LL(1) Parser Generator. First, Follow, & Predict Sets. Table

Overview

Given a grammar in (limited) EBNF, this online tool automatically calculates the first, follow, and predict sets. It also generates LL(1) parser tables from the predict sets, as done by Fischer & LeBlanc.

The sets are shown in two formats: human-friendly tables, and machine-friendly JSON dumps. Use a JSON library to read those tables into your programs to rapidly iterate on your parser's design.

First Set

Non-Terminal Symbol	First Set
if	if
else	else
while	while
break	break
;	;
{	{
}	}
ε	ε
continue	continue
return	return
=	=
*=	*=
/=	/=
+=	+=
-=	-=
&&	&&
==	==
!=	!=
<	<
<=	<=
>	>
>=	>=
+	+
-	-
*	*
/	/
!	!
++	++
--	--
.	.
identifier	identifier
(	(
)	)
INT-LITERAL	INT-LITERAL
BOOL-LITERAL	BOOL-LITERAL
,	,

var	var
class	class
const	const
:	:
int	int
bool	bool
if-statement	if
while-statement	while
break-statement	break
compound-statement	{
statement-list	ε, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class
continue-statement	continue
return-statement	return
expression-statement	;, ε, -, !, ++, --
expression-list	ε, -, !, ++, --
class-body	{
variable-declaration-list	ε, var
assignment-operator	=, *=, /=, +=, -=
condition-or-expression-tail	ε, -, !, ++, --
condition-and-expression-tail	&&, ε
equality-expression-tail	ε, ==, !=
rel-expression-tail	ε, <, <=, >, >=
additive-expression-tail	ε, +, -
m-d-expression-tail	ε, *, /
u-expression	-, !, ++, --
post-expression-tail	., ++, --, ε
primary-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
para-list	(
proper-para-list-tail	,, ε
arg-list	(
proper-arg-list-tail	,, ε
function-declaration	identifier
variable-declaration	var
class-declaration	class
constant-declaration	const
init-expression	=
type-annotation	:
type	int, bool
top-level	ε, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class
statement	{, while, continue, if, return, break, ;, ε, -, !, ++, --, identifier, var, const, class
m-d-expression	-, !, ++, --
post-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
para-declaration	int, bool
declaration-statement	identifier, var, const, class
additive-expression	-, !, ++, --
proper-para-list	int, bool
rel-expression	-, !, ++, --
equality-expression	-, !, ++, --
condition-and-expression	-, !, ++, --
condition-or-expression	-, !, ++, --
assignment-expression	-, !, ++, --
expression	-, !, ++, --
arg	-, !, ++, --
proper-arg-list	-, !, ++, --

Follow Set

Non-Terminal Symbol	Follow Set
statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
if-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
while-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
break-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
compound-statement	else, $, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
statement-list	}
continue-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
return-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
expression-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
expression-list	;
class-body	

variable-declaration-list	}
expression	), ;, {, ,
assignment-expression	), ;, {, ,
assignment-operator	
condition-or-expression	), ;, {, ,
condition-or-expression-tail	), ;, {, ,
condition-and-expression	-, !, ++, --,), ;, {, ,
condition-and-expression-tail	-, !, ++, --,), ;, {, ,
equality-expression	==, !=, &&, -, !, ++, --,), ;, {, ,
equality-expression-tail	==, !=, &&, -, !, ++, --,), ;, {, ,
rel-expression	==, !=, &&, -, !, ++, --,), ;, {, ,
rel-expression-tail	==, !=, &&, -, !, ++, --,), ;, {, ,
additive-expression	<, <=, >, >=, ==, !=, &&, -, !, ++, --,), ;, {, ,
additive-expression-tail	<, <=, >, >=, ==, !=, &&, -, !, ++, --,), ;, {, ,
m-d-expression	+, -, <, <=, >, >=, ==, !=, &&, !, ++, --,), ;, {, ,
m-d-expression-tail	+, -, <, <=, >, >=, ==, !=, &&, !, ++, --,), ;, {, ,
u-expression	*, /, +, -, <, <=, >, >=, ==, !=, &&, !, ++, --,), ;, {, ,
post-expression	*, /, +, -, <, <=, >, >=, ==, !=, &&, !, ++, --,), ;, {, ,
post-expression-tail	*, /, +, -, <, <=, >, >=, ==, !=, &&, !, ++, --,), ;, {, ,
primary-expression	., ++, --, *, /, +, -, <, <=, >, >=, ==, !=, &&, !,), ;, {, ,
para-list	{
proper-para-list	)
proper-para-list-tail	)
para-declaration	,,)
arg-list	., ++, --, *, /, +, -, <, <=, >, >=, ==, !=, &&, !,), ;, {, ,
proper-arg-list	)
proper-arg-list-tail	)
arg	,,)
declaration-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
function-declaration	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
variable-declaration	var, $, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, const, class, }
class-declaration	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
constant-declaration	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class, }
init-expression	;
type-annotation	;
type	identifier, ;
top-level	

