Top-Down Parsing: LL Parsin

Introduction to Top-Down Parsing

1. Previous lesson showed topdown parser that parses from left to right fulfills our syntactic checking needs for the compiler
   - This lesson covers LL(1) parsing
     - Top down, left to right
     - Can be automatically generated from its grammar
     - Automated scanners and parsers simplify the construction of the compiler front end

LL(1) Parser Quiz

1. Which of the following characteristics are true for an LL(1) parser?
   - They perform backtracking
   - They perform leftmost derivation (true)
   - They perform one token look-ahead (true)
   - Not every context free language can be parsed by LL(1) parsers (true)

LL(1) Parsing

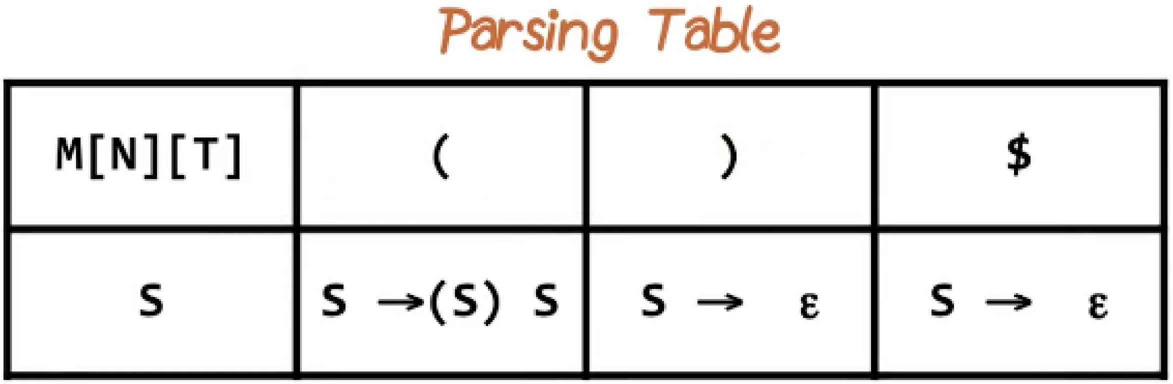
1. Overview
   - Push start symbol onto stack
   - Replace non-terminal symbol on stack using grammar rules
   - If top of stack matches input token, both are to be discarded, mismatch is a syntax error
   - If, eventually, both stack and input string are empty then it is a successful parse

Sample Example

1. The Grammar
   - S -> (s) S | ' '
2. The Input
   - ()$
   - $ means end of input
3. The Stack
   - $
   - $S (Replace S with a rule from the grammar)
   - $S)S((Match the top of the stack with the next input character)
   - $S)S (Remove the matching symbol)
   - $S) (Pick S going to epsilon)
   - $S (Remove the matching symbol)
   - $ (Pick S going to epsilon)

LL(1) Parser

1. The top of the stack may contain tokens or non-terminals
2. How does the parser pick the right rule to match the input?
   - LL(1) parser is deterministic: rule for expansion selected by 1 token lookahead



Parsing Table

M[N][T]	(	)	$
S	S →(S) S	S → ε	S → ε

Parsing Table

Parse Table

1. Parse table is the brain of the parser
    - LL(1) parse table consists of a column for each token and a row for each non-terminal symbol
    - A grammar is LL(1) grammar if the associated LL(1) parsing table has at most one production rule in each table entry
    - LL(1) grammar is a proper subset of context-free grammar

