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
func	func
var	var
class	class
:	:
int	int
bool	bool
if-statement	if
while-statement	while
break-statement	break
compound-statement	{
statement-list	ε, {, while, continue, if, return, break, const, class, var, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
continue-statement	continue
return-statement	return
expression-statement	;, ε, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
expression-list	ε, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
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	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
post-expression-tail	., ++
primary-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
para-list	(
proper-para-list-tail	,, ε
para-declaration	const, int, bool
arg-list	(
proper-arg-list-tail	,, ε
function-definition	func
variable-declaration	var
class-declaration	class
constant-declaration	const
init-expression	=
type-annotation	:
type	int, bool
top-level	ε, {, while, continue, if, return, break, const, class, var, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, func
statement	{, while, continue, if, return, break, const, class, var, ;, ε, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL

post-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
declaration-statement	const, class, var
proper-para-list	const, 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
proper-arg-list	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL

Follow Set

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

function-definition	{, while, continue, if, return, break, const, class, var, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, func
variable-declaration	var, $, {, while, continue, if, return, break, const, class, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, func, }
class-declaration	$, {, while, continue, if, return, break, const, class, var, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, func, }
constant-declaration	$, {, while, continue, if, return, break, const, class, var, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, func, }
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	;, -, !, ++, --, 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, const, class, var, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
15	continue-statement → continue ;	continue
16	return-statement → return expression ;	return
17	return-statement → return ;	return
18	expression-statement → expression-list ;	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, ;
19	expression-list → expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
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	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
25	assignment-expression → condition-or-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
26	assignment-expression → u-expression assignment-operator condition-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
27	assignment-operator → =	=
28	assignment-operator → *=	*=
29	assignment-operator → /=	/=
30	assignment-operator → +=	+=
31	assignment-operator → -=	-=
32	assignment-operator → &&=	&&=
33	assignment-operator → XX=	XX=
34	condition-or-expression → condition-and-expression condition-or-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
35	condition-or-expression-tail → ε	), ;, {, ,
36	condition-or-expression-tail → XX condition-and-expression condition-or-expression-tail	XX
37	condition-and-expression → equality-expression condition-and-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
38	condition-and-expression-tail → && equality-expression equality-expression-tail	&&
39	condition-and-expression-tail → ε	XX,), ;, {, ,
40	equality-expression → rel-expression equality-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
41	equality-expression-tail → ε	==, !=, &&, XX,), ;, {, ,
42	equality-expression-tail → == rel-expression equality-expression-tail	==
43	equality-expression-tail → != rel-expression equality-expression-tail	!=
44	rel-expression → additive-expression rel-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL

45	rel-expression-tail → ε	==, !=, &&, XX,), ;, {, ,
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	rel-expression-tail → >= additive-expression rel-expression-tail	>=
50	additive-expression → m-d-expression additive-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
51	additive-expression-tail → ε	<, <=, >, >=, ==, !=, &&, XX,), ;, {, ,
52	additive-expression-tail → + m-d-expression additive-expression-tail	+
53	additive-expression-tail → - m-d-expression additive-expression-tail	_
54	m-d-expression → u-expression m-d-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
55	m-d-expression-tail → ε	+, -, <, <=, >, >=, ==, !=, &&, XX,), ;, {, ,
56	m-d-expression-tail → * u-expression m-d-expression-tail	*
57	m-d-expression-tail → / u-expression m-d-expression-tail	/
58	u-expression → - u-expression	-
59	u-expression → ! u-expression	!
60	u-expression → ++ u-expression	++
61	u-expression → -- u-expression	--
62	u-expression → post-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
63	post-expression → primary-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
64	post-expression → primary-expression post-expression-tail	identifier, (, INT-LITERAL, BOOL-LITERAL
65	post-expression-tail → . identifier post-expression-tail	.
66	post-expression-tail → ++ post-expression-tail	++
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	const, 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	para-declaration → const type identifier	const
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	declaration-statement → constant-declaration	const
86	declaration-statement → variable-declaration	var
87	declaration-statement → class-declaration	class
88	function-definition → func identifier para-list compound-statement	func
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, const, class, var, ;, -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL

100 top-level → function-definition top-level func
101 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","func","var","class",":","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],
[0,2,103,3,4,7,1,102,5,6,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,7,103,103,7,7,7,103,7,7,103,7,7,103,8,10
[0,9,103,102,102,102,102,102,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,102,103,103,102,102,102,103,1
[0,102,103,10,102,102,102,102,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,102,103,103,102,102,102,103,
[0,102,103,102,11,102,102,102,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,102,103,103,102,102,102,103,
[0,102,102,102,102,102,12,102,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,102,103,103,102,102,102,103,
[0,14,103,14,14,14,14,13,14,14,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,14,103,103,14,14,14,103,14,14,103,1
[0,102,103,102,102,102,102,102,15,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,102,103,103,102,102,102,103,
[0,102,103,102,102,102,102,102,102,17,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,102,103,103,102,102,102,103,
[0,102,103,102,102,18,102,102,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,18,103,103,18,18,18,103,18,1
[0,103,103,103,103,20,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,19,103,103,19,19,19,103,19,1
[0,103,103,103,103,103,21,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,
[0,103,103,103,103,103,103,23,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,
[0,103,103,103,103,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,24,103,103,24,24,24,103,24,
[0,103,103,103,103,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,26,103,103,26,26,26,103,26,
[0,103,103,103,103,103,103,103,103,103,102,27,28,29,30,31,32,33,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,10
[0,103,103,103,103,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,34,103,103,34,34,34,103,34,
[0,103,103,103,103,35,35,103,103,103,103,103,103,103,103,103,103,36,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,10
[0,103,103,103,103,102,102,103,103,103,103,103,103,103,103,103,103,103,102,103,103,103,103,103,103,103,103,37,103,103,37,37,37,103,37,
[0,103,103,103,103,39,39,103,103,103,103,103,103,103,103,103,103,103,39,38,103,103,103,103,103,103,103,103,103,103,103,103,103,103,10
[0,103,103,103,103,102,102,103,103,103,103,103,103,103,103,103,103,103,102,102,102,102,103,103,103,103,103,40,103,103,40,40,40,103,40,
[0,103,103,103,103,41,41,103,103,103,103,103,103,103,103,103,103,103,41,41,42,43,103,103,103,103,103,103,103,103,103,103,103,103,103,1
[0,103,103,103,103,102,102,103,103,103,103,103,103,103,103,103,103,103,102,102,102,102,103,103,103,103,103,44,103,103,44,44,44,103,44,
[0,103,103,103,103,45,45,103,103,103,103,103,103,103,103,103,103,103,45,45,45,45,46,47,48,49,103,103,103,103,103,103,103,103,103,103,4
[0,103,103,103,103,102,102,103,103,103,103,103,103,103,103,103,103,103,102,102,102,102,102,102,102,102,103,50,103,103,50,50,50,103,50,
[0,103,103,103,103,51,51,103,103,103,103,103,103,103,103,103,103,103,51,51,51,51,51,51,51,51,52,53,103,103,103,103,103,103,103,103,51,
[0,103,103,103,103,102,102,103,103,103,103,103,103,103,103,103,103,103,102,102,102,102,102,102,102,102,102,54,103,103,54,54,54,103,54,
[0,103,103,103,103,55,55,103,103,103,103,103,103,103,103,103,103,103,55,55,55,55,55,55,55,55,55,55,56,57,103,103,103,103,103,103,55,10
[0,103,103,103,103,102,102,103,103,103,103,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,58,102,102,59,60,61,103,62,
[0,103,103,103,103,102,102,103,103,103,103,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,103,103,103,10
[0,103,103,103,103,102,102,103,103,103,103,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,103,66,103,65,1
[0,103,103,103,103,102,102,103,103,103,103,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,103,102,103,10
[0,103,103,103,103,103,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,10
[0,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,10
[0,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,10
[0,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,10
[0,103,103,103,103,102,102,103,103,103,103,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,103,102,103,10
[0,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,81,103,103,81,81,81,103,81,
[0,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,10
[0,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,84,103,103,84,84,84,103,84,
[0,102,103,102,102,102,102,102,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,102,103,103,102,102,102,10
[0,102,103,102,102,102,102,103,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,102,103,103,102,102,102,10
[0,102,103,102,102,102,102,102,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,102,103,103,102,102,102,10
[0,102,103,102,102,102,102,102,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,102,103,103,102,102,102,10
[0,102,103,102,102,102,102,102,102,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,102,103,103,102,102,102,10
[0,103,103,103,103,102,103,103,103,103,95,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,
[0,103,103,103,103,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,10
[0,103,103,103,103,102,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,10
[0,99,103,99,99,99,99,103,99,99,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,103,99,103,103,99,99,99,103,99,99,103,

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":[-10,15,28],"27":[-11],"28":[-12],"29":[-13],"30":[-14],"31":[-15],"32":[-16],"33":[-17],"34":[17,18],"36":[17,18,-18],"37":[19,20],"38":[21,20,-19],"40":[21,22],"42":[21,22,-20],"43":[21,22,-21],"44":[23,24],"46":[23,24,-22],"47":[23,24,-23],"48":[23,24,-24],"49":[23,24,-25],"50":[25,26],"52":[25,26,-26],"53":[25,26,-27],"54":[27,28],"56":[27,28,-28],"57":[27,28,-29],"58":[28,-27],"59":[28,-30],"60":[28,-31],"61":[28,-32],"62":[29],"63":[31],"64":[30,31],"65":[30,-34,-33],"66":[30,-31],"67":[-34],"68":[36,-34],"69":[-36,13,-35],"70":[-37],"71":[-38],"72":[-36,-35],"73":[-36,33,-35],"74":[34,35],"75":[34,35,-39],"77":[-34,47],"78":[-34,47,-40],"79":[-36,-35],"80":[-36,37,-35],"81":[38,39],"82":[38,39,-39],"84":[13],"85":[44],"86":[42],"87":[43],"88":[5,32,-34,-41],"89":[-5,45,-34,-42],"90":[-5,46,-34,-42],"91":[-5,45,-34,-43],"92":[-5,46,-34,-43],"93":[-5,45,-34,-40],"94":[-5,46,-34,-40],"95":[13,-11],"96":[47,-44],"97":[-45],"98":[-46],"99":[48,1],"100":[48,41]}

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)