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 **L**eft-to-right, using **L**eftmost-derivations).

Follow sets are used in top-down parsers, but also in LR parsers (bottom-up parsers, reading **L**eft-to-right, using **R**ightmost 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. →

```
Prog -> LAMBDA |
Func Prog
Func -> TipoRet
ident ( ListaParam )
Corpo
TipoRet -> void |
Tipo
ListaParam -> LAMBDA
| Param OpcParams
OpcParams -> LAMBDA
| , Param OpcParams
Param -> Tipo ident
Corpo -> {
ListaDeclara
ListaComando }
ListaDeclara ->
LAMBDA | Declara
ListaDeclara
ListaComando ->
LAMBDA | Comando
```

Click for Predict, First, and Follow Sets

First Set

Non-Terminal Symbol	First Set
λ	λ
ident	ident
(	(
)	)
void	void
,	,
{	{
}	}
[	[
]	]
int	int
float	float
string	string
;	;
=	=
if	if
else	else
elif	elif
for	for
foreach	foreach
:	:
while	while
return	return
continue	continue
break	break
read	read
write	write
valint	valint
valfloat	valfloat
valstring	valstring
lambda	lambda
+	+
-	-
*	*
/	/
mod	mod
div	div
==	==
!=	!=
<=	<=
>=	>=
>	>
<	<
not	not
and	and
or	or
Prog	λ, void, [, int, float, string

TipoRet	void, [, int, float, string
ListaParam	λ, [, int, float, string
OpcParams	λ, ,
Corpo	{
ListaDeclara	λ, [, int, float, string
ListaComando	λ, ident, for, foreach, return, break, write, if, continue, {, while, read
Tipo	[, int, float, string
Primitivo	int, float, string
RestoListaVar	λ, ,
Var	ident
OpcValor	λ, =
ComAtrib	ident
PosicaoOpc	λ, [
ComIf	if
OpcElse	λ, else, elif
ComFor	for, foreach
ComWhile	while
ComReturn	return
OpcRet	λ, not, +, -, ident, (, valint, valfloat, valstring, [
ComContinue	continue
ComBreak	break
ComEntrada	read
ComSaida	write
RestoListaOut	λ, ,
ComBloco	{
Folha	ident, (, valint, valfloat, valstring, [
ValPrim	valint, valfloat, valstring
ValLista	[
Recorte	λ, [
Dentro	:, not, +, -, ident, (, valint, valfloat, valstring, [
RestoDentro	λ, :
OpcInt	λ, not, +, -, ident, (, valint, valfloat, valstring, [
ListaArgs	λ, not, +, -, ident, (, valint, valfloat, valstring, [
RestoListaArgs	lambda, ,
ListaExp	λ, not, +, -, ident, (, valint, valfloat, valstring, [
OpcListaExp	λ, ,
Uno	+, -, ident, (, valint, valfloat, valstring, [
RestoMult	λ, *, /, mod, div
RestoSoma	λ, +, -
RestoRel	λ, ==, !=, <=, >=, >, <
Nao	not, +, -, ident, (, valint, valfloat, valstring, [
RestoJunc	λ, and
RestoExp	λ, or
ListaVar	ident
Comando	ident, for, foreach, return, break, write, if, continue, {, while, read
Param	[, int, float, string
Declara	[, int, float, string
Func	void, [, int, float, string
Mult	+, -, ident, (, valint, valfloat, valstring, [
Soma	+, -, ident, (, valint, valfloat, valstring, [
Rel	+, -, ident, (, valint, valfloat, valstring, [
Junc	not, +, -, ident, (, valint, valfloat, valstring, [
Exp	not, +, -, ident, (, valint, valfloat, valstring, [
ListaOut	not, +, -, ident, (, valint, valfloat, valstring, [

Follow Set

Non-Terminal Symbol	Follow Set
Prog	$
Func	void, [, int, float, string, $
TipoRet	ident
ListaParam	)
OpcParams	)
Param	,,)
Corpo	void, [, int, float, string, $
ListaDeclara	ident, for, foreach, return, break, write, if, continue, {, while, read
ListaComando	}
Tipo	ident
Primitivo	], ident
Declara	[, int, float, string, ident, for, foreach, return, break, write, if, continue, {, while, read
ListaVar	;
RestoListaVar	;
Var	,, ;
OpcValor	,, ;
Comando	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
ComAtrib	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
PosicaoOpc	=
ComIf	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
OpcElse	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
ComFor	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
ComWhile	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
ComReturn	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
OpcRet	;
ComContinue	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
ComBreak	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
ComEntrada	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
ComSaida	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
ListaOut	)
RestoListaOut	)
ComBloco	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
Folha	*, /, mod, div, +, -, ==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,]
ValPrim	*, /, mod, div, +, -, ==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,]