Table Construction

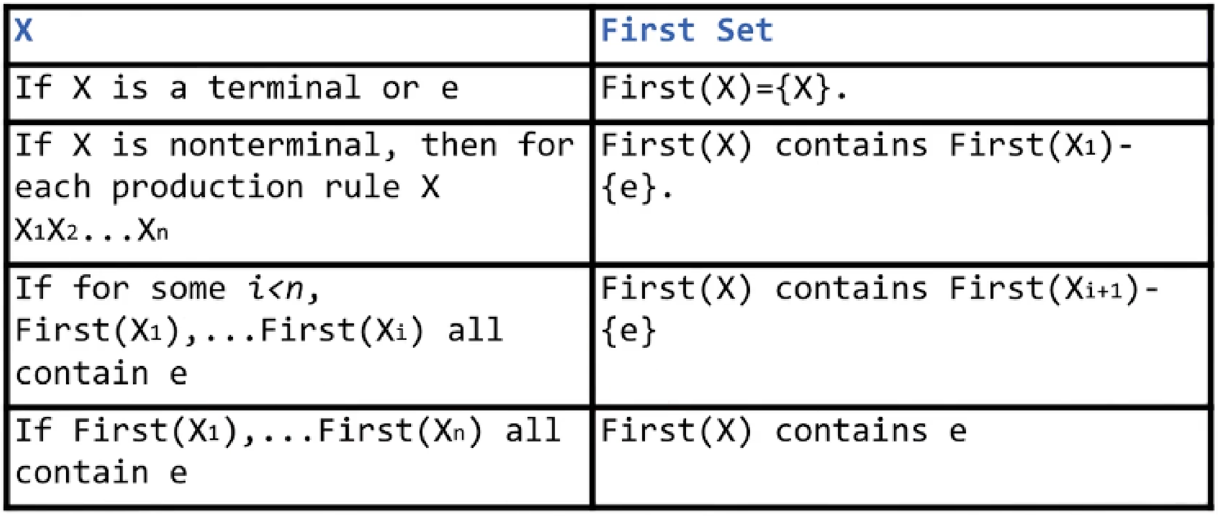
1. How to construct the parsing table if grammar is complex?
2. Grammar
    - exp -> term exp’
    - exp’ -> addop term exp’ | ’ ’
    - addop -> + | -
    - term -> factor term’
    - term’ -> mulop factor term’ | ’ ’
    - mulop -> *
    - factor -> (exp) | num

Grammar Rules

1. The grammar must not be ambiguous
2. The grammar for LL(1) parsing must not be left recursive

First Sets

1. First Set:
    - X -> X1X2X3X4…Xn
    - First set for a symbol (on the left hand side of a rule) is the set of tokens that we find beginning the right hand side of the rule
2. Let X be a grammar symbol (a terminal or nonterminal) or e. Then the set First(X) is defined as follows:
    - Continue to grow the first set until we find that a first of some Xk is not null