Predict Set

#	Expression	Predict
1	statement → compound-statement	{
2	statement → if-statement	if
3	statement → while-statement	while
4	statement → break-statement	break
5	statement → continue-statement	continue
6	statement → return-statement	return
7	statement → expression-statement	;, -, !, ++, --
8	statement → declaration-statement	identifier, var, const, class
9	if-statement → if expression compound-statement else compound-statement	if
10	while-statement → while expression compound-statement	while
11	break-statement → break ;	break
12	compound-statement → { statement-list }	{
13	statement-list → ε	}
14	statement-list → statement statement-list	{, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class
15	continue-statement → continue ;	continue
16	return-statement → return expression ;	return
17	return-statement → return ;	return
18	expression-statement → expression-list ;	-, !, ++, --, ;
19	expression-list → expression	-, !, ++, --
20	expression-list → ε	;
21	class-body → { variable-declaration-list }	{
22	variable-declaration-list → variable-declaration variable-declaration-list	var
23	variable-declaration-list → ε	}
24	expression → assignment-expression	-, !, ++, --
25	assignment-expression → condition-or-expression	-, !, ++, --
26	assignment-operator → =	=

27	assignment-operator → *=	*=
28	assignment-operator → /=	/=
29	assignment-operator → +=	+=
30	assignment-operator → -=	-=
31	condition-or-expression → condition-and-expression condition-or-expression-tail	-, !, ++, --
32	condition-or-expression-tail → ε	), ;, {, ,
33	condition-or-expression-tail → condition-and-expression condition-or-expression-tail	-, !, ++, --
34	condition-and-expression → equality-expression condition-and-expression-tail	-, !, ++, --
35	condition-and-expression-tail → && equality-expression equality-expression-tail	&&
36	condition-and-expression-tail → ε	-, !, ++, --,), ;, {, ,
37	equality-expression → rel-expression equality-expression-tail	-, !, ++, --
38	equality-expression-tail → ε	==, !=, &&, -, !, ++, --,), ;, {, ,
39	equality-expression-tail → == rel-expression equality-expression-tail	==
40	equality-expression-tail → != rel-expression equality-expression-tail	!=
41	rel-expression → additive-expression rel-expression-tail	-, !, ++, --
42	rel-expression-tail → ε	==, !=, &&, -, !, ++, --,), ;, {, ,
43	rel-expression-tail → < additive-expression rel-expression-tail	<
44	rel-expression-tail → <= additive-expression rel-expression-tail	<=
45	rel-expression-tail → > additive-expression rel-expression-tail	>
46	rel-expression-tail → >= additive-expression rel-expression-tail	>=
47	additive-expression → m-d-expression additive-expression-tail	-, !, ++, --
48	additive-expression-tail → ε	<, <=, >, >=, ==, !=, &&, -, !, ++, --,), ;, {, ,
49	additive-expression-tail → + m-d-expression additive-expression-tail	+
50	additive-expression-tail → - m-d-expression additive-expression-tail	-
51	m-d-expression → u-expression m-d-expression-tail	-, !, ++, --
52	m-d-expression-tail → ε	+, -, <, <=, >, >=, ==, !=, &&, !, ++, --,), ;, {, ,
53	m-d-expression-tail → * u-expression m-d-expression-tail	*
54	m-d-expression-tail → / u-expression m-d-expression-tail	/
55	u-expression → - u-expression	-
56	u-expression → ! u-expression	!
57	u-expression → ++ post-expression	++
58	u-expression → -- post-expression	--
59	post-expression → primary-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
60	post-expression → primary-expression post-expression-tail	identifier, (, INT-LITERAL, BOOL-LITERAL
61	post-expression-tail → . identifier post-expression-tail	.
62	post-expression-tail → ++ post-expression	++
63	post-expression-tail → -- post-expression	--
64	post-expression-tail → ε	*, /, +, -, <, <=, >, >=, ==, !=, &&, !, ++, --,), ;, {, ,
65	primary-expression → identifier	identifier
66	primary-expression → identifier arg-list	identifier
67	primary-expression → (expression)	(
68	primary-expression → INT-LITERAL	INT-LITERAL
69	primary-expression → BOOL-LITERAL	BOOL-LITERAL
70	para-list → ( )	(
71	para-list → (proper-para-list)	(
72	proper-para-list → para-declaration proper-para-list-tail	int, bool
73	proper-para-list-tail → , para-declaration proper-para-list-tail	,
74	proper-para-list-tail → ε	)
75	para-declaration → type identifier	int, bool
76	arg-list → ( )	(
77	arg-list → (proper-arg-list)	(
78	proper-arg-list → arg proper-arg-list-tail	-, !, ++, --
79	proper-arg-list-tail → , arg proper-arg-list-tail	,
80	proper-arg-list-tail → ε	)
81	arg → expression	-, !, ++, --
82	declaration-statement → function-declaration	identifier
83	declaration-statement → constant-declaration	const
84	declaration-statement → variable-declaration	var
85	declaration-statement → class-declaration	class
86	function-declaration → identifier para-list compound-statement	identifier
87	variable-declaration → var identifier init-expression ;	var

