PLC READ ME

The txt fille has comment about on what is wrong with the file. Also to change file, it is in main.

1 Define the rules for recognizing all lexemes as their proper token, and clearly define integer token codes for each token required for this language • Should have Regular Grammar, Regular Expression, or Finite Automat defined

Identifies [A-Za-z]{6,8}

Start \^

End \$

Int_literal [0-9]* (v|r|e)

Num #

Add_op +

Sub_op –

Mult_op *

Div_op /

Left_paren \(

Right_paren \)

Mod_op %

Left_brack \{

Right_brack]

Less_than >

Less_than_equal >=

greater_than <

Greater_than_equal <=

If_code con

Loop loop

Noequal !

EqualTo ==

Semicolon ;

v is 1 byte num is 2 byte r is 4 byte e is 8 byte

2 Define production rules for implementing the mathematical syntax of operators and operands, loops, variable declaration, selection statements • Enforce a non PEMDAS (BODMAS) order of operation, must have at least 6 levels of precedence • Keywords cannot use the words while, for, do, if, int, short, long i. Keywords should be unique, if others share your same words, you may lose more points than this problem is worth • You must clearly state the structure of your language with production rules

<begin> -> Start <statement> {<statement>} End

<statement>-> <ifsmt> | <loop> | <expr>

<loop> -> loop’(‘<boolexpr>’)’ '{

' <statement> '}' {<statement>}

<ifsmt>-> con'('<boolexpr>')''{'<statement>'}' {<statement>}

<boolexpr> -> identifer ‘(‘<|>|<=|>=|==|!’)’ int_literal

<expr> -> <term> {(+ | *) <term>}

<term> -> <comp> {(- | /) <comp>}

<comp>-> <eqaulity> {(<|>|<=|>=|==) <eqaulity>}

<eqaulity> -> <mod>{(=|!=)<mod>

<mod>-> <factor> {(%)<factor> |;}

<factor> -> identifes | int_literal | num ident idenfies semicolon | (<expr>) | semicolon

3 Show whether every rule set in your language conforms to the standard of an LL Grammar

First, we would need to if it passes pairwise disjoint

<begin> -> Start <statement> End | <statement>

First(<begin> = Start {<ifsmt> | <loop> | <expr>}

Passes since not have same values

<statement>-> <ifsmt> | <loop> | <expr>

First (statement)= con|loop| identifes | int_literal | num ident | (<expr>) | semicolon

Passes

<Loop> -> loop’(‘<boolexpr>’)’ ‘{‘<statement>’}’ <statement>

First(Loop)=loop

Passes

<ifsmt>-> con'('<boolexpr>')''{'<statement>'}' {<statement>}

First(ifsmt)=con

<boolexpr>-> identifer ‘(‘<|>|<=|>=|==|!’)’ int_literal

First(boolexpr)=identifier

<expr> -> <term> {(+ | *) <term>}

First(expr) -> <comp>

<term> -> <comp> {(- | /) <comp>}

First(term) -><eqaulity>

<comp>-> <eqaulity> {(<|>|<=|>=|==) <eqaulity>}

Firsrt(comp)=<mod>

<eqaulity> -> <mod>{(=|!=)<mod>

First(equality)=<factor>

<mod>-> <factor> {(%|;)<factor>}

First<mod>=<factor> -> identifes | int_literal | num ident | (| semicolon

<factor> -> identifes | int_literal | num ident idenfies semicolon | (<expr>) | semicolon

First(factor)= <factor> -> identifes | int_literal | num ident | (| semicolon

Since all pass pairwise disjoint and there is no left recursive since nonterminal don't repeat the same nonterminal instead going to a different version. So it passes ll grammer

Create a LR (1) parse table for your language. And show the trace of 4 code samples. Each must have 6 or more tokens.

begin-> start state end

begin-> state

state->expr

expr-> term + term

expr-> term * term

expr-> term

term->' '

term-> comp

term-> comp - comp

term-> comp / comp

comp-> equal < equal

comp-> equal > equal

comp-> equal == equal

comp-> equal <= equal

comp-> equal >= equal

comp-> ' '

comp-> equal

equal-> mod = mod

equal-> mod != mod

equal-> ' '

equal-> mod

mod-> factor % factor

mod->;

mod-> factor

mod->' '

factor-> id

factor-> num

factor->(expr)

factor-> numid

factor-> ;