ValLista	*, /, mod, div, +, -, ==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,]
Recorte	*, /, mod, div, +, -, ==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,]
Dentro	]
RestoDentro	]
OpcInt	]
ListaArgs	)
RestoListaArgs	)
ListaExp	]
OpcListaExp	]
Uno	*, /, mod, div, +, -, ==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,]
Mult	+, -, ==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,]
RestoMult	+, -, ==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,]
Soma	==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,]
RestoSoma	==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,]
Rel	and, or, ,, lambda, :,), ;,]
RestoRel	and, or, ,, lambda, :,), ;,]
Nao	and, or, ,, lambda, :,), ;,]
Junc	or, ,, lambda, :,), ;,]
RestoJunc	or, ,, lambda, :,), ;,]
Exp	,, lambda, :,), ;,]
RestoExp	,, lambda, :,), ;,]

Predict Set

#	Expression	Predict
1	Prog → λ	$
2	Prog → Func Prog	void, , int, float, string
3	Func → TipoRet ident (ListaParam) Corpo	void, , int, float, string
4	TipoRet → void	void
5	TipoRet → Tipo	, int, float, string
6	ListaParam → λ	)
7	ListaParam → Param OpcParams	, int, float, string
8	OpcParams → λ	)
9	OpcParams → , Param OpcParams	,
10	Param → Tipo ident	, int, float, string
11	Corpo → { ListaDeclara ListaComando }	{
12	ListaDeclara → λ	ident, for, foreach, return, break, write, if, continue, {, while, read
13	ListaDeclara → Declara ListaDeclara	, int, float, string
14	ListaComando → λ	}
15	ListaComando → Comando ListaComando	ident, for, foreach, return, break, write, if, continue, {, while, read
16	Tipo → Primitivo	int, float, string
17	Tipo → [Primitivo]	
18	Primitivo → int	int
19	Primitivo → float	float
20	Primitivo → string	string
21	Declara → Tipo ListaVar ;	, int, float, string
22	ListaVar → Var RestoListaVar	ident
23	RestoListaVar → λ	;
24	RestoListaVar → , ListaVar	,
25	Var → ident OpcValor	ident
26	OpcValor → λ	,, ;
27	OpcValor → = Exp	=
28	Comando → ComAtrib	ident
29	Comando → ComIf	if
30	Comando → ComFor	for, foreach
31	Comando → ComWhile	while
32	Comando → ComReturn	return
33	Comando → ComContinue	continue
34	Comando → ComBreak	break
35	Comando → ComEntrada	read
36	Comando → ComSaida	write
37	Comando → ComBloco	{
38	ComAtrib → ident PosicaoOpc = Exp ;	ident
39	PosicaoOpc → λ	=
40	PosicaoOpc → [Exp]	
41	ComIf → if (Exp) Comando OpcElse	if
42	OpcElse → λ	else, elif, ident, for, foreach, return, break, write, if, continue, {, while, read, }
43	OpcElse → else Comando	else
44	OpcElse → elif (Exp) Comando OpcElse	elif
45	ComFor → for (ident = Exp ; Exp ; ident = Exp) Comando	for
46	ComFor → foreach ident = Exp : Comando	foreach
47	ComWhile → while (Exp) Comando	while
48	ComReturn → return OpcRet ;	return
49	OpcRet → λ	;
50	OpcRet → Exp	not, +, -, ident, (, valint, valfloat, valstring,
51	ComContinue → continue ;	continue
52	ComBreak → break ;	break
53	ComEntrada → read (ident) ;	read
54	ComSaida → write (ListaOut) ;	write
55	ListaOut → Exp RestoListaOut	not, +, -, ident, (, valint, valfloat, valstring,
56	RestoListaOut → λ	)
57	RestoListaOut → , ListaOut	,
58	ComBloco → { ListaComando }	{
59	Folha → ValPrim	valint, valfloat, valstring
60	Folha → ident Recorte	ident
61	Folha → ident (ListaArgs)	ident
62	Folha → (Exp)	(
63	Folha → ValLista	
64	ValPrim → valint	valint
65	ValPrim → valfloat	valfloat
66	ValPrim → valstring	valstring
67	ValLista → [ListaExp]	

