Generate Predict, First, and Follow Sets from EBNF (Extended Backus Naur Form) Grammar

---

Provide a grammar in Extended Backus-Naur form (EBNF) to automatically calculate its first, follow, and predict sets. See the sidebar for an example.

First sets are used in LL parsers (top-down parsers reading Left-to-right, using Leftmost-derivations).

Follow sets are used in top-down parsers, but also in LR parsers (bottom-up parsers, reading Left-to-right, using Rightmost derivations). These include LR(0), SLR(1), LR(k), and LALR parsers.

Predict sets, derived from the above two, are used by Fischer & LeBlanc to construct LL(1) top-down parsers.

Input Your Grammar

For more details, and a well-formed example, check out the sidebar. →

```
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 ->
```

Click for Predict, First, and Follow Sets

First Set

Non-Terminal	Symbol	First Set
if	if	
else	else	
while	while	
break	break	
;	;	
{	{	
}	}	
ε	ε	
continue	continue	
return	return	
condition-expression	condition-expression	
=	=	
*=	*=	
/=	/=	
+=	+=	
-=	-=	
&&=	&&=	

XX=	XX=
XX	XX
&&	&&
==	==
!=	!=
<	<
<=	<=
>	>
>=	>=
+	+
-	-
*	*
/	/
!	!
++	++
--	--
.	.
identifier	identifier
(	(
)	)
INT-LITERAL	INT-LITERAL
BOOL-LITERAL	BOOL-LITERAL
,	,
const	const
-initial	-initial
func	func
->	->
void	void
var	var
class	class
init-expression	init-expression
:	:
anonymous-annotation-internal	anonymous-annotation-internal
int	int
bool	bool
if-statement	if
while-statement	while
break-statement	break
compound-statement	{
statement-list	ε , {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var
continue-statement	continue
return-statement	return
expression-statement	;, ε , -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
expression-list	ε , -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
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	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
post-expression-tail	., ++, --, ε
primary-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
para-list	(
proper-para-list-tail	,, ε
para-declaration	const, identifier
arg-list	(
proper-arg-list-tail	,, ε
function-definition	func
return-type	void, int, bool, identifier, {
variable-declaration	var
unpack-declaration	var
unpack-initial	identifier
unpack-decls	{
unpack-decl-internal-tail	,, ε
unpack-element	identifier, {
class-declaration	class

class-body	{,
class-member	ε, const, class, var, func
constant-declaration	const
initial	=
anonymous-initial	{,
anonymous-initial-internal-tail	,, ε
type-annotation	:
anonymous	{,
anonymous-internal-tail	,, ε
anonymous-type	int, bool, {,
type	int, bool, identifier, {,
top-level	ε, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var, func
statement	{, while, continue, if, return, break, ;, ε, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var
post-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
proper-para-list	const, identifier
declaration-statement	const, class, var
unpack-decl-internal	identifier, {,
anonymous-internal	int, bool, {,
m-d-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
additive-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
rel-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
equality-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
condition-and-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
condition-or-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
assignment-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
arg	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, {,
anonymous-initial-element	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, {,
proper-arg-list	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, {,
anonymous-initial-internal	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, {,