factor-> factor + state

factor->end

state-> loop

loop-> Loop(boolexp){

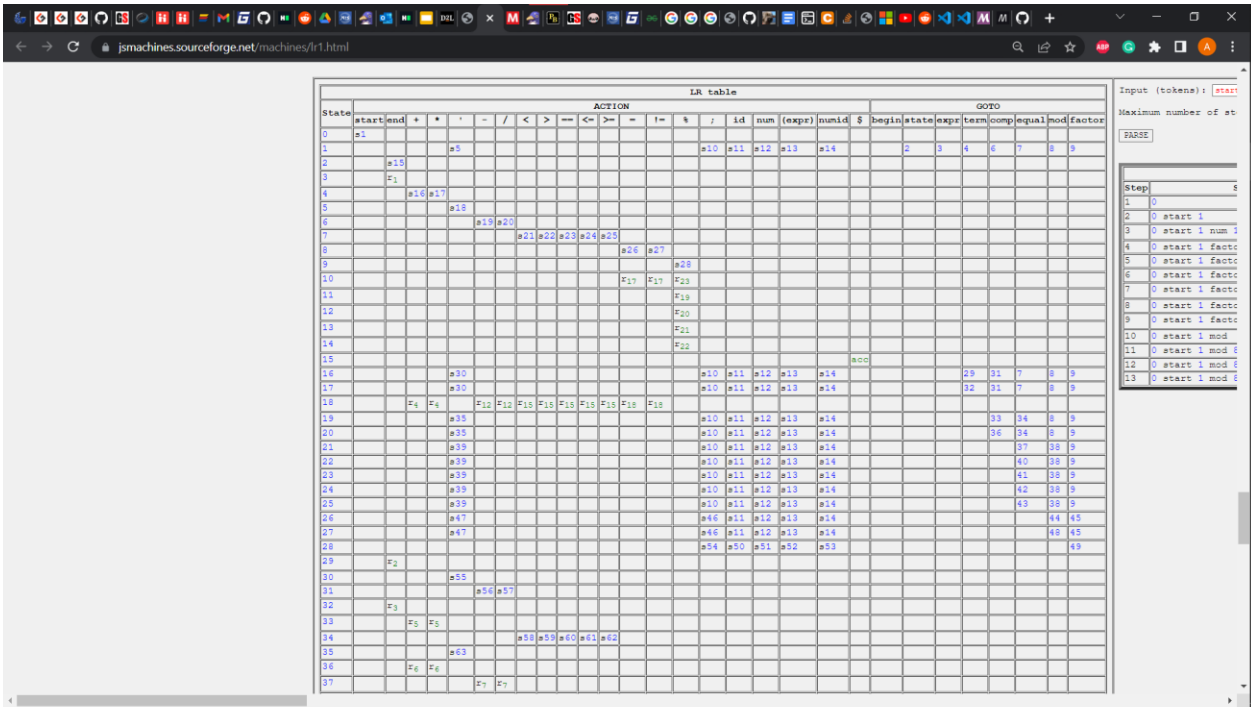
state} state

boolexp-> comp

state-> ifsmt

ifsmt-> con(boolexp){ state } state

The table



LR table

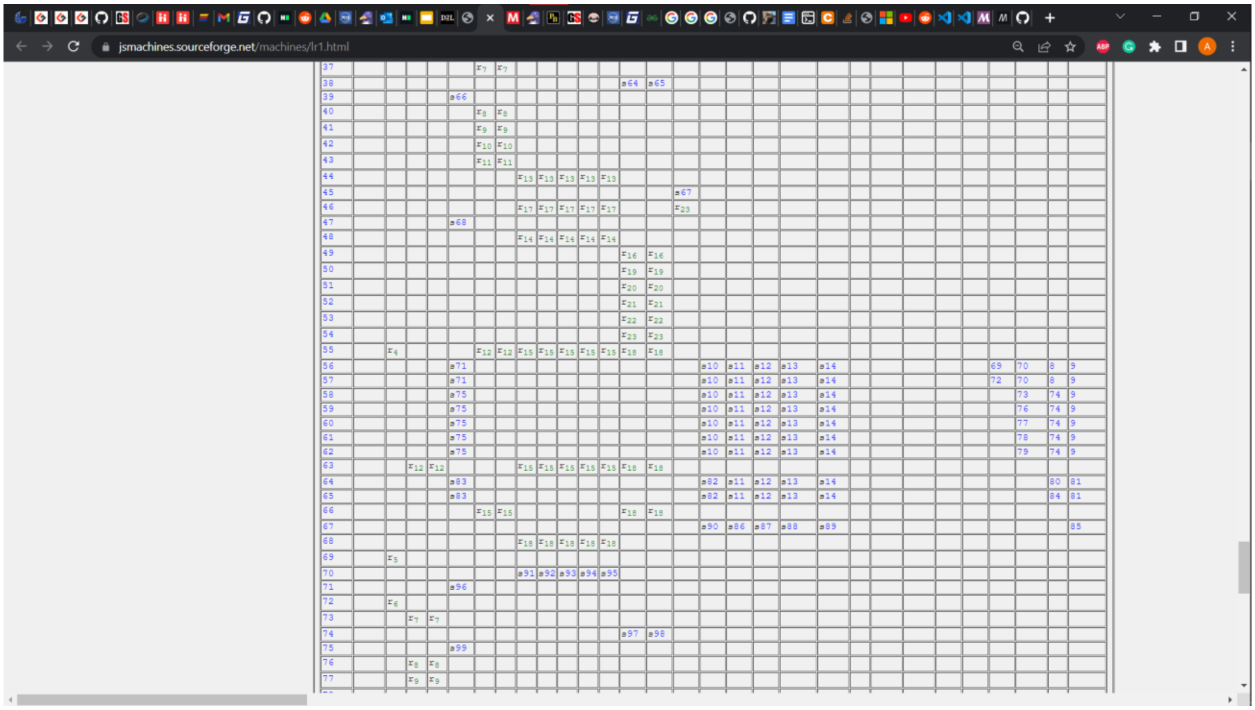
State	ACTION																				GOTO							
	start	end	+	*	'	-	/	<	>	==	<=	>=	=	!=	%	;	id	num	(expr)	numid	$	begin	state	expr	term	comp	equal	mod	factor
0	s1																												
1					s5											s10	s11	s12	s13	s14			2	3	4	6	7	8	9
2		s15																											
3		r_{1}																											
4			s16	s17																									
5					s18																								
6						s19	s20																						
7								s21	s22	s23	s24	s25																	
8													s26	s27															
9															s28														
10													r_{17}	r_{17}	r_{23}														
11															r_{19}														
12															r_{20}														
13															r_{21}														
14															r_{22}														
15																					acc								
16					s30											s10	s11	s12	s13	s14					29	31	7	8	9
17					s30											s10	s11	s12	s13	s14					32	31	7	8	9
18			r_{4}	r_{4}		r_{12}	r_{12}	r_{15}	r_{15}	r_{15}	r_{15}	r_{15}	r_{18}	r_{18}															
19					s35											s10	s11	s12	s13	s14						33	34	8	9
20					s35											s10	s11	s12	s13	s14						36	34	8	9
21					s39											s10	s11	s12	s13	s14							37	38	9
22					s39											s10	s11	s12	s13	s14							40	38	9
23					s39											s10	s11	s12	s13	s14							41	38	9
24					s39											s10	s11	s12	s13	s14							42	38	9
25					s39											s10	s11	s12	s13	s14							43	38	9
26					s47											s46	s11	s12	s13	s14								44	45
27					s47											s46	s11	s12	s13	s14								48	45
28																s54	s50	s51	s52	s53									49
29		r_{2}																											
30					s55																								
31						s56	s57																						
32		r_{3}																											
33			r_{5}	r_{5}																									
34								s58	s59	s60	s61	s62																	
35					s63																								
36			r_{6}	r_{6}																									
37						r_{7}	r_{7}																						

