

[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](#)

# 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
,	,
-initial	-initial
func	func
->	->
void	void
var	var
class	class
const	const
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	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    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 $\rightarrow$ expression-list ;	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, ;
20	expression-list $\rightarrow$ expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
21	expression-list $\rightarrow$ $\epsilon$	;
22	expression $\rightarrow$ assignment-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
23	assignment-expression $\rightarrow$ condition-or-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
24	assignment-expression $\rightarrow$ u-expression assignment-operator condition-expression	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
25	assignment-operator $\rightarrow$ =	=
26	assignment-operator $\rightarrow$ *=	*=
27	assignment-operator $\rightarrow$ /=	/=
28	assignment-operator $\rightarrow$ +=	+=
29	assignment-operator $\rightarrow$ -=	-=
30	assignment-operator $\rightarrow$ &&=	&&=
31	assignment-operator $\rightarrow$ XX=	XX=
32	condition-or-expression $\rightarrow$ condition-and-expression condition-or-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
33	condition-or-expression-tail $\rightarrow$ $\epsilon$	), :, {, ,, }
34	condition-or-expression-tail $\rightarrow$ XX condition-and- expression condition-or-expression-tail	XX
35	condition-and-expression $\rightarrow$ equality-expression condition-and-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
36	condition-and-expression-tail $\rightarrow$ && equality-expression equality-expression-tail	&&
37	condition-and-expression-tail $\rightarrow$ $\epsilon$	XX, ), :, {, ,, }
38	equality-expression $\rightarrow$ rel-expression equality- expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
39	equality-expression-tail $\rightarrow$ $\epsilon$	==, !=, &&, XX, ), :, {, ,, }
40	equality-expression-tail $\rightarrow$ == rel-expression equality- expression-tail	==
41	equality-expression-tail $\rightarrow$ != rel-expression equality- expression-tail	!=
42	rel-expression $\rightarrow$ additive-expression rel-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
43	rel-expression-tail $\rightarrow$ $\epsilon$	==, !=, &&, XX, ), :, {, ,, }
44	rel-expression-tail $\rightarrow$ < additive-expression rel- expression-tail	<
45	rel-expression-tail $\rightarrow$ <= additive-expression rel- expression-tail	<=
46	rel-expression-tail $\rightarrow$ > additive-expression rel- expression-tail	>
47	rel-expression-tail $\rightarrow$ >= additive-expression rel- expression-tail	>=
48	additive-expression $\rightarrow$ m-d-expression additive- expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
49	additive-expression-tail $\rightarrow$ $\epsilon$	<, <=, >, >=, ==, !=, &&, XX, ), :, {, ,, }
50	additive-expression-tail $\rightarrow$ + m-d-expression additive- expression-tail	+
51	additive-expression-tail $\rightarrow$ - m-d-expression additive- expression-tail	-
52	m-d-expression $\rightarrow$ u-expression m-d-expression-tail	-, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
53	m-d-expression-tail $\rightarrow$ $\epsilon$	+, -, <, <=, >, >=, ==, !=, &&, XX, ), :, {, ,, }
54	m-d-expression-tail $\rightarrow$ * u-expression m-d-expression-tail *	
55	m-d-expression-tail $\rightarrow$ / u-expression m-d-expression-tail /	
56	u-expression $\rightarrow$ - u-expression	-
57	u-expression $\rightarrow$ ! u-expression	!
58	u-expression $\rightarrow$ ++ u-expression	++
59	u-expression $\rightarrow$ -- u-expression	--
60	u-expression $\rightarrow$ post-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
61	post-expression $\rightarrow$ primary-expression	identifier, (, INT-LITERAL, BOOL-LITERAL
62	post-expression $\rightarrow$ primary-expression post-expression- tail	identifier, (, INT-LITERAL, BOOL-LITERAL
63	post-expression-tail $\rightarrow$ . identifier post-expression-tail .	
64	post-expression-tail $\rightarrow$ ++ post-expression-tail	++
65	post-expression-tail $\rightarrow$ -- post-expression-tail	--
66	post-expression-tail $\rightarrow$ $\epsilon$	*, /, =, *=, /=, +=, -=, &&=, XX=, +, -, <, <=, >, >=, ==, !=, &&, XX, ) , :, {, ,, }
67	primary-expression $\rightarrow$ identifier	identifier
68	primary-expression $\rightarrow$ identifier arg-list	identifier
69	primary-expression $\rightarrow$ ( expression )	(
70	primary-expression $\rightarrow$ INT-LITERAL	INT-LITERAL
71	primary-expression $\rightarrow$ BOOL-LITERAL	BOOL-LITERAL
72	para-list $\rightarrow$ ( )	(
73	para-list $\rightarrow$ ( proper-para-list )	(
74	proper-para-list $\rightarrow$ para-declaration proper-para-list-	identifier

```

tail
75 proper-para-list-tail → , para-declaration proper-para-
   list-tail
76 proper-para-list-tail → ε
77 para-declaration → identifier type-annotation identifier
78 arg-list → ( )
79 arg-list → ( proper-arg-list )
80 proper-arg-list → arg proper-arg-list-tail -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, {
81 proper-arg-list-tail → , arg proper-arg-list-tail
82 proper-arg-list-tail → ε
83 arg → expression -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
84 arg → anonymous -initial {
85 declaration-statement → constant-declaration const
86 declaration-statement → variable-declaration var
87 declaration-statement → class-declaration class
88 declaration-statement → unpack-declaration var
89 function-definition → func identifier para-list → func
   return-type compound-statement
90 return-type → type int, bool, identifier, {
91 return-type → void
92 variable-declaration → var identifier initial ; var
93 variable-declaration → var identifier type-annotation ; var
94 unpack-declaration → var unpack-decls = unpack-initial ; var
95 unpack-initial → identifier identifier
96 unpack-initial → identifier arg-list identifier
97 unpack-decls → { unpack-decl-internal } {
98 unpack-decl-internal → unpack-element unpack-decl-
   internal-tail identifier, {
99 unpack-decl-internal-tail → , unpack-element unpack-
   decl-internal-tail
100 unpack-decl-internal-tail → ε
101 unpack-element → identifier identifier
102 unpack-element → unpack-decls {
103 class-declaration → class identifier class-body ; class
104 class-body → { class-member } {
105 class-member → declaration-statement class-member const, class, var
106 class-member → function-definition class-member func
107 class-member → ε }
108 constant-declaration → const identifier init-expression const
   ;
109 constant-declaration → const identifier type-annotation const
   ;
110 initial → = expression =
111 initial → = anonymous-initial =
112 anonymous-initial → { anonymous-initial-internal } {
113 anonymous-initial-internal → anonymous-initial-element -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL, {
   anonymous-initial-internal-tail
114 anonymous-initial-internal-tail → , anonymous-initial-
   element anonymous-initial-internal-tail
115 anonymous-initial-internal-tail → ε
116 anonymous-initial-element → expression -, !, ++, --, identifier, (, INT-LITERAL, BOOL-LITERAL
117 anonymous-initial-element → anonymous-initial {
118 type-annotation → : type :
119 anonymous → { anonymous-annotation-internal } {
120 anonymous-internal → anonymous-type anonymous-internal-
   tail int, bool, {
121 anonymous-internal-tail → , anonymous-type anonymous-
   internal-tail
122 anonymous-internal-tail → ε
123 anonymous-type → int int
124 anonymous-type → bool bool
125 anonymous-type → anonymous {
126 type → int int
127 type → bool bool
128 type → identifier identifier
129 type → anonymous {
130 top-level → statement top-level {, while, continue, if, return, break, ;, -, !, ++, --, identifier, (,
   INT-LITERAL, BOOL-LITERAL, const, class, var
131 top-level → function-definition top-level func
132 top-level → ε