88	variable-declaration → var identifier type-annotation ;	var
89	class-declaration → class identifier init-expression ;	class
90	class-declaration → class identifier type-annotation ;	class
91	constant-declaration → const identifier init-expression ;	const
92	constant-declaration → const identifier type-annotation ;	const
93	init-expression → = expression	=
94	type-annotation → : type	:
95	type → int	int
96	type → bool	bool
97	top-level → statement top-level	{, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class
98	top-level → ε	

LL(1) Parsing Table

On the LL(1) Parsing Table's Meaning and Construction

- The top row corresponds to the columns for all the potential terminal symbols, augmented with $ to represent the end of the parse.
- The leftmost column and second row are all zero filled, to accomodate the way Fischer and LeBlanc wrote their parser's handling of abs().
- The remaining rows correspond to production rules in the original grammar that you typed in.
- Each entry in that row maps the left-hand-side (LHS) of a production rule onto a line-number. That number is the line in which the LHS had that specific column symbol in its predict set.

- If a terminal is absent from a non-terminal's predict set, an error code is placed in the table. If that terminal is in follow(that non-terminal), the error is a POP error. Else, it's a SCAN error.

  POP error code = # of predict table productions + 1

  SCAN error code = # of predict table productions + 2

In practice, you'd want to tear the top, label row off of the table and stick it in a comment, so that you can make sense of your table. The remaining table can be used as is.