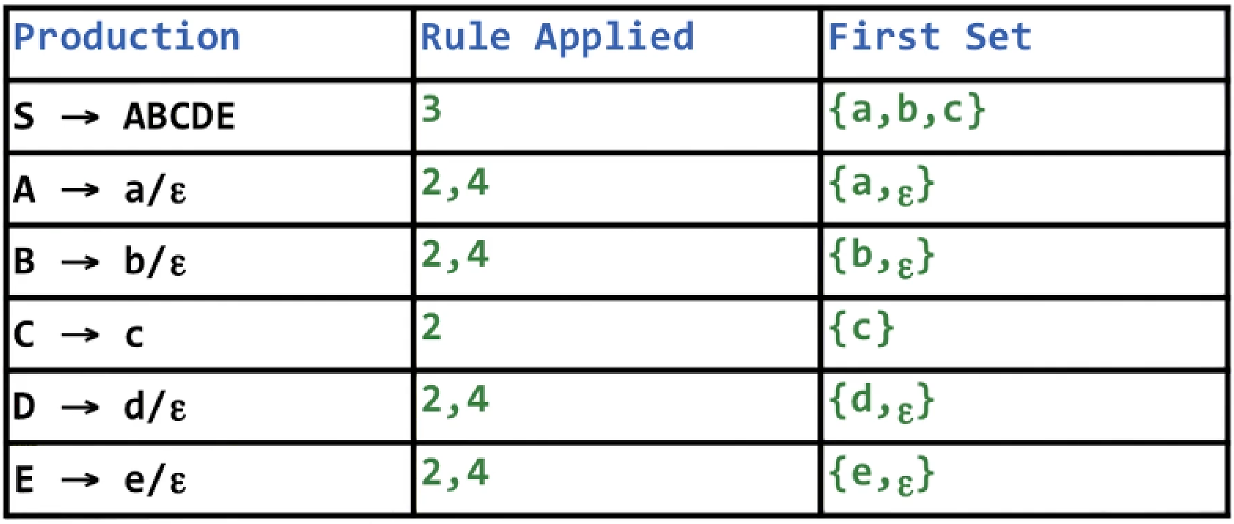


X	First Set
If X is a terminal or e	First(X)={X}.
If X is nonterminal, then for each production rule X $X_1X_2...X_n$	First(X) contains First(X_1)-{e}.
If for some *i<n*, First(X_1),...First(X_i) all contain e	First(X) contains First(X_{i+1})-{e}
If First(X_1),...First(X_n) all contain e	First(X) contains e

First Sets

First Set Quiz

1. Given the following grammar:
   - S -> ABCDE
   - A -> a | ' '
   - B -> b | ' '
   - C -> c
   - D -> d | ' '
   - E -> e | ' '
2. Apply the following rules and find first sets for each of the non-terminals: S, A, B, C, D, and E. Show rules applied to find the sets for each non-terminal.



Production	Rule Applied	First Set
S → ABCDE	3	{a,b,c}
A → a/ε	2,4	{a,ε}
B → b/ε	2,4	{b,ε}
C → c	2	{c}
D → d/ε	2,4	{d,ε}
E → e/ε	2,4	{e,ε}

First Sets Quiz

First Set Algorithm

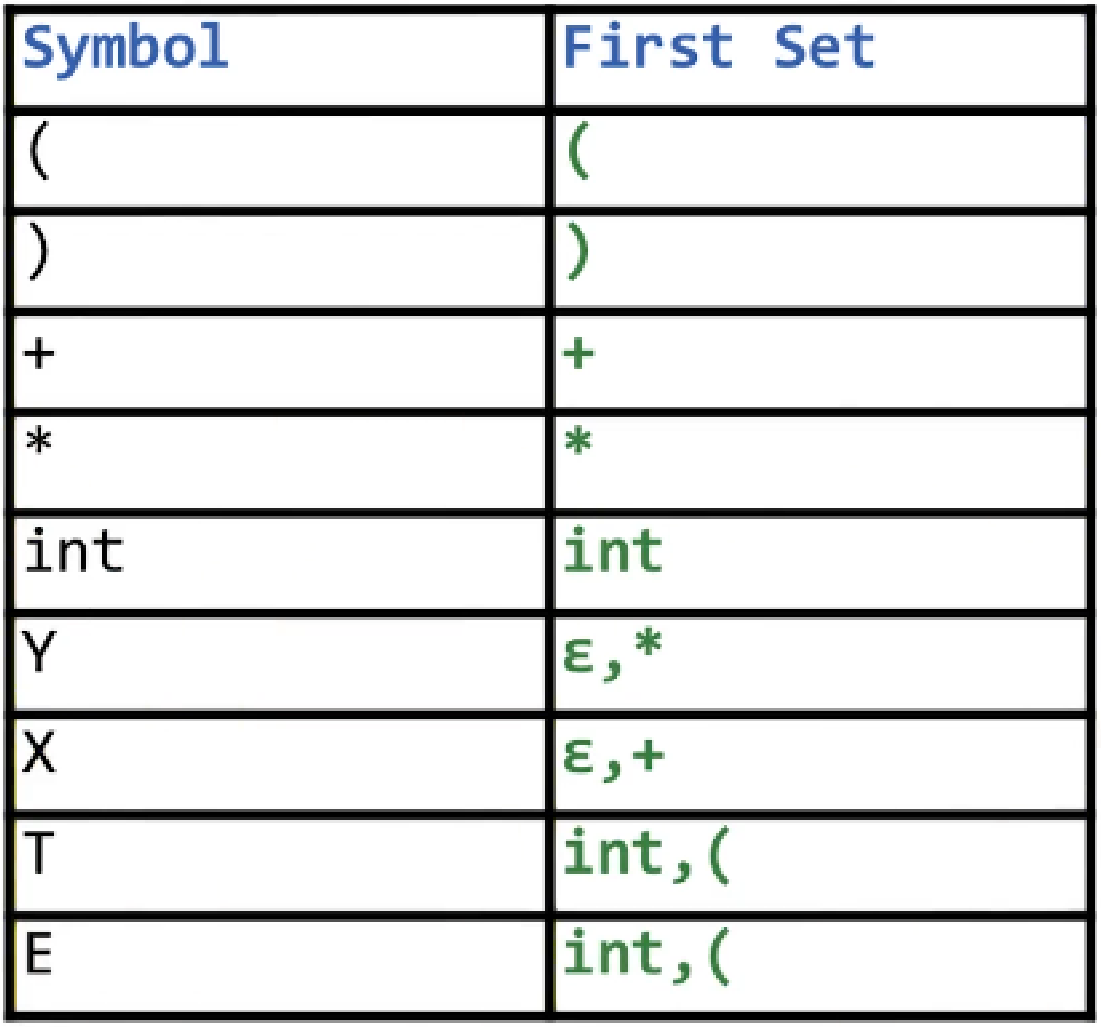
1. Algorithm:

```
for each nonterminal X do First(X) := {}
while there are changes to any First(X) do
    for each production rule X -> X1X2...Xn do
        k := 1;
        while k <= n do
            First(X) = First(X) U (First(Xk) - {e})
            if e is not in First(Xk) then break;
            k := k + 1;
        if (k > n) then First(X) = First(X) U {e}
```