Follow Set

Non-Terminal Symbol	Follow Set
statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var, func, }
if-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var, func, }
while-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var, func, }
break-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var, func, }
compound-statement	else, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var, func, $, }
statement-list	}
continue-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var, func, }
return-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var, func, }
expression-statement	$, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var, func, }
expression-list	;
expression	), ;, {, ,, }
assignment-expression	), ;, {, ,, }
assignment-operator	condition-expression
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=, +, -, <, <=, >, >=, ==, !=, &&, XX,), ;, {, ,, }
post-expression	*, /, =, *=, /=, +=, -=, &&=, XX=, +, -, <, <=, >, >=, ==, !=, &&, XX,), ;, {, ,, }
post-expression-tail	*, /, =, *=, /=, +=, -=, &&=, XX=, +, -, <, <=, >, >=, ==, !=, &&, XX,), ;, {, ,, }
primary-expression	., ++, --, *, /, =, *=, /=, +=, -=, &&=, XX=, +, -, <, <=, >, >=, ==, !=, &&, XX,), ;, {, ,, }
para-list	->
proper-para-list	)
proper-para-list-tail	)
para-declaration	,,)
arg-list	;, ., ++, --, *, /, =, *=, /=, +=, -=, &&=, XX=, +, -, <, <=, >, >=, ==, !=, &&, XX,), {, ,, }
proper-arg-list	)
proper-arg-list-tail	)
arg	,,)
declaration-statement	const, class, var, func, }, $, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
function-definition	{, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var, func, }
return-type	{
variable-declaration	const, class, var, func, }, $, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
unpack-declaration	const, class, var, func, }, $, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
unpack-initial	;
unpack-decls	=, ,, }
unpack-decl-internal	}
unpack-decl-internal-tail	}
unpack-element	,, }
class-declaration	const, class, var, func, }, $, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
class-body	;
class-member	}
constant-declaration	const, class, var, func, }, $, {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
initial	;
anonymous-initial	;, ,, }
anonymous-initial-internal	}
anonymous-initial-internal-tail	}
anonymous-initial-element	,, }
type-annotation	;, ,,)
anonymous	-initial, ,, ;,), {
anonymous-internal	
anonymous-internal-tail	
anonymous-type	,
type	;, ,,), {
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	;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
8	statement → declaration-statement	const, class, var
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, (, INT-LITERAL, BOOL-LITERAL, const, class, var
15	continue-statement → continue ;	continue
16	return-statement → return expression ;	return
17	return-statement → return anonymous-initial ;	return
18	return-statement → return ;	return

19	expression-statement → expression-list ;	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, ;
20	expression-list → expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
21	expression-list → ε	;
22	expression → assignment-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
23	assignment-expression → condition-or-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
24	assignment-expression → u-expression assignment-operator condition-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
25	assignment-operator → =	=
26	assignment-operator → *=	*=
27	assignment-operator → /=	/=
28	assignment-operator → +=	+=
29	assignment-operator → -=	-=
30	assignment-operator → &&=	&&=
31	assignment-operator → XX=	XX=
32	condition-or-expression → condition-and-expression condition-or-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
33	condition-or-expression-tail → ε	), ;, {, ,, }
34	condition-or-expression-tail → XX condition-and-expression condition-or-expression-tail	XX
35	condition-and-expression → equality-expression condition-and-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
36	condition-and-expression-tail → && equality-expression equality-expression-tail	&&
37	condition-and-expression-tail → ε	XX,), ;, {, ,, }
38	equality-expression → rel-expression equality-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
39	equality-expression-tail → ε	==, !=, &&, XX,), ;, {, ,, }
40	equality-expression-tail → == rel-expression equality-expression-tail	==
41	equality-expression-tail → != rel-expression equality-expression-tail	!=
42	rel-expression → additive-expression rel-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
43	rel-expression-tail → ε	==, !=, &&, XX,), ;, {, ,, }
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	rel-expression-tail → >= additive-expression rel-expression-tail	>=
48	additive-expression → m-d-expression additive-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
49	additive-expression-tail → ε	<, <=, >, >=, ==, !=, &&, XX,), ;, {, ,, }
50	additive-expression-tail → + m-d-expression additive-expression-tail	+
51	additive-expression-tail → - m-d-expression additive-expression-tail	-
52	m-d-expression → u-expression m-d-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
53	m-d-expression-tail → ε	+, -, <, <=, >, >=, ==, !=, &&, XX,), ;, {, ,, }
54	m-d-expression-tail → * u-expression m-d-expression-tail	*
55	m-d-expression-tail → / u-expression m-d-expression-tail	/
56	u-expression → - u-expression	-
57	u-expression → ! u-expression	!
58	u-expression → ++ u-expression	++
59	u-expression → -- u-expression	--
60	u-expression → post-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
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-tail	++
65	post-expression-tail → -- post-expression-tail	--
66	post-expression-tail → ε	*, /, =, *=, /=, +=, -=, &&=, XX=, +, -, <, <=, >, >=, ==, !=, &&, 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-	const, identifier