68	Recorte → λ	*, /, mod, div, +, -, ==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,
69	Recorte → [Dentro]	
70	Dentro → Exp RestoDentro	not, +, -, ident, (, valint, valfloat, valstring,
71	Dentro → : OpcInt	:
72	RestoDentro → λ	
73	RestoDentro → : OpcInt	:
74	OpcInt → λ	
75	OpcInt → Exp	not, +, -, ident, (, valint, valfloat, valstring,
76	ListaArgs → λ	)
77	ListaArgs → Exp RestoListaArgs	not, +, -, ident, (, valint, valfloat, valstring,
78	RestoListaArgs → lambda	lambda
79	RestoListaArgs → , Exp RestoListaArgs	,
80	ListaExp → λ	
81	ListaExp → Exp OpcListaExp	not, +, -, ident, (, valint, valfloat, valstring,
82	OpcListaExp → λ	
83	OpcListaExp → , Exp OpcListaExp	,
84	Uno → + Uno	+
85	Uno → - Uno	-
86	Uno → Folha	ident, (, valint, valfloat, valstring,
87	Mult → Uno RestoMult	+, -, ident, (, valint, valfloat, valstring,
88	RestoMult → λ	+, -, ==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,
89	RestoMult → * Uno RestoMult	*
90	RestoMult → / Uno RestoMult	/
91	RestoMult → mod Uno RestoMult	mod
92	RestoMult → div Uno RestoMult	div
93	Soma → Mult RestoSoma	+, -, ident, (, valint, valfloat, valstring,
94	RestoSoma → λ	==, !=, <=, >=, >, <, and, or, ,, lambda, :,), ;,
95	RestoSoma → + Mult RestoSoma	+
96	RestoSoma → - Mult RestoSoma	-
97	Rel → Soma RestoRel	+, -, ident, (, valint, valfloat, valstring,
98	RestoRel → λ	and, or, ,, lambda, :,), ;,
99	RestoRel → == Soma	==
100	RestoRel → != Soma	!=
101	RestoRel → <= Soma	<=
102	RestoRel → >= Soma	>=
103	RestoRel → > Soma	>
104	RestoRel → < Soma	<
105	Nao → not Nao	not
106	Nao → Rel	+, -, ident, (, valint, valfloat, valstring,
107	Junc → Nao RestoJunc	not, +, -, ident, (, valint, valfloat, valstring,
108	RestoJunc → λ	or, ,, lambda, :,), ;,
109	RestoJunc → and Nao RestoJunc	and
110	Exp → Junc RestoExp	not, +, -, ident, (, valint, valfloat, valstring,
111	RestoExp → λ	,, lambda, :,), ;,
112	RestoExp → or Junc RestoExp	or