LL(1) Parsing Table as JSON (for Easy Import)

```
[[0,"if","else","while","break",";","{","}","continue","return","=","*=","/=","+=","-=","&&","==","!=","<","<=",">",">=","+","-
","*","/","!","++","--",".","identifier","(",")","INT-LITERAL","BOOL-LITERAL",",","var","class","const",":","int","bool","$"],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,2,100,3,4,7,1,99,5,6,100,100,100,100,100,100,100,100,100,100,100,100,100,7,100,100,7,7,7,100,8,100,100,100,100,100,8,8,8,100,100,1
[0,9,100,99,99,99,99,99,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,99,99,99,100,99,100,100,100,100,100,99,9
[0,99,100,10,99,99,99,99,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,99,99,99,100,99,100,100,100,100,100,99,
[0,99,100,99,11,99,99,99,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,99,99,99,100,99,100,100,100,100,100,99,
[0,99,99,99,99,99,12,99,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,99,99,99,100,99,100,100,100,100,100,99,9
[0,14,100,14,14,14,14,13,14,14,100,100,100,100,100,100,100,100,100,100,100,100,100,14,100,100,14,14,14,100,14,100,100,100,100,100,14,1
[0,99,100,99,99,99,99,99,15,99,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,99,99,99,100,99,100,100,100,100,100,99,
[0,99,100,99,99,99,99,99,99,17,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,99,99,99,100,99,100,100,100,100,100,99,
[0,99,100,99,99,18,99,99,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,18,100,100,18,18,18,100,99,100,100,100,100,100,99,
[0,100,100,100,100,20,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,19,100,100,19,19,19,100,100,100,100,100,100,100,
[0,100,100,100,100,100,21,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,
[0,100,100,100,100,100,100,23,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,24,100,100,24,24,24,100,100,100,99,100,100,9
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,25,100,100,25,25,25,100,100,100,99,100,100,9
[0,100,100,100,100,100,100,100,100,100,26,27,28,29,30,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,31,100,100,31,31,31,100,100,100,99,100,100,9
[0,100,100,100,100,32,32,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,33,100,100,33,33,33,100,100,100,32,100,100,3
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,34,100,100,34,34,34,100,100,100,99,100,100,9
[0,100,100,100,100,36,36,100,100,100,100,100,100,100,100,35,100,100,100,100,100,100,36,100,100,36,36,36,100,100,100,36,100,100,36,
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,99,99,99,100,100,100,100,100,37,100,100,37,37,37,100,100,100,99,100,100,99,1
[0,100,100,100,100,38,38,100,100,100,100,100,100,100,100,38,39,40,100,100,100,100,100,38,100,100,38,38,38,100,100,100,38,100,100,38,1
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,99,99,99,100,100,100,100,100,41,100,100,41,41,41,100,100,100,99,100,100,99,1
[0,100,100,100,100,42,42,100,100,100,100,100,100,100,100,42,42,42,43,44,45,46,100,42,100,100,42,42,42,100,100,100,42,100,100,42,100,1
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,99,99,99,99,99,99,99,100,47,100,100,47,47,47,100,100,100,99,100,100,99,100,1
[0,100,100,100,100,48,48,100,100,100,100,100,100,100,100,48,48,48,48,48,48,48,49,50,100,100,48,48,48,100,100,100,48,100,100,48,100,10
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,99,99,99,99,99,99,99,99,51,100,100,51,51,51,100,100,100,99,100,100,99,100,10
[0,100,100,100,100,52,52,100,100,100,100,100,100,100,100,52,52,52,52,52,52,52,52,53,54,52,52,52,100,100,100,52,100,100,52,100,100,1
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,99,99,99,99,99,99,99,99,55,99,99,56,57,58,100,100,100,99,100,100,99,100,100,1
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,99,99,99,99,99,99,99,99,99,99,99,99,99,100,60,60,99,60,60,99,100,100,100,1
[0,100,100,100,100,64,64,100,100,100,100,100,100,100,100,64,64,64,64,64,64,64,64,64,64,64,64,64,61,100,100,64,100,100,64,100,100,1
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,99,99,99,99,99,99,99,99,99,99,99,99,99,66,67,99,68,69,99,100,100,100,1
[0,100,100,100,100,100,99,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,71,100,100,1
[0,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,
[0,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,74,100,
[0,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,
[0,100,100,100,100,99,99,100,100,100,100,100,100,100,100,99,99,99,99,99,99,99,99,99,99,99,99,99,100,77,99,100,100,99,100,100,10
[0,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,78,100,100,78,78,78,100,100,100,99,100,100,
[0,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,80,100,
[0,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,81,100,100,81,81,81,100,100,100,99,100,100,
[0,99,100,99,99,99,99,99,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,99,99,99,100,82,100,100,100,100,100,84,8
[0,99,100,99,99,99,99,99,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,99,99,99,100,86,100,100,100,100,100,99,
[0,99,100,99,99,99,99,99,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,99,99,99,100,99,100,100,100,100,100,88,
[0,99,100,99,99,99,99,99,99,99,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,99,99,99,100,99,100,100,100,100,100,99,
[0,99,100,99,99,99,99,99,99,99,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,99,99,99,100,99,100,100,100,100,100,99,9
[0,100,100,100,100,99,100,100,100,100,93,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,1
[0,100,100,100,100,99,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,
[0,100,100,100,100,99,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,99,100,100,100,1
[0,97,100,97,97,97,97,100,97,97,100,100,100,100,100,100,100,100,100,100,100,100,97,100,100,97,97,97,100,97,100,100,100,100,100,97,
```