	tail	
75	proper-para-list-tail → , para-declaration proper-para-list-tail	,
76	proper-para-list-tail → ε	)
77	para-declaration → const identifier type-annotation	const
78	para-declaration → identifier type-annotation	identifier
79	arg-list → ( )	(
80	arg-list → (proper-arg-list)	(
81	proper-arg-list → arg proper-arg-list-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, {
82	proper-arg-list-tail → , arg proper-arg-list-tail	,
83	proper-arg-list-tail → ε	)
84	arg → expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
85	arg → anonymous -initial	{
86	declaration-statement → constant-declaration	const
87	declaration-statement → variable-declaration	var
88	declaration-statement → class-declaration	class
89	declaration-statement → unpack-declaration	var
90	function-definition → func identifier para-list → return-type compound-statement	func
91	return-type → type	int, bool, identifier, {
92	return-type → void	void
93	variable-declaration → var identifier initial ;	var
94	variable-declaration → var identifier type-annotation ;	var
95	unpack-declaration → var unpack-decls = unpack-initial ;	var
96	unpack-initial → identifier	identifier
97	unpack-initial → identifier arg-list	identifier
98	unpack-decls → { unpack-decl-internal }	{
99	unpack-decl-internal → unpack-element unpack-decl-internal-tail	identifier, {
100	unpack-decl-internal-tail → , unpack-element unpack-decl-internal-tail	,
101	unpack-decl-internal-tail → ε	}
102	unpack-element → identifier	identifier
103	unpack-element → unpack-decls	{
104	class-declaration → class identifier class-body ;	class
105	class-body → { class-member }	{
106	class-member → declaration-statement class-member	const, class, var
107	class-member → function-definition class-member	func
108	class-member → ε	}
109	constant-declaration → const identifier init-expression ;	const
110	constant-declaration → const identifier type-annotation ;	const
111	initial → = expression	=
112	initial → = anonymous-initial	=
113	anonymous-initial → { anonymous-initial-internal }	{
114	anonymous-initial-internal → anonymous-initial-element anonymous-initial-internal-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, {
115	anonymous-initial-internal-tail → , anonymous-initial-element anonymous-initial-internal-tail	,
116	anonymous-initial-internal-tail → ε	}
117	anonymous-initial-element → expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
118	anonymous-initial-element → anonymous-initial	{
119	type-annotation → : type	:
120	anonymous → { anonymous-annotation-internal }	{
121	anonymous-internal → anonymous-type anonymous-internal-tail	int, bool, {
122	anonymous-internal-tail → , anonymous-type anonymous-internal-tail	,
123	anonymous-internal-tail → ε	
124	anonymous-type → int	int
125	anonymous-type → bool	bool
126	anonymous-type → anonymous	{
127	type → int	int
128	type → bool	bool
129	type → identifier	identifier
130	type → anonymous	{
131	top-level → statement top-level	{, while, continue, if, return, break, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, const, class, var
132	top-level → function-definition top-level	func
133	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","condition-expression","=","*=","/=","+=","-
=","&&=","XX=","XX","&&","==","!=","<","<=",">",">=","+","-","*","/","!","++","--",".","identifier","(",")","INT-LITERAL","BOOL-
LITERAL",",","const","-initial","func","->","void","var","class","init-expression",":","anonymous-annotation-
internal","int","bool","$"],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,2,135,3,4,7,1,134,5,6,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,7,135,135,7,7,7,135,7,7,135,7,7,135,8,135
[0,9,135,134,134,134,134,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135,1
[0,134,135,10,134,134,134,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135,
[0,134,135,134,11,134,134,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135,
[0,134,134,134,134,134,12,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135,
[0,14,135,14,14,14,14,13,14,14,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,14,135,135,14,14,14,135,14,14,135,1
[0,134,135,134,134,134,134,134,15,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135,
[0,134,135,134,134,134,134,134,134,18,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135,
[0,134,135,134,134,19,134,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,19,135,135,19,19,19,135,19,1
[0,135,135,135,135,21,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,20,135,135,20,20,20,135,20,2