Input (tokens): start

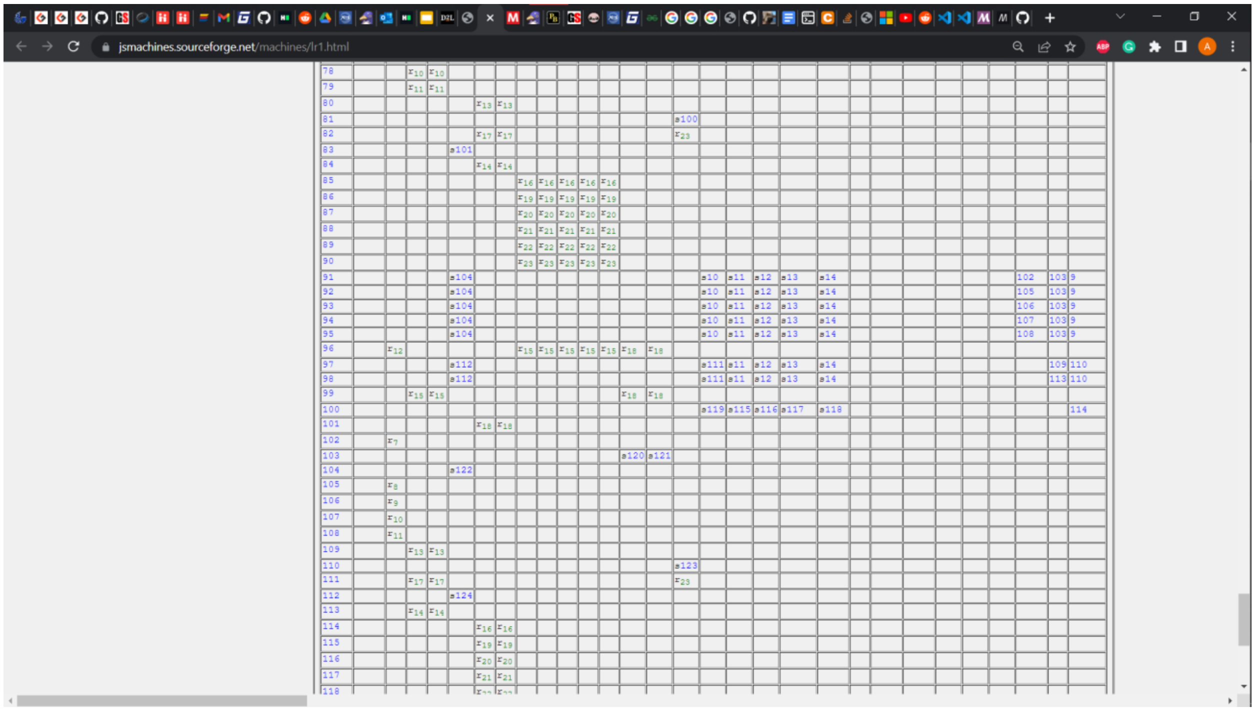
Maximum number of st

PARSE

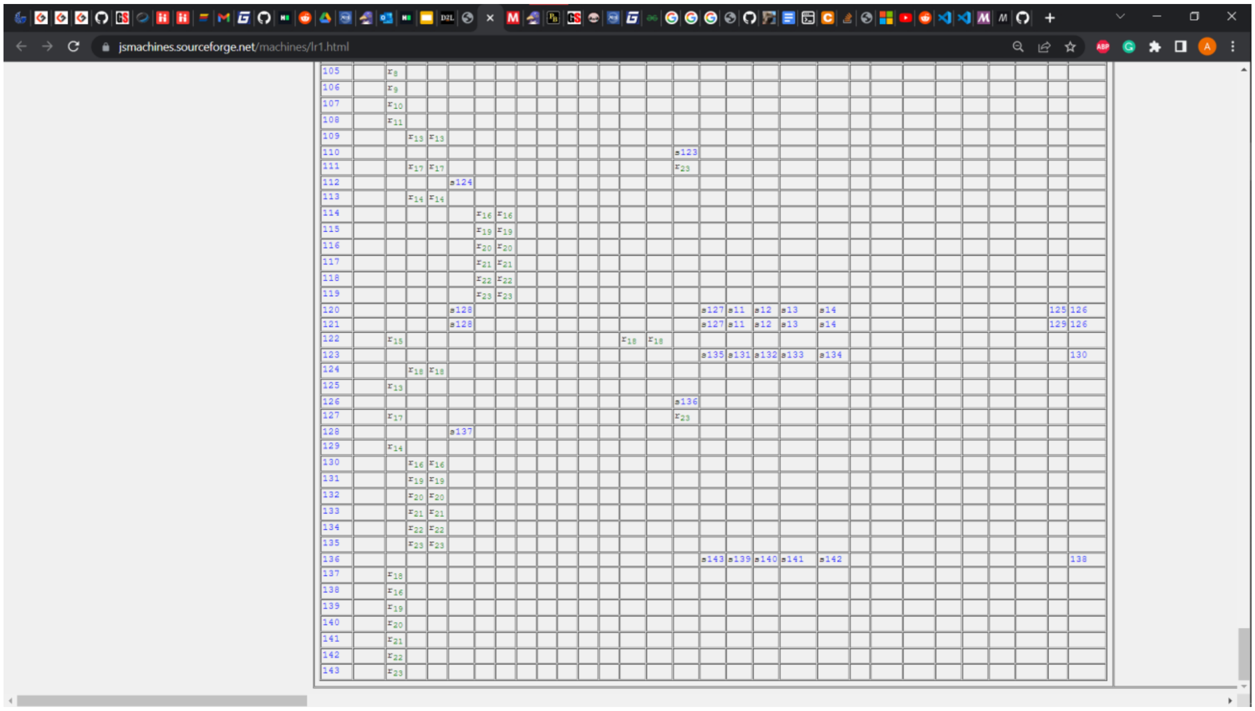
Step	S
1	0
2	0 start 1
3	0 start 1 num 1
4	0 start 1 facto
5	0 start 1 facto
6	0 start 1 facto
7	0 start 1 facto
8	0 start 1 facto
9	0 start 1 facto
10	0 start 1 mod
11	0 start 1 mod 8
12	0 start 1 mod 8
13	0 start 1 mod 8



37						r_{7}	r_{7}																						
38													s64	s65															
39					s66																								
40						r_{8}	r_{8}																						
41						r_{9}	r_{9}																						
42						r_{10}	r_{10}																						
43						r_{11}	r_{11}																						
44								r_{13}	r_{13}	r_{13}	r_{13}	r_{13}																	
45															s67														
46								r_{17}	r_{17}	r_{17}	r_{17}	r_{17}			r_{23}														
47					s68																								
48								r_{14}	r_{14}	r_{14}	r_{14}	r_{14}																	
49													r_{16}	r_{16}															
50													r_{19}	r_{19}															
51													r_{20}	r_{20}															
52													r_{21}	r_{21}															
53													r_{22}	r_{22}															
54													r_{23}	r_{23}															
