser recursiva à esquerda

2. Estar fatorada

3. Para todo A ∈ N tal que A ∗⇒ ε, First(A) ∩ Follow(A) = ∅

Assim, consideremos a gramática definida no arquivo **grammar.txt**, que foi modificada para não ser recursiva à esquerda e estar fatorada. Demonstraremos que, de fato, essas modificações acarretaram em uma gramática que satisfaça as condições 1 e 2 definidas acima. Para isso, podemos montar a tabela de predição da gramática, como abaixo:

#	Expressão	Predição
1	PROGRAM → STATEMENT	{, break, ;, int, float, string, print, return, for, ident, if, read
2	PROGRAM → FUNCLIST	def
3	PROGRAM → EPISILON	EPISILON
4	FUNCLIST → FUNCDEF FUNCLISTAUX	def
5	FUNCLISTAUX → FUNCLIST	def
6	FUNCLISTAUX → EPISILON	EPISILON
7	FUNCDEF → def ident (PARAMLIST) { STATELIST }	def
8	PARAMLIST → DATATYPE ident PARAMLISTAUX	int, float, string
9	PARAMLIST → EPISILON	EPISILON
10	PARAMLISTAUX → , PARAMLIST	,
11	PARAMLISTAUX → EPISILON	EPISILON
12	DATATYPE → int	int
13	DATATYPE → float	float
14	DATATYPE → string	string
15	STATEMENT → VARDECL ;	int, float, string

16	STATEMENT → ATRIBSTAT ;	ident
17	STATEMENT → PRINTSTAT ;	print
18	STATEMENT → READSTAT ;	read
19	STATEMENT → RETURNSTAT ;	return
20	STATEMENT → IFSTAT	if
21	STATEMENT → FORSTAT	for
22	STATEMENT → { STATELIST }	{
23	STATEMENT → break ;	break
24	STATEMENT → ;	;
25	VARDECL → DATATYPE ident OPT_VECTOR	int, float, string
26	OPT_VECTOR → [int_constant] OPT_VECTOR	
27	OPT_VECTOR → EPISILON	EPISILON
28	ATRIBSTAT → LVALUE = ATRIBSTAT_RIGHT	ident
29	ATRIBSTAT_RIGHT → FUNCCALL_OR_EXPRESSION	=+
30	ATRIBSTAT_RIGHT → ALLOCEXPRESSION	new
31	FUNCCALL_OR_EXPRESSION → + FACTOR REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR	=+
32	FOLLOW_IDENT → OPT_ALLOC_NUMEXP REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR	, EPISILON
33	FOLLOW_IDENT → (PARAMLISTCALL)	(
34	PARAMLISTCALL → ident PARAMLISTCALLAUX	ident
35	PARAMLISTCALL → EPISILON	EPISILON
36	PARAMLISTCALLAUX → , PARAMLISTCALL	,
37	PARAMLISTCALLAUX → EPISILON	EPISILON
38	PRINTSTAT → print EXPRESSION	print
39	READSTAT → read LVALUE	read
40	RETURNSTAT → return	return
41	IFSTAT → if (EXPRESSION) { STATELIST } OPT_ELSE	if
42	OPT_ELSE → else { STATELIST }	else
43	OPT_ELSE → EPISILON	EPISILON
44	FORSTAT → for (ATRIBSTAT ; EXPRESSION ; ATRIBSTAT) STATEMENT	for

45	STATELIST → STATEMENT OPT_STATELIST;	{, break, ;, int, float, string, print, return, for, ident, if, read
46	OPT_STATELIST → STATELIST	{, break, ;, int, float, string, print, return, for, ident, if, read
47	OPT_STATELIST → EPISILON	EPISILON
48	ALLOCEXPRESSION → new DATATYPE [NUMEXPRESSION] OPT_ALLOC_NUMEXP	new
49	OPT_ALLOC_NUMEXP → [NUMEXPRESSION] OPT_ALLOC_NUMEXP	
50	OPT_ALLOC_NUMEXP → EPISILON	EPISILON
51	EXPRESSION → NUMEXPRESSION OPT_REL_OP_NUM_EXPR	=+, -, int_constant
52	OPT_REL_OP_NUM_EXPR → REL_OP NUMEXPRESSION	<, >, <=, >=, ==, /=
53	OPT_REL_OP_NUM_EXPR → EPISILON	EPISILON
54	REL_OP → <	<
55	REL_OP → >	>
56	REL_OP → <=	<=
57	REL_OP → >=	>=
58	REL_OP → ==	==
59	REL_OP → /=	/=
60	NUMEXPRESSION → TERM REC_PLUS_MINUS_TERM	=+, -, int_constant
61	REC_PLUS_MINUS_TERM → PLUS_OR_MINUS TERM REC_PLUS_MINUS_TERM	=+, -
62	REC_PLUS_MINUS_TERM → EPISILON	EPISILON
63	PLUS_OR_MINUS → +	=+
64	PLUS_OR_MINUS → -	-
65	TERM → UNARYEXPR REC_UNARYEXPR	=+, -, int_constant
66	REC_UNARYEXPR → UNARYEXPR_OP TERM	*, /, %
67	REC_UNARYEXPR → EPISILON	EPISILON
68	UNARYEXPR_OP → *	*
69	UNARYEXPR_OP → /	/
70	UNARYEXPR_OP → %	%
71	UNARYEXPR → PLUS_OR_MINUS FACTOR	=+, -
72	UNARYEXPR → FACTOR	int_constant
73	FACTOR → int_constant	int_constant
74	LVALUE → ident OPT_ALLOC_NUMEXP	ident

Perceba que nessa tabela, as produções de cada não terminal satisfazem as seguintes condições:

1. **Para as produções associadas a cada não-terminal x, x -> \alpha_1 pr(r_1) | ... \alpha_n pr(r_n), First(\alpha_i) interseção First(\alpha_j) é vazio sempre que i =\= j. Ou seja, os corpos das produções têm de ter primeiros conjuntos disjuntos.**

2. **Para cada não-terminal x e o conjunto de produção associado ao não-terminal x -> \alpha_1 pr(r_1)0 |...|\alpha_n pr(r_n) no máximo um \alpha_i é anulável.**

Portanto, a gramática está devidamente fatorada e satisfaz as primeiras condições para estar em LL (1). Para finalizar a prova, basta demonstrar que:

Para todo A ∈ N tal que A $*\Rightarrow$ ε, First(A) ∩ Follow(A) = ∅

Ou seja, para todos os não terminais anuláveis, a intersecção entre os Follows e Firsts desses não terminais deve ser o conjunto vazio. Observando a tabela acima e a gramática alvo, podemos listar os não terminais anuláveis como sendo:

- PROGRAM;
- FUNCLISTAUX;
- PARAMLIST;
- PARAMLISTAUX;
- OPT_VECTOR;
- FOLLOW_IDENT;
- PARAMLISTCALL;
- OPT_ELSE;
- OPT_STATELIST;
- OPT_REL_OP_NUM_EXPR;
- REC_PLUS_MINUS_TERM;
- REC_UNARYEXPR

Montando os Firsts e Follows desses não terminais, temos os resultados presentes nos arquivos **firsts.txt e follows.txt**, é possível perceber a partir desses arquivos que a intersecção entre estes é vazia, portanto mostrando a última condição e chegando à conclusão de que a gramática alvo é LL (1).