[0,135,135,135,135,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,22,135,135,22,22,22,135,22,
[0,135,135,135,135,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,24,135,135,24,24,24,135,24,
[0,135,135,135,135,135,135,135,135,135,134,25,26,27,28,29,30,31,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,13
[0,135,135,135,135,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,32,135,135,32,32,32,135,32,
[0,135,135,135,135,33,33,33,135,135,135,135,135,135,135,135,135,135,34,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,134,134,134,135,135,135,135,135,135,135,135,135,135,135,134,135,135,135,135,135,135,135,35,135,135,35,35,35,135,35,
[0,135,135,135,135,37,37,37,135,135,135,135,135,135,135,135,135,135,37,36,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,
[0,135,135,135,135,134,134,134,135,135,135,135,135,135,135,135,135,135,134,134,134,134,135,135,135,135,135,38,135,135,38,38,38,135,38,
[0,135,135,135,135,39,39,39,135,135,135,135,135,135,135,135,135,135,39,39,40,41,135,135,135,135,135,135,135,135,135,135,135,135,135,13
[0,135,135,135,135,134,134,134,135,135,135,135,135,135,135,135,135,135,134,134,134,134,135,135,135,135,135,42,135,135,42,42,42,135,42,
[0,135,135,135,135,43,43,43,135,135,135,135,135,135,135,135,135,135,43,43,43,43,44,45,46,47,135,135,135,135,135,135,135,135,135,135,43
[0,135,135,135,135,134,134,134,135,135,135,135,135,135,135,135,135,135,134,134,134,134,134,134,134,134,135,48,135,135,48,48,48,135,48,
[0,135,135,135,135,49,49,49,135,135,135,135,135,135,135,135,135,135,49,49,49,49,49,49,49,49,50,51,135,135,135,135,135,135,135,135,49,1
[0,135,135,135,135,134,134,134,135,135,135,135,135,135,135,135,135,135,134,134,134,134,134,134,134,134,134,52,135,135,52,52,52,135,52,
[0,135,135,135,135,53,53,53,135,135,135,135,135,135,135,135,135,135,53,53,53,53,53,53,53,53,53,53,54,55,135,135,135,135,135,135,53,135
[0,135,135,135,135,134,134,134,135,135,135,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,56,134,134,57,58,59,135,60,
[0,135,135,135,135,134,134,134,135,135,135,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,135,135,135,135
[0,135,135,135,135,66,66,66,135,135,135,66,66,66,66,66,66,66,66,66,66,66,66,66,66,66,66,66,66,66,135,64,65,63,135,135,66,135,135,66,13
[0,135,135,135,135,134,134,134,135,135,135,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,135,134,134,134
[0,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,134,134,134,135,135,135,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,134,135,134,134,134
[0,135,135,135,135,135,81,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,81,135,135,81,81,81,135,81,8
[0,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,135,85,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,84,135,135,84,84,84,135,84,8
[0,134,135,134,134,134,134,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135
[0,134,135,134,134,134,134,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135
[0,135,135,135,135,135,91,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,
[0,134,135,134,134,134,134,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135
[0,134,135,134,134,134,134,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135
[0,135,135,135,135,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,135,98,134,135,135,135,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,
[0,135,135,135,135,135,99,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,
[0,135,135,135,135,135,135,101,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,135,103,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,134,135,134,134,134,134,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135
[0,135,135,135,135,134,105,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,135,135,108,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,134,135,134,134,134,134,134,134,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,134,135,135,134,134,134,135
[0,135,135,135,135,134,135,135,135,135,112,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,134,113,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,135,114,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,114,135,135,114,114,114,135
[0,135,135,135,135,135,135,116,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,135,118,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,117,135,135,117,117,117,135
[0,135,135,135,135,134,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,134,120,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,135,121,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135