55		r_{4}				r_{12}	r_{12}	r_{15}	r_{15}	r_{15}	r_{15}	r_{15}	r_{18}	r_{18}															
56					s71											s10	s11	s12	s13	s14						69	70	8	9
57					s71											s10	s11	s12	s13	s14						72	70	8	9
58					s75											s10	s11	s12	s13	s14							73	74	9
59					s75											s10	s11	s12	s13	s14							76	74	9
60					s75											s10	s11	s12	s13	s14							77	74	9
61					s75											s10	s11	s12	s13	s14							78	74	9
62					s75											s10	s11	s12	s13	s14							79	74	9
63			r_{12}	r_{12}				r_{15}	r_{15}	r_{15}	r_{15}	r_{15}	r_{18}	r_{18}															
64					s83											s82	s11	s12	s13	s14								80	81
65					s83											s82	s11	s12	s13	s14								84	81
66						r_{15}	r_{15}						r_{18}	r_{18}															
67																s90	s86	s87	s88	s89									85
68								r_{18}	r_{18}	r_{18}	r_{18}	r_{18}																	
69		r_{5}																											
70								s91	s92	s93	s94	s95																	
71					s96																								
72		r_{6}																											
73			r_{7}	r_{7}																									
74													s97	s98															
75					s99																								
76			r_{8}	r_{8}																									
77			r_{9}	r_{9}																									