2. Real world example:
    - stmt -> if-stmt | other
    - if-stmt -> if (exp) stmt else-part
    - else-part -> else stmt | e
    - exp -> 0 | 1
    - Tokens are if, else, other, 0, 1
3. First sets
    - First(stmt) = {other} U First{if-stmt} = {other, if}
    - First{if-stmt} = {if}
    - First{else-part} = {else, e}
    - First{exp} = {0, 1}

First Set Example Quiz

1. Given the following grammar, determine the first sets.
    - E -> T X
    - X -> + E
    - X -> e
    - T -> int Y
    - T -> (E)
    - Y -> * T
    - Y -> e

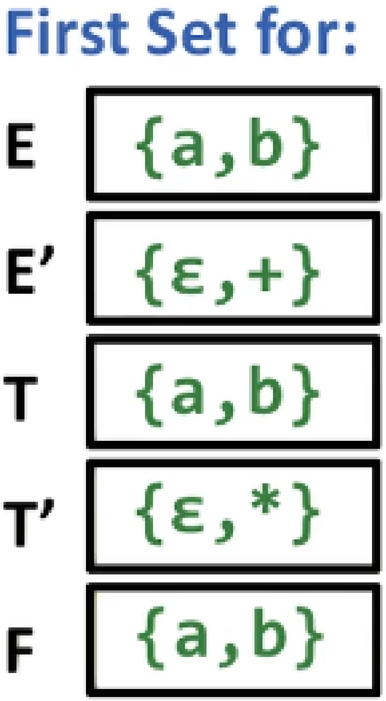


Symbol	First Set
(	(
)	)
+	+
*	*
int	int
Y	ε,*
X	ε,+
T	int,(
E	int,(

First Sets Quiz

First Set Quiz 2

1. Given the following grammar, determine the first sets.
    - E -> TE’
    - E’ -> +TE’
    - E’ -> e
    - T -> FT’
    - T’ -> *FT’
    - T’ -> e
    - F -> a
    - F -> b



First Set for:

E	{a,b}
E’	{ε,+}
T	{a,b}
T’	{ε,*}
F	{a,b}

First Sets Quiz

Follow Sets Part 1

1. What is a follow set?
   - Follow set of A is those symbols which will follow after A and is used to determine if a rule such as A -> e should be invoked to remove the A to expose the tokens that follow A for matching them.
2. Given a nonterminal A, the set Follow(A) is defined as:
   - If A is start symbol, then $ is in Follow(A)
   - If there is a production rule B -> a A B’, then Follow(A) contains First(B’) - {e}
   - If there is a production fule B -> a A B’ and B’ is nullable, then Follow(A) contains Follow(B)
   - Notes:
     - $ is needed to indicate end of string
     - e is never a member of Follow set

Follow Sets Part 2

1. Construction:

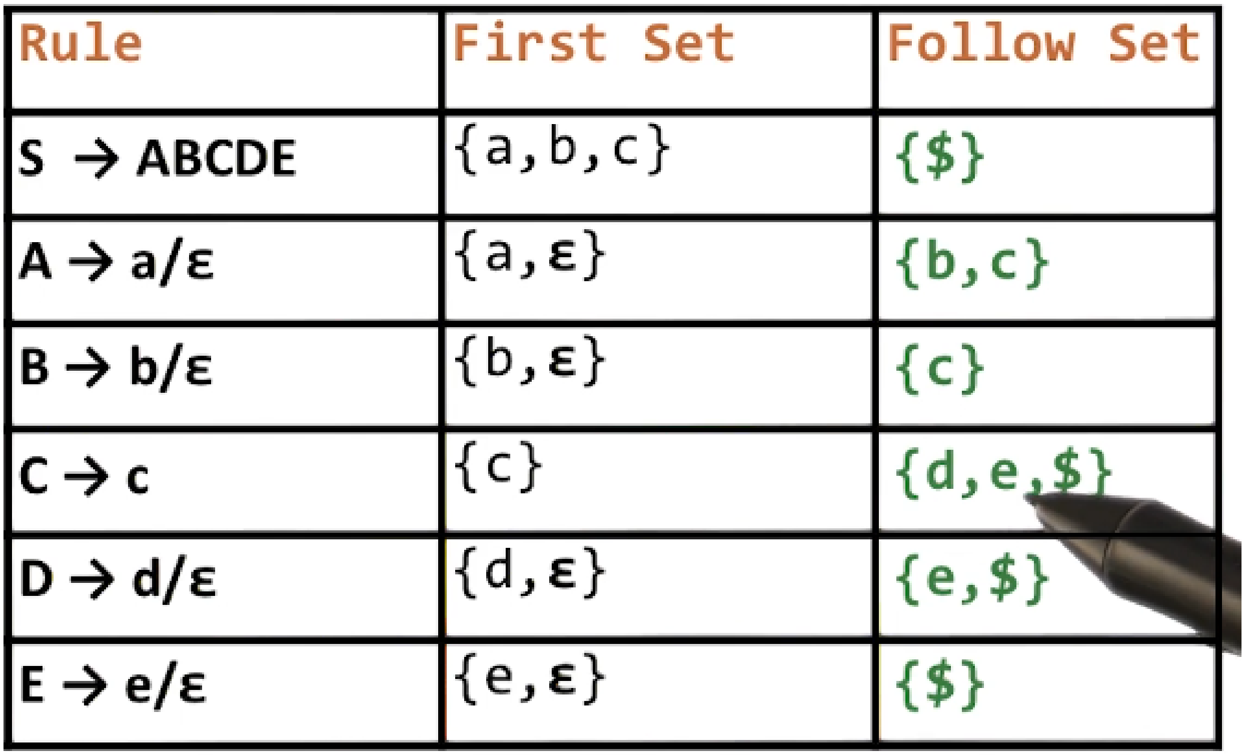
```
for each nonterminal X do
    Follow(X) := {$} for start symbol or {} for others

while there are changes to any Follow(X) do
    for each production rule X -> X1X2...Xn do
        for each Xi that is a nonterminal do
            Follow(Xi) = Follow(Xi) U (First(Xi+1...Xn) - {e}
                if e is in First(Xi+1...Xn) then
                    Follow(Xi) = Follow(Xi) U Follow(X)
```