```

## LL(1) Parsing Table

- 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.

SCAN error code = # of predict table productions + 2

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

<http://hackincoff.com/compilers/predict-first-follow-set> 7/8

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

[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], "78":[-36,-35], "79":[-36,35,-35], "80":[36,37], "81":[36,37,-39], "83":[11], "84":[-40,58], "85":[51], "86":  
[41], "87":[44], "88":[42], "89":[30,-34,-41], "90":[62], "91":[-43], "92":[-5,52,-34,-44], "93":[-5,57,-34,-44], "94":  
[-5,43,-11,44,-44], "95":[-34], "96":[34,-34], "97":[-7,45,-6], "98":[46,47], "99":[46,47,-39], "101":[-34], "102":[44], "103":  
[-5,49,-34,-45], "104":[-7,50,-6], "105":[50,38], "106":[50,39], "108":[-5,-47,-34,-46], "109":[-5,57,-34,-46], "110":[11,-11], "111":  
[53,-11], "112":[-7,54,-6], "113":[55,56], "114":[55,56,-39], "116":[11], "117":[53], "118":[62,-48], "119":[-7,-49,-6], "120":  
[60,61], "121":[60,61,-39], "123":[-50], "124":[-51], "125":[58], "126":[-50], "127":[-51], "128":[-34], "129":[58], "130":[63,1], "131":  
[63,39]]

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

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

Productions use the following format:

- Symbols are inferred as terminal by absence from the left hand side of production rules.
- “ $\rightarrow$ ” designates definition, “ $|$ ” designates alternation, and newlines designate termination.
- $x \rightarrow y \mid z$  is EBNF short-hand for
 
$$\begin{array}{l} x \rightarrow y \\ x \rightarrow z \end{array}$$
- Use “EPSILON” to represent  $\epsilon$  or “LAMBDA” for  $\lambda$  productions. (The two function identically.) E.g.,  $A \rightarrow b \mid \text{EPSILON}$ .
- Be certain to place spaces between things you don't want read as one symbol.  $(A) \neq (A)$

### 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 \(LALR, in this case\)](#)