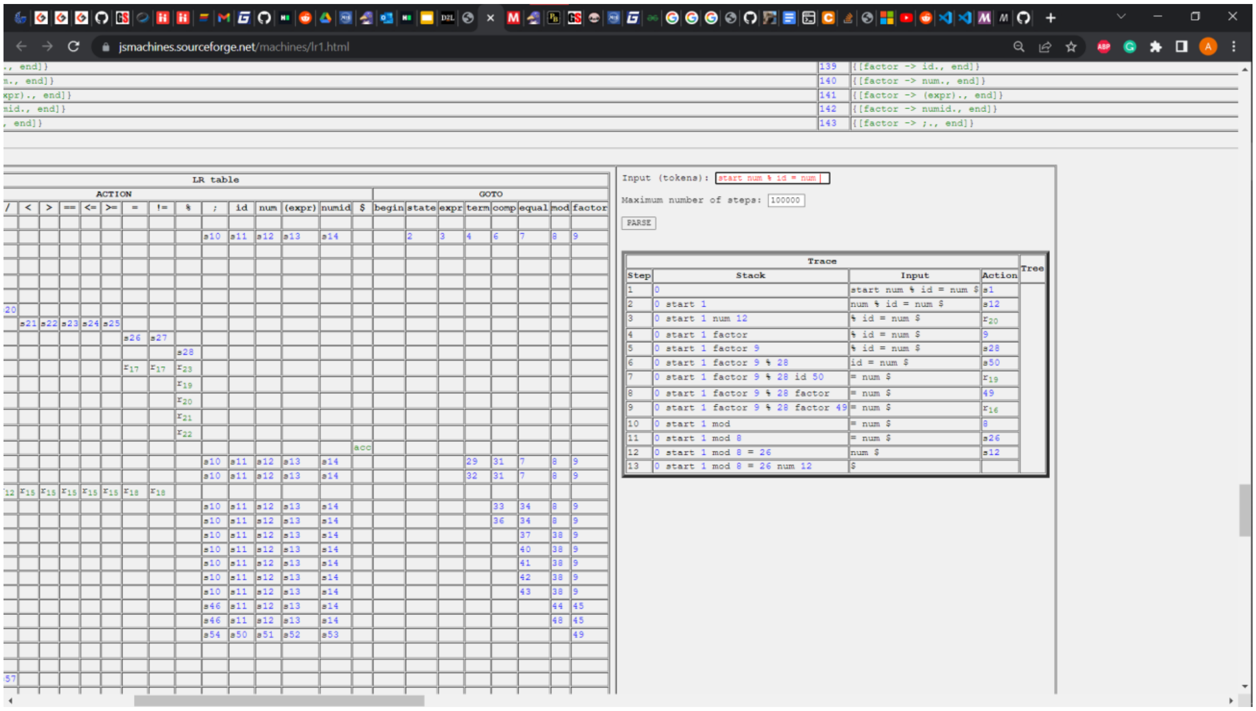
78			r_{10}	r_{10}																									
79			r_{11}	r_{11}																									
80						r_{13}	r_{13}																						
81															s100														
82						r_{17}	r_{17}								r_{23}														
83					s101																								
84						r_{14}	r_{14}																						
85								r_{16}	r_{16}	r_{16}	r_{16}	r_{16}																	
86								r_{19}	r_{19}	r_{19}	r_{19}	r_{19}																	
87								r_{20}	r_{20}	r_{20}	r_{20}	r_{20}																	
88								r_{21}	r_{21}	r_{21}	r_{21}	r_{21}																	
89								r_{22}	r_{22}	r_{22}	r_{22}	r_{22}																	
90								r_{23}	r_{23}	r_{23}	r_{23}	r_{23}																	
91					s104											s10	s11	s12	s13	s14							102	103	9
92					s104											s10	s11	s12	s13	s14							105	103	9
93					s104											s10	s11	s12	s13	s14							106	103	9
94					s104											s10	s11	s12	s13	s14							107	103	9
95					s104											s10	s11	s12	s13	s14							108	103	9
96		r_{12}						r_{15}	r_{15}	r_{15}	r_{15}	r_{15}	r_{18}	r_{18}															
97					s112											s111	s11	s12	s13	s14								109	110
98					s112											s111	s11	s12	s13	s14								113	110
99			r_{15}	r_{15}									r_{18}	r_{18}															
100																s119	s115	s116	s117	s118									114
101						r_{18}	r_{18}																						
102		r_{7}																											
103													s120	s121															
104					s122																								
105		r_{8}																											
106		r_{9}																											
107		r_{10}																											
108		r_{11}																											
109			r_{13}	r_{13}																									
110															s123														
111			r_{17}	r_{17}											r_{23}														
112					s124																								
113			r_{14}	r_{14}																									
114						r_{16}	r_{16}																						
115						r_{19}	r_{19}																						
116						r_{20}	r_{20}																						
117						r_{21}	r_{21}																						
118						r_{22}	r_{22}																						