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,"ident","(",")","void",",","{","}","[","]","int","float","string",";","=","if","else","elif","for","foreach",":","while","return","continue","break","read","write","valint","valfloat","valstring","lambda","+","-","*","/","mod","div","==","!=","<=",">=",">","<","not","and","or","$"],[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],
[0,114,114,114,2,114,114,114,2,114,2,2,2,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,1],
[0,114,114,114,3,114,114,114,3,114,3,3,3,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,113],
[0,113,114,114,4,114,114,114,5,114,5,5,5,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,114,114,6,114,114,114,114,7,114,7,7,7,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,114,114,8,114,9,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,114,114,113,114,113,114,114,10,114,10,10,10,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,114,114,114,113,114,11,114,113,114,113,113,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,113],
[0,12,114,114,114,114,12,114,13,114,13,13,13,114,114,12,114,114,12,12,114,12,12,12,12,12,12,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,15,114,114,114,114,15,14,114,114,114,114,114,114,114,15,114,114,15,15,114,15,15,15,15,15,15,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,113,114,114,114,114,114,114,17,114,16,16,16,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,113,114,114,114,114,114,114,114,113,18,19,20,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,113,114,114,114,114,113,114,21,114,21,21,21,114,114,113,114,114,113,113,114,113,113,113,113,113,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,22,114,114,114,114,114,114,114,114,114,114,114,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,114,114,114,114,24,114,114,114,114,114,114,114,23,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,25,114,114,114,113,114,114,114,114,114,114,114,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,114,114,114,114,26,114,114,114,114,114,114,114,26,27,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,28,114,114,114,114,37,113,114,114,114,114,114,114,114,29,113,113,30,30,114,31,32,33,34,35,36,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,38,114,114,114,114,113,113,114,114,114,114,114,114,114,113,113,113,113,113,114,113,113,113,113,113,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,114,114,114,114,114,114,114,40,114,114,114,114,114,39,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,113,114,114,114,114,113,113,114,114,114,114,114,114,114,41,113,113,113,113,114,113,113,113,113,113,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,42,114,114,114,114,42,42,114,114,114,114,114,114,114,42,43,44,42,42,114,42,42,42,42,42,42,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,113,114,114,114,114,113,113,114,114,114,114,114,114,114,113,113,113,45,46,114,113,113,113,113,113,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,113,114,114,114,114,113,113,114,114,114,114,114,114,114,113,113,113,113,113,114,47,113,113,113,113,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,113,114,114,114,114,113,113,114,114,114,114,114,114,114,113,113,113,113,113,114,113,48,113,113,113,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,50,50,114,114,114,114,114,50,114,114,114,114,49,114,114,114,114,114,114,114,114,114,114,114,114,114,50,50,50,114,50,50,114,114,114,114,114,114,114,114,114,114,50,114,114,114],
[0,113,114,114,114,114,113,113,114,114,114,114,114,114,114,113,113,113,113,113,114,113,113,51,113,113,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,113,114,114,114,114,113,113,114,114,114,114,114,114,114,113,113,113,113,113,114,113,113,113,52,113,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,113,114,114,114,114,113,113,114,114,114,114,114,114,114,113,113,113,113,113,114,113,113,113,113,53,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,113,114,114,114,114,113,113,114,114,114,114,114,114,114,113,113,113,113,113,114,113,113,113,113,113,54,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,55,55,113,114,114,114,114,55,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,55,55,55,114,55,55,114,114,114,114,114,114,114,114,114,114,55,114,114,114],
[0,114,114,56,114,57,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,113,114,114,114,114,58,113,114,114,114,114,114,114,114,113,113,113,113,113,114,113,113,113,113,113,113,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,61,62,113,114,113,114,114,63,113,114,114,114,113,114,114,114,114,114,114,113,114,114,114,114,114,114,59,59,59,113,113,113,113,113,113,113,113,113,113,113,113,113,114,113,113,114],
[0,114,114,113,114,113,114,114,114,113,114,114,114,113,114,114,114,114,114,114,113,114,114,114,114,114,114,64,65,66,113,113,113,113,113,113,113,113,113,113,113,113,113,114,113,113,114],
[0,114,114,113,114,113,114,114,67,113,114,114,114,113,114,114,114,114,114,114,113,114,114,114,114,114,114,114,114,114,113,113,113,113,113,113,113,113,113,113,113,113,113,114,113,113,114],

