HackingOff

- Home
- Blog
- Compiler Construction Toolkit
  - Overview
  -
  - Scanner Generator
  - Regex to NFA & DFA
  - NFA to DFA
  - BNF to First, Follow, & Predict sets
  -
  - Parser Generator Overview
  - LL(1) Parser Generator
  - LR(0) Parser Generator
  - SLR(1) Parser Generator

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
- Follow Set
- Predict Set
- LL(1) Table
- Grammar Input

First Set

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

86	declaration-statement → variable-declaration	var
87	declaration-statement → class-declaration	class
88	function-declaration → identifier para-list compound-statement	identifier
89	variable-declaration → var identifier init-expression ;	var
90	variable-declaration → var identifier type-annotation ;	var
91	class-declaration → class identifier init-expression ;	class
92	class-declaration → class identifier type-annotation ;	class
93	constant-declaration → const identifier init-expression ;	const
94	constant-declaration → const identifier type-annotation ;	const
95	init-expression → = expression	=
96	type-annotation → : type	:
97	type → int	int
98	type → bool	bool
99	top-level → statement top-level	{, while, continue, if, return, break, ;, -, !, ++, --, identifier, var, const, class
100	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","=","*=","/=","+=","-=","&&=","XX=","XX","&&","==","!=","<","
<=",">",">=","+","-","*","/","!","++","--",".","identifier","(",")","INT-LITERAL","BOOL-
LITERAL",",","var","class","const",":","int","bool","$"],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,2,102,3,4,7,1,101,5,6,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,7,102,102,7,7,7,102,8,102,102,102,102,102,8,
[0,9,102,101,101,101,101,101,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,101,102,102,101,101,101,102,101,
[0,101,102,10,101,101,101,101,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,101,102,102,101,101,101,102,101,
[0,101,102,101,11,101,101,101,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,101,102,102,101,101,101,102,101,
[0,101,101,101,101,101,12,101,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,101,102,102,101,101,101,102,101,
[0,14,102,14,14,14,14,13,14,14,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,14,102,102,14,14,14,102,14,102,102,102,
[0,101,102,101,101,101,101,101,15,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,101,102,102,101,101,101,102,101,
[0,101,102,101,101,101,101,101,101,17,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,101,102,102,101,101,101,102,101,
[0,101,102,101,101,18,101,101,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,18,102,102,18,18,18,102,101,102,
[0,102,102,102,102,20,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,19,102,102,19,19,19,102,102,102,
[0,102,102,102,102,102,21,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,
[0,102,102,102,102,102,102,23,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,24,102,102,24,24,24,102,102,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,25,102,102,25,25,25,102,102,
[0,102,102,102,102,102,102,102,102,102,26,27,28,29,30,31,32,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,33,102,102,33,33,33,102,102,
[0,102,102,102,102,34,34,102,102,102,102,102,102,102,102,102,102,35,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,101,102,102,102,102,102,102,102,102,36,102,102,36,36,36,102,102,
[0,102,102,102,102,38,38,102,102,102,102,102,102,102,102,102,102,38,37,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,101,101,101,101,102,102,102,102,102,39,102,102,39,39,39,102,102,
[0,102,102,102,102,40,40,102,102,102,102,102,102,102,102,102,102,40,40,41,42,102,102,102,102,102,102,102,102,102,102,102,102,102,102,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,101,101,101,101,102,102,102,102,102,43,102,102,43,43,43,102,102,
[0,102,102,102,102,44,44,102,102,102,102,102,102,102,102,102,102,44,44,44,44,45,46,47,48,102,102,102,102,102,102,102,102,102,102,44,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,101,101,101,101,101,101,101,101,102,49,102,102,49,49,49,102,102,
[0,102,102,102,102,50,50,102,102,102,102,102,102,102,102,102,102,50,50,50,50,50,50,50,50,51,52,102,102,102,102,102,102,102,102,50,102,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,101,101,101,101,101,101,101,101,101,53,102,102,53,53,53,102,102,
[0,102,102,102,102,54,54,102,102,102,102,102,102,102,102,102,102,54,54,54,54,54,54,54,54,54,54,55,56,102,102,102,102,102,102,54,102,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,101,101,101,101,101,101,101,101,101,57,101,101,58,59,60,102,102,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,101,101,101,101,101,101,101,101,101,101,101,101,102,102,102,102,62,
[0,102,102,102,102,66,66,102,102,102,102,102,102,102,102,102,102,66,66,66,66,66,66,66,66,66,66,66,102,64,65,63,102,102,66,102,102,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,101,101,101,101,101,101,101,101,101,101,101,101,102,101,101,101,68,
[0,102,102,102,102,102,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,
[0,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,
[0,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,
[0,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,
[0,102,102,102,102,101,101,102,102,102,102,102,102,102,102,102,102,101,101,101,101,101,101,101,101,101,101,101,101,102,101,101,101,
[0,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,80,102,102,80,80,80,102,102,

```
[0,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,10
[0,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,83,102,102,83,83,83,102,102,10
[0,101,102,101,101,101,101,101,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,101,102,102,101,101,101,102,84,
[0,101,102,101,101,101,101,101,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,101,102,102,101,101,101,102,88,
[0,101,102,101,101,101,101,101,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,101,102,102,101,101,101,102,10
[0,101,102,101,101,101,101,101,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,101,102,102,101,101,101,102,10
[0,101,102,101,101,101,101,101,101,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,101,102,102,101,101,101,102,10
[0,102,102,102,102,101,102,102,102,102,95,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,
[0,102,102,102,102,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,10
[0,102,102,102,102,101,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,10
[0,99,102,99,99,99,99,102,99,99,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,99,102,102,99,99,99,102,99,102,102,10
```

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":[-15],"32":[-16],"33":[17,18],"35":[17,18,-17],"36":[19,20],"37":[21,20,-18],"39":[21,22],"41":[21,22,-19],"42":[21,22,-20],"43":[23,24],"45":[23,24,-21],"46":[23,24,-22],"47":[23,24,-23],"48":[23,24,-24],"49":[25,26],"51":[25,26,-25],"52":[25,26,-26],"53":[27,28],"55":[27,28,-27],"56":[27,28,-28],"57":[28,-26],"58":[28,-29],"59":[29,-30],"60":[29,-31],"61":[31],"62":[30,31],"63":[30,-33,-32],"64":[29,-30],"65":[29,-31],"67":[-33],"68":[36,-33],"69":[-35,13,-34],"70":[-36],"71":[-37],"72":[-35,-34],"73":[-35,33,-34],"74":[34,35],"75":[34,35,-38],"77":[-33,47],"78":[-35,-34],"79":[-35,37,-34],"80":[38,39],"81":[38,39,-38],"83":[13],"84":[41],"85":[44],"86":[42],"87":[43],"88":[5,32,-33],"89":[-5,45,-33,-39],"90":[-5,46,-33,-39],"91":[-5,45,-33,-40],"92":[-5,46,-33,-40],"93":[-5,45,-33,-41],"94":[-5,46,-33,-41],"95":[13,-10],"96":[47,-42],"97":[-43],"98":[-44],"99":[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.