LL(1) Parsing Push-Map (as JSON)

This structure maps each production rule in the expanded grammar (seen as the middle column in the predict table above) to a series of states that the LL parser pushes onto the stack.

```
{"1":[5],"2":[2],"3":[3],"4":[4],"5":[7],"6":[8],"7":[9],"8":[40],"9":[5,-2,5,13,-1],"10":[5,13,-3],"11":[-5,-4],"12":
[-7,6,-6],"14":[6,1],"15":[-5,-8],"16":[-5,13,-9],"17":[-5,-9],"18":[-5,10],"19":[13],"21":[-7,12,-6],"22":[12,42],"24":[14],"25":
[16],"26":[-10],"27":[-11],"28":[-12],"29":[-13],"30":[-14],"31":[17,18],"33":[17,18],"34":[19,20],"35":[21,20,-15],"37":
[21,22],"39":[21,22,-16],"40":[21,22,-17],"41":[23,24],"43":[23,24,-18],"44":[23,24,-19],"45":[23,24,-20],"46":[23,24,-21],"47":
[25,26],"49":[25,26,-22],"50":[25,26,-23],"51":[27,28],"53":[27,28,-24],"54":[27,28,-25],"55":[28,-23],"56":[28,-26],"57":
[29,-27],"58":[29,-28],"59":[31],"60":[30,31],"61":[30,-30,-29],"62":[29,-27],"63":[29,-28],"65":[-30],"66":[36,-30],"67":
[-32,13,-31],"68":[-33],"69":[-34],"70":[-32,-31],"71":[-32,33,-31],"72":[34,35],"73":[34,35,-35],"75":[-30,47],"76":[-32,-31],"77":
[-32,37,-31],"78":[38,39],"79":[38,39,-35],"81":[13],"82":[41],"83":[44],"84":[42],"85":[43],"86":[5,32,-30],"87":
[-5,45,-30,-36],"88":[-5,46,-30,-36],"89":[-5,45,-30,-37],"90":[-5,46,-30,-37],"91":[-5,45,-30,-38],"92":[-5,46,-30,-38],"93":
[13,-10],"94":[47,-39],"95":[-40],"96":[-41],"97":[48,1]}
```

Feed me your delicious grammar, mortal.

```
statement ->
compound-statement |
if-statement | while-
statement | break-
statement | continue-
statement | return-
statement |
expression-statement
|declaration-
statement
if-statement -> if
expression compound-
statement else
compound-statement
while-statement ->
while expression
compound-statement
break-statement ->
break ;
compound-statement ->
```

Generate LL(1) Parsing Table, First Set, Follow Set, & Predict Set

Grammar Specification Requirements

Productions use the following format:

```
Goal -> A
A -> ( A ) | Two
Two -> a
Two -> b
```

- "->" separates the non-terminal on the left-hand-side from what it produces.
- x -> y | z is EBNF short-hand for
  x -> y
  x -> z

Be certain to place spaces between things you don't want read as one symbol. (A) ≠ (A)

About This Tool

Intended Audience

Computer science students & autodidacts studying compilers or parsing.

Purpose

This tool provides rapid feedback loop for learning about grammars. How?

- Rapid visualization of grammars enables convenient tweaking. Botched a production? No problem; tweak it and everything's spit back out.
- Ability to dump LR(0) and SLR(1) tables. Helps with manual parse tracing and hand-writing parsers.
- Assisting with coursework.

Underlying Theory

How to draw NFAs for SLR(0) and LR(1) grammars. Want to learn how it works or how to do it by hand? Read that.