jsmachines.sourceforge.net/machines/lr1.html

105		r_{8}																											
106		r_{9}																											
107		r_{10}																											
108		r_{11}																											
109			r_{13}	r_{13}																									
110															s123														
111			r_{17}	r_{17}											r_{23}														
112					s124																								
113			r_{14}	r_{14}																									
114						r_{16}	r_{16}																						
115						r_{19}	r_{19}																						
116						r_{20}	r_{20}																						
117						r_{21}	r_{21}																						
118						r_{22}	r_{22}																						
119						r_{23}	r_{23}																						
120					s128											s127	s11	s12	s13	s14								125	126
121					s128											s127	s11	s12	s13	s14								129	126
122		r_{15}											r_{18}	r_{18}															
123																s135	s131	s132	s133	s134									130
124			r_{18}	r_{18}																									
125		r_{13}																											
126															s136														
127		r_{17}													r_{23}														
128					s137																								
129		r_{14}																											
130			r_{16}	r_{16}																									
131			r_{19}	r_{19}																									
132			r_{20}	r_{20}																									
133			r_{21}	r_{21}																									
134			r_{22}	r_{22}																									
135			r_{23}	r_{23}																									
136																s143	s139	s140	s141	s142									138
137		r_{18}																											
138		r_{16}																											
139		r_{19}																											
140		r_{20}																											
141		r_{21}																											
142		r_{22}																											
143		r_{23}																											

The one that pass



jsmachines.sourceforge.net/machines/lr1.html

```
., end]}                                   139 {[factor -> id., end]}
n., end]}                                  140 {[factor -> num., end]}
xpr)., end]}                               141 {[factor -> (expr)., end]}
nid., end]}                                142 {[factor -> numid., end]}
, end]}                                    143 {[factor -> ;., end]}
```

LR table

ACTION															GOTO							
/	<	>	==	<=	>=	=	!=	%	;	id	num	(expr)	numid	$	begin	state	expr	term	comp	equal	mod	factor
									s10	s11	s12	s13	s14			2	3	4	6	7	8	9
20																						
	s21	s22	s23	s24	s25																	
						s26	s27															
								s28														
						r_{17}	r_{17}	r_{23}														
								r_{19}														
								r_{20}														
								r_{21}														
								r_{22}														
														acc								
									s10	s11	s12	s13	s14					29	31	7	8	9
									s10	s11	s12	s13	s14					32	31	7	8	9
$_{12}$	r_{15}	r_{15}	r_{15}	r_{15}	r_{15}	r_{18}	r_{18}															
									s10	s11	s12	s13	s14						33	34	8	9
									s10	s11	s12	s13	s14						36	34	8	9
									s10	s11	s12	s13	s14							37	38	9
									s10	s11	s12	s13	s14							40	38	9
									s10	s11	s12	s13	s14							41	38	9
									s10	s11	s12	s13	s14							42	38	9
									s10	s11	s12	s13	s14							43	38	9
									s46	s11	s12	s13	s14								44	45
									s46	s11	s12	s13	s14								48	45
									s54	s50	s51	s52	s53									49
57																						