[0,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,135,126,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,135,135,135,135,134,130,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135
[0,131,135,131,131,131,131,135,131,131,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,135,131,135,135,131,131,131,135

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":[38],"9":[5,-2,5,11,-1],"10":[5,11,-3],"11":[-5,-4],"12":[-7,6,-6],"14":[6,1],"15":[-5,-8],"16":[-5,11,-9],"17":[-5,53,-9],"18":[-5,-9],"19":[-5,10],"20":[11],"22":[12],"23":[14],"24":[-10,13,26],"25":[-11],"26":[-12],"27":[-13],"28":[-14],"29":[-15],"30":[-16],"31":[-17],"32":[15,16],"34":[15,16,-18],"35":[17,18],"36":[19,18,-19],"38":[19,20],"40":[19,20,-20],"41":[19,20,-21],"42":[21,22],"44":[21,22,-22],"45":[21,22,-23],"46":[21,22,-24],"47":[21,22,-25],"48":[23,24],"50":[23,24,-26],"51":[23,24,-27],"52":[25,26],"54":[25,26,-28],"55":[25,26,-29],"56":[26,-27],"57":[26,-30],"58":[26,-31],"59":[26,-32],"60":[27],"61":[29],"62":[28,29],"63":[28,-34,-33],"64":[28,-31],"65":[28,-32],"67":[-34],"68":[34,-34],"69":[-36,11,-35],"70":[-37],"71":[-38],"72":[-36,-35],"73":[-36,31,-35],"74":[32,33],"75":[32,33,-39],"77":[57,-34,-40],"78":[57,-34],"79":[-36,-35],"80":[-36,35,-35],"81":[36,37],"82":[36,37,-39],"84":[11],"85":[-41,58],"86":[51],"87":[41],"88":[48],"89":[42],"90":[30,-34,-42],"91":[62],"92":[-44],"93":[-5,52,-34,-45],"94":[-5,57,-34,-45],"95":[-5,43,-11,44,-45],"96":[-34],"97":[34,-34],"98":[-7,45,-6],"99":[46,47],"100":[46,47,-39],"102":[-34],"103":[44],"104":[-5,49,-34,-46],"105":[-7,50,-6],"106":[50,38],"107":[50,39],"109":[-5,-47,-34,-40],"110":[-5,57,-34,-40],"111":[11,-11],"112":[53,-11],"113":[-7,54,-6],"114":[55,56],"115":[55,56,-39],"117":[11],"118":[53],"119":[62,-48],"120":[-7,-49,-6],"121":[60,61],"122":[60,61,-39],"124":[-50],"125":[-51],"126":[58],"127":[-50],"128":[-51],"129":[-34],"130":[58],"131":[63,1],"132":[63,39]}

How to Calculate First, Follow, & Predict Sets

Specify your grammar in EBNF and slam the button. That's it.

EBNF Grammar Specification Requirements

Productions use the following format:

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

- Symbols are inferred as terminal by absence from the left hand side of production rules.
- "->" designates definition, "|" designates alternation, and newlines designate termination.
- x -> y | z is EBNF short-hand for
  x -> y
  x -> z
- Use "EPSILON" to represent ε or "LAMBDA" for λ productions. (The two function identically.) E.g., A -> b | EPSILON.
- 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 compiler design or parsing.

Purpose

Automatic generation of first sets, follow sets, and predict sets speeds up the process of writing parsers. Generating these sets by hands is tedious; this tool helps ameliorate that. Goals:

- Tight feedback loops for faster learning.
- Convenient experimentation with language tweaks. (Write a generic, table/dictionary-driven parser and just plug in the JSON output to get off the ground quickly.)
- Help with tackling existing coursework or creating new course material.

Underlying Theory

I'll do a write-up on this soon. In the interim, you can read about:

- how to determine first and follow sets (PDF from Programming Languages course at University of Alaska Fairbanks)
- significance of first and follow sets in top-down (LL(1)) parsing.
- follow sets' involvement in bottom-up parsing (LALR, in this case)