Follow Sets Part 3

1. Grammar:
   - stmt -> if-stmt | other
   - if-stmt -> if (exp) stmt else-part
   - else-part -> else stmt | e
   - exp -> 0 | 1
2. Example:
   - Follow(exp) = {)}
   - Follow(else-part) = Follow(if-stmt) = Follow(stmt)
   - Follow(stmt) = {$} U First(else-part)-{e} U Follow(if-stmt) = {$,else}

Follow Set Quiz

1. Given the following grammar:
   - S -> ABCDE
   - A -> a | ’ ’
   - B -> b | ’ ’
   - C -> c
   - D -> d | ’ ’
   - E -> e | ’ ’
2. Find the follow sets:



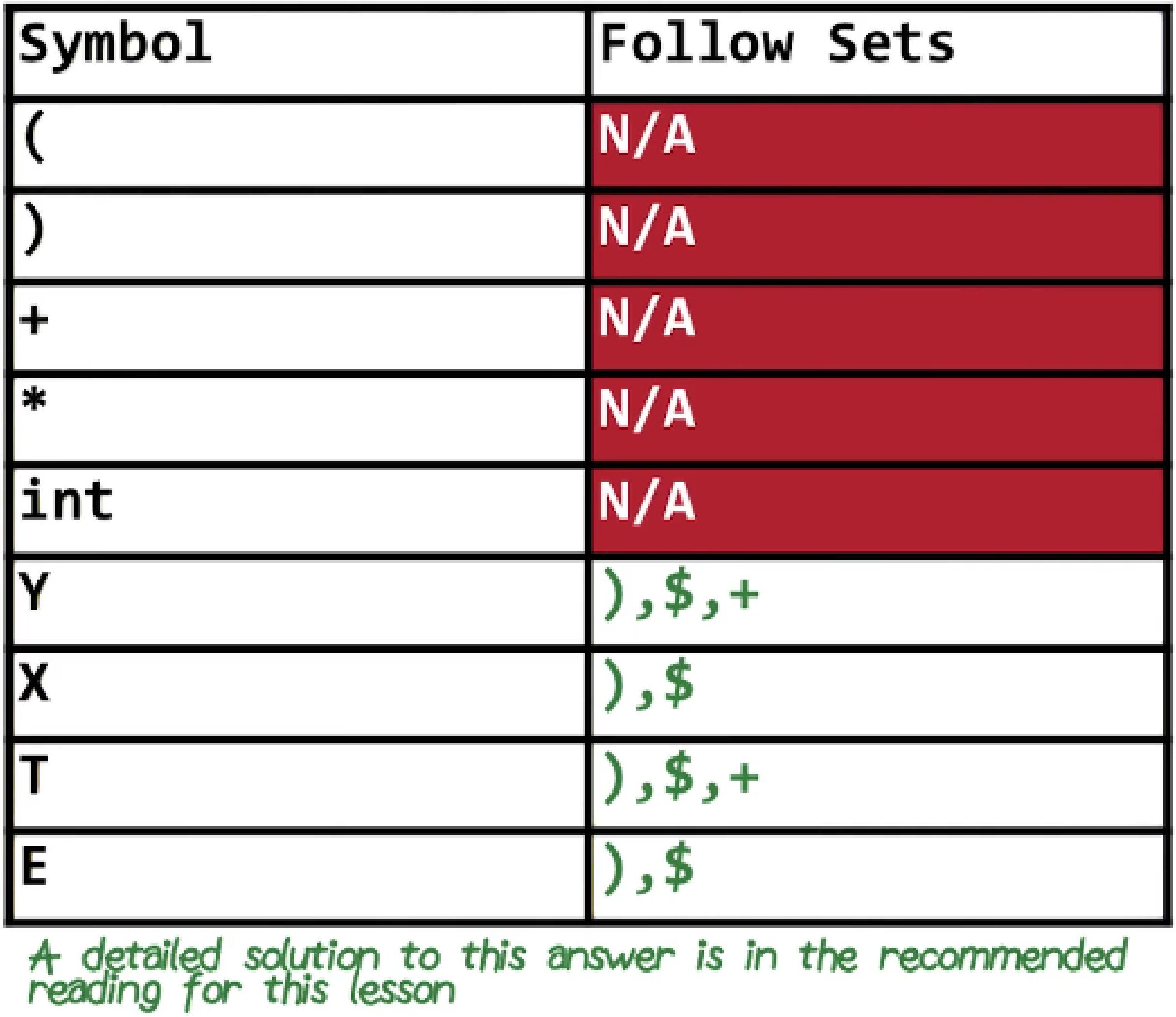
Rule	First Set	Follow Set
S → ABCDE	{a,b,c}	{$}
A → a/ε	{a,ε}	{b,c}
B → b/ε	{b,ε}	{c}
C → c	{c}	{d,e,$}
D → d/ε	{d,ε}	{e,$}
E → e/ε	{e,ε}	{$}