Input (tokens): start num % id = num

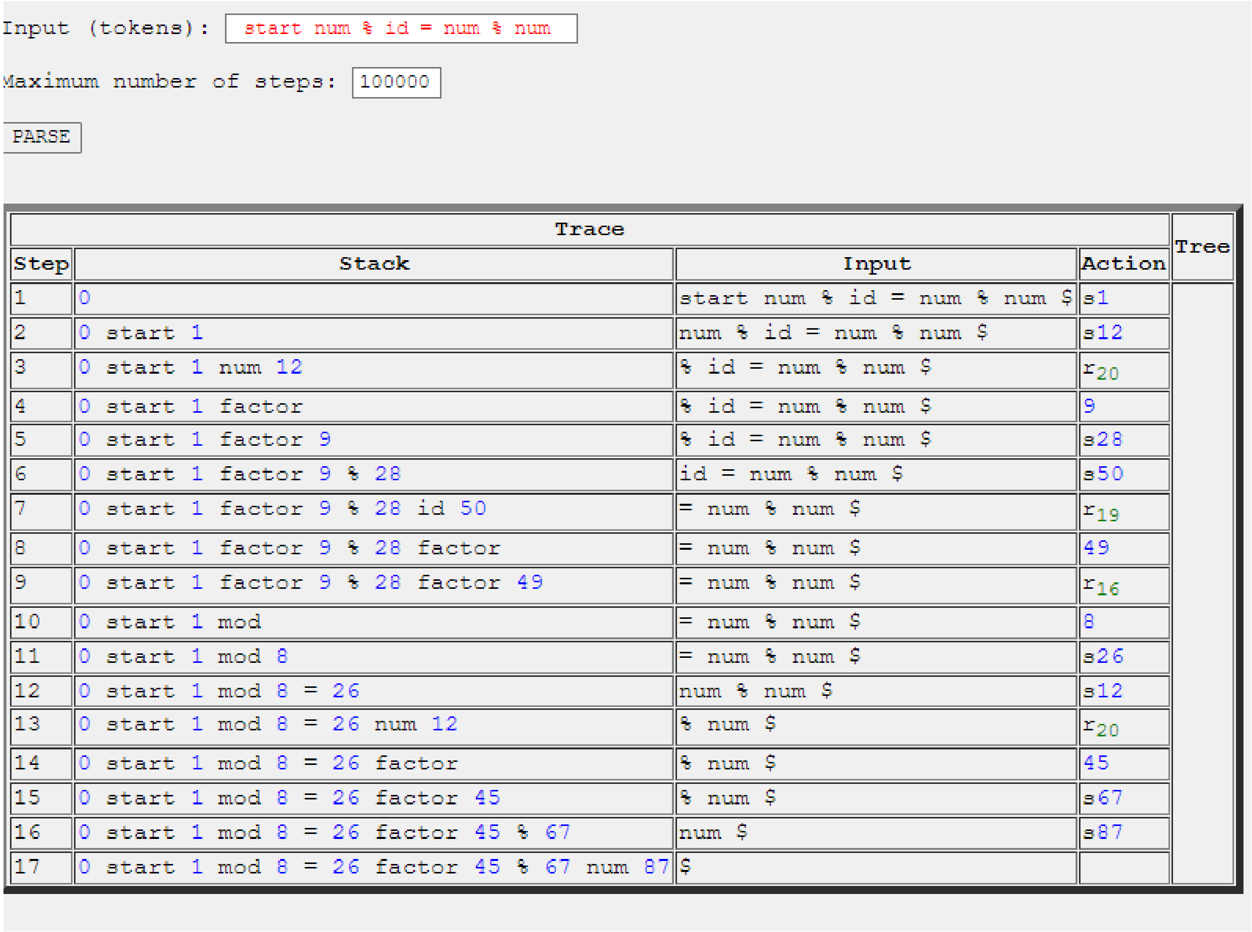
Maximum number of steps: 100000

PARSE

Trace

Step	Stack	Input	Action	Tree
1	0	start num % id = num $	s1	
2	0 start 1	num % id = num $	s12	
3	0 start 1 num 12	% id = num $	r_{20}	
4	0 start 1 factor	% id = num $	9	
5	0 start 1 factor 9	% id = num $	s28	
6	0 start 1 factor 9 % 28	id = num $	s50	
7	0 start 1 factor 9 % 28 id 50	= num $	r_{19}	
8	0 start 1 factor 9 % 28 factor	= num $	49	
9	0 start 1 factor 9 % 28 factor 49	= num $	r_{16}	
10	0 start 1 mod	= num $	8	
11	0 start 1 mod 8	= num $	s26	
12	0 start 1 mod 8 = 26	num $	s12	
13	0 start 1 mod 8 = 26 num 12	$		

It pass the test at step 13



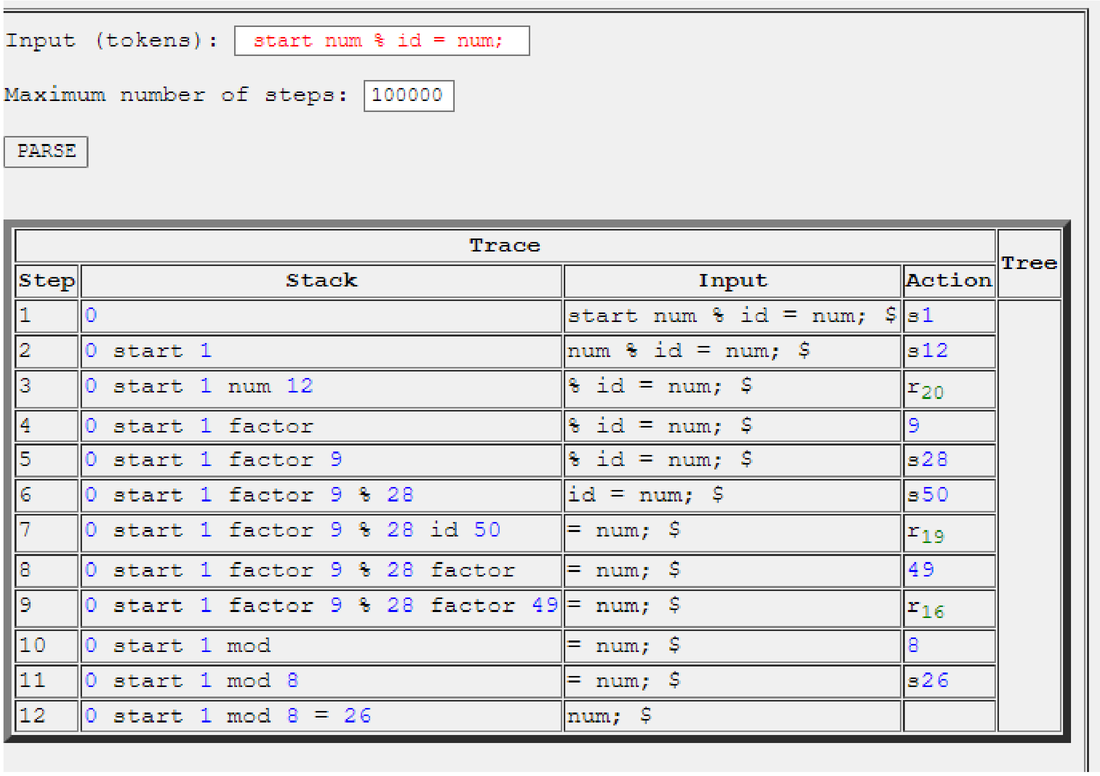
Input (tokens): start num % id = num % num