[0,114,114,68,114,68,114,114,69,68,114,114,114,68,114,114,114,114,114,68,114,114,114,114,114,114,114,114,68,68,68,68,68,68,68,68,68,68,68,68,114,68,68,114],
[0,70,70,114,114,114,114,114,70,113,114,114,114,114,114,114,114,114,114,114,114,71,114,114,114,114,114,114,114,70,70,70,114,70,70,114,114,114,114,114,114,114,114,114,114,70,114,114,114],
[0,114,114,114,114,114,114,114,72,114,114,114,114,114,114,114,114,114,114,114,73,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,75,75,114,114,114,114,114,75,74,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,75,75,75,114,75,75,114,114,114,114,114,114,114,114,114,114,75,114,114,114],
[0,77,77,76,114,114,114,114,77,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,77,77,77,114,77,77,114,114,114,114,114,114,114,114,114,114,77,114,114,114],
[0,114,114,113,114,79,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,78,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,81,81,114,114,114,114,114,81,80,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,81,81,81,114,81,81,114,114,114,114,114,114,114,114,114,114,114,81,114,114,114],
[0,114,114,114,114,83,114,114,114,82,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114],
[0,86,86,113,114,113,114,114,86,113,114,114,114,114,113,114,114,114,114,114,114,113,114,114,114,114,114,114,114,86,86,86,113,84,85,113,113,113,113,113,113,113,113,113,113,113,114,113,113,114],
[0,87,87,113,114,113,114,114,87,113,114,114,114,113,114,114,114,114,114,114,113,114,114,114,114,114,114,87,87,87,113,87,87,114,114,114,114,113,113,113,113,113,113,114,114,113,113,114],
[0,114,114,88,114,88,114,114,114,88,114,114,114,88,114,114,114,114,114,88,114,114,114,114,114,114,114,114,88,88,88,89,90,91,92,88,88,88,88,88,88,114,88,88,114],
[0,93,93,113,114,113,114,114,93,113,114,114,114,113,114,114,114,114,114,114,113,114,114,114,114,114,114,114,93,93,93,113,93,93,114,114,114,114,113,113,113,113,113,113,114,113,113,114],
[0,114,114,94,114,94,114,114,114,94,114,114,114,94,114,114,114,114,114,114,94,114,114,114,114,114,114,114,114,114,94,95,96,114,114,114,114,94,94,94,94,94,94,114,94,94,114],
[0,97,97,113,114,113,114,114,97,113,114,114,114,113,114,114,114,114,114,114,113,114,114,114,114,114,114,114,97,97,97,113,97,97,114,114,114,114,114,114,114,114,114,114,114,114,113,113,114],
[0,114,114,98,114,98,114,114,114,98,114,114,114,98,114,114,114,114,114,114,98,114,114,114,114,114,114,114,114,98,114,114,114,114,114,114,99,100,101,102,103,104,114,98,98,114],
[0,106,106,113,114,113,114,114,106,113,114,114,114,113,114,114,114,114,114,114,113,114,114,114,114,114,114,114,106,106,106,113,106,106,114,114,114,114,114,114,114,114,114,114,114,105,113,113,114],
[0,107,107,113,114,113,114,114,107,113,114,114,114,113,114,114,114,114,114,114,113,114,114,114,114,114,114,114,107,107,107,113,107,107,114,114,114,114,114,114,114,114,114,114,114,107,114,113,114],
[0,114,114,108,114,108,114,114,114,108,114,114,114,108,114,114,114,114,114,114,108,114,114,114,114,114,114,114,114,114,108,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,114,109,108,114],
[0,110,110,113,114,113,114,114,110,113,114,114,114,113,114,114,114,114,114,114,113,114,114,114,114,114,114,114,110,110,110,113,110,110,114,114,114,114,114,114,114,114,114,114,110,114,114,114],
[0,114,114,111,114,111,114,114,114,111,114,114,114,111,114,114,114,114,114,114,111,114,114,114,114,114,114,114,114,114,111,114,114,114,114,114,114,114,114,114,114,114,114,114,114,112,114]]

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.

{"2":[1,2],"3":[7,-3,4,-2,-1,3],"4":[-4],"5":[10],"7":[5,6],"9":[5,6,-5],"10":[-1,10],"11":[-7,9,8,-6],"13":[8,12],"15":[9,17],"16":[11],"17":[-9,11,-8],"18":[-10],"19":[-11],"20":[-12],"21":[-13,13,10],"22":[14,15],"24":[13,-5],"25":[16,-1],"27":[54,-14],"28":[18],"29":[20],"30":[22],"31":[23],"32":[24],"33":[26],"34":[27],"35":[28],"36":[29],"37":[32],"38":[-13,54,-14,19,-1],"40":[-9,54,-8],"41":[21,17,-3,54,-2,-15],"43":[17,-16],"44":[21,17,-3,54,-2,-17],"45":[17,-3,54,-14,-1,-13,54,-13,54,-14,-1,-2,-18],"46":[17,-20,54,-14,-1,-19],"47":[17,-3,54,-2,-21],"48":[-13,25,-22],"50":[54],"51":[-13,-23],"52":[-13,-24],"53":[-13,-3,-1,-2,-25],"54":[-13,-3,30,-2,-26],"55":[31,54],"57":[30,-5],"58":[-7,9,-6],"59":[34],"60":[36,-1],"61":[-3,40,-2,-1],"62":[-3,54,-2],"63":[35],"64":[-27],"65":[-28],"66":[-29],"67":[-9,42,-8],"69":[-9,37,-8],"70":[38,54],"71":[39,-20],"73":[39,-20],"75":[54],"77":[41,54],"78":[-30],"79":[41,54,-5],"81":[43,54],"83":[43,54,-5],"84":[44,-31],"85":[44,-32],"86":[33],"87":[46,44],"89":[46,44,-33],"90":[46,44,-34],"91":[46,44,-35],"92":[46,44,-36],"93":[48,45],"95":[48,45,-31],"96":[48,45,-32],"97":[50,47],"99":[47,-37],"100":[47,-38],"101":[47,-39],"102":[47,-40],"103":[47,-41],"104":[47,-42],"105":[51,-43],"106":[49],"107":[53,51],"109":[53,51,-44],"110":[55,52],"112":[55,52,-45]}

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)