Maximum number of steps: 100000

Trace				Tree
Step	Stack	Input	Action	
1	0	start num % id = num % num $	s1	
2	0 start 1	num % id = num % num $	s12	
3	0 start 1 num 12	% id = num % num $	r$_{20}$	
4	0 start 1 factor	% id = num % num $	9	
5	0 start 1 factor 9	% id = num % num $	s28	
6	0 start 1 factor 9 % 28	id = num % num $	s50	
7	0 start 1 factor 9 % 28 id 50	= num % num $	r$_{19}$	
8	0 start 1 factor 9 % 28 factor	= num % num $	49	
9	0 start 1 factor 9 % 28 factor 49	= num % num $	r$_{16}$	
10	0 start 1 mod	= num % num $	8	
11	0 start 1 mod 8	= num % num $	s26	
12	0 start 1 mod 8 = 26	num % num $	s12	
13	0 start 1 mod 8 = 26 num 12	% num $	r$_{20}$	
14	0 start 1 mod 8 = 26 factor	% num $	45	
15	0 start 1 mod 8 = 26 factor 45	% num $	s67	
16	0 start 1 mod 8 = 26 factor 45 % 67	num $	s87	
17	0 start 1 mod 8 = 26 factor 45 % 67 num 87	$		

It pass the test at step 17

The one that fail



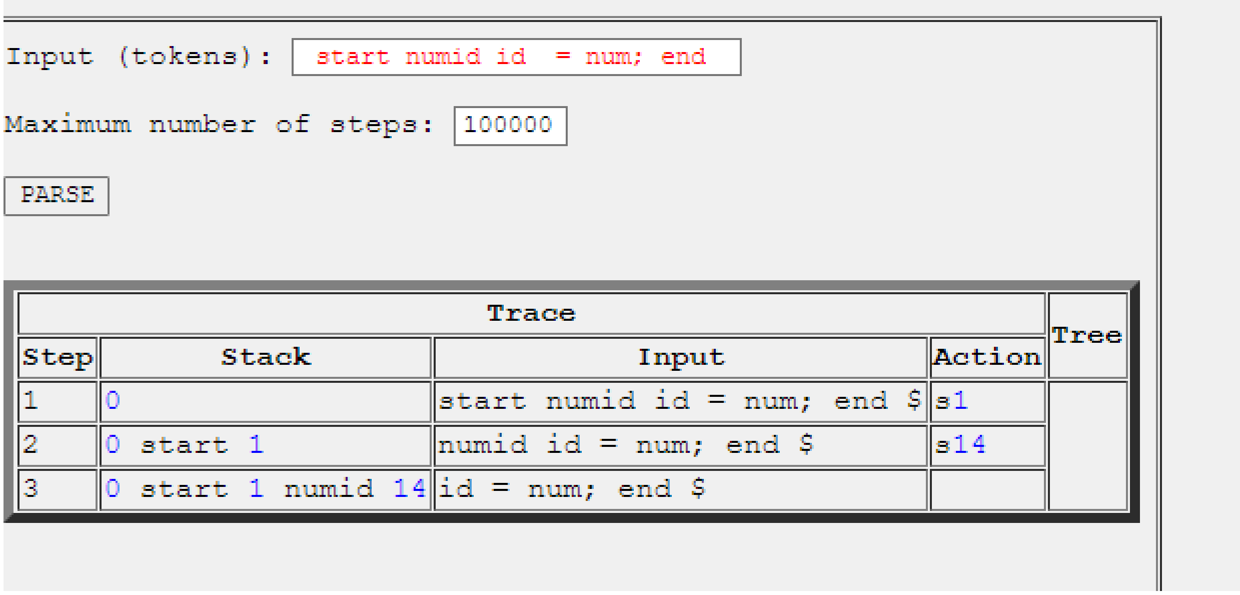
Input (tokens): start num % id = num;

Maximum number of steps: 100000

PARSE

Trace				Tree
Step	Stack	Input	Action	
1	0	start num % id = num; $	s1	
2	0 start 1	num % id = num; $	s12	
3	0 start 1 num 12	% id = num; $	r_{20}	
4	0 start 1 factor	% id = num; $	9	
5	0 start 1 factor 9	% id = num; $	s28	
6	0 start 1 factor 9 % 28	id = num; $	s50	
7	0 start 1 factor 9 % 28 id 50	= num; $	r_{19}	
8	0 start 1 factor 9 % 28 factor	= num; $	49	
9	0 start 1 factor 9 % 28 factor 49	= num; $	r_{16}	
10	0 start 1 mod	= num; $	8	
11	0 start 1 mod 8	= num; $	s26	
12	0 start 1 mod 8 = 26	num; $		

It fail at step 12



Input (tokens): start numid id = num; end

Maximum number of steps: 100000

PARSE

Trace				Tree
Step	Stack	Input	Action	
1	0	start numid id = num; end $	s1	
2	0 start 1	numid id = num; end $	s14	
3	0 start 1 numid 14	id = num; end $		

It fails at step 3