Follow Sets Quiz

Follow Set Example Quiz

1. Given the following grammar, determine the follow sets.
   - E -> T X

- X -> + E
- X -> e
- T -> int Y
- T -> (E)
- Y -> * T
- Y -> e



Symbol	Follow Sets
(	N/A
)	N/A
+	N/A
*	N/A
int	N/A
Y	),$,+
X	),$
T	),$,+
E	),$

A detailed solution to this answer is in the recommended reading for this lesson

Follow Sets Quiz

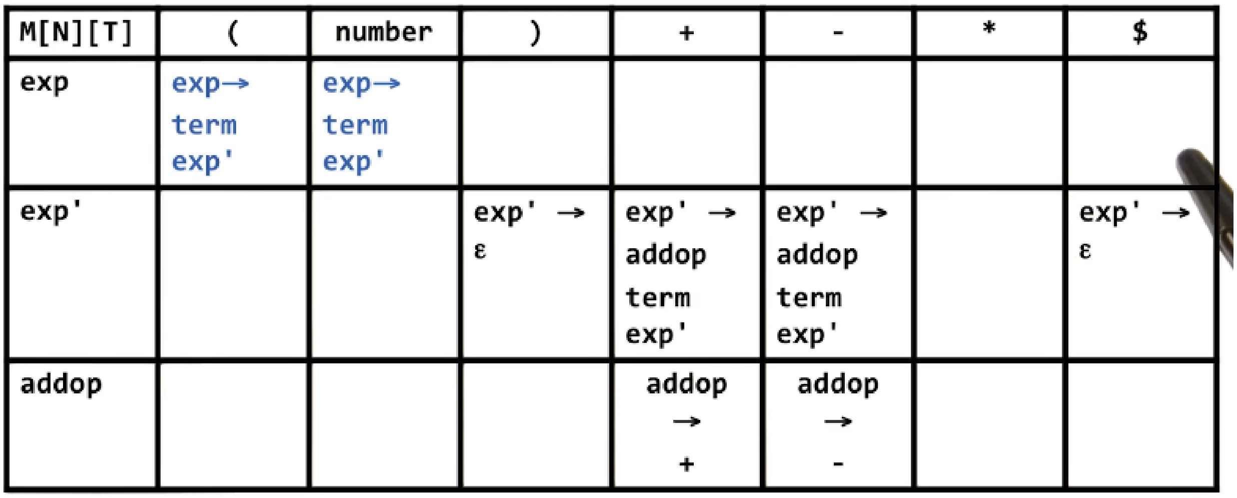
Parsing Tables

1. Repeat the following two steps for each nonterminal A and production choice A -> a
    - For each token a in First(a), add A -> a to the entry M[A,a]
    - If e is in First(a), for each element a in Follow(A) (a token or $), add A -> a to the entry M[A,a]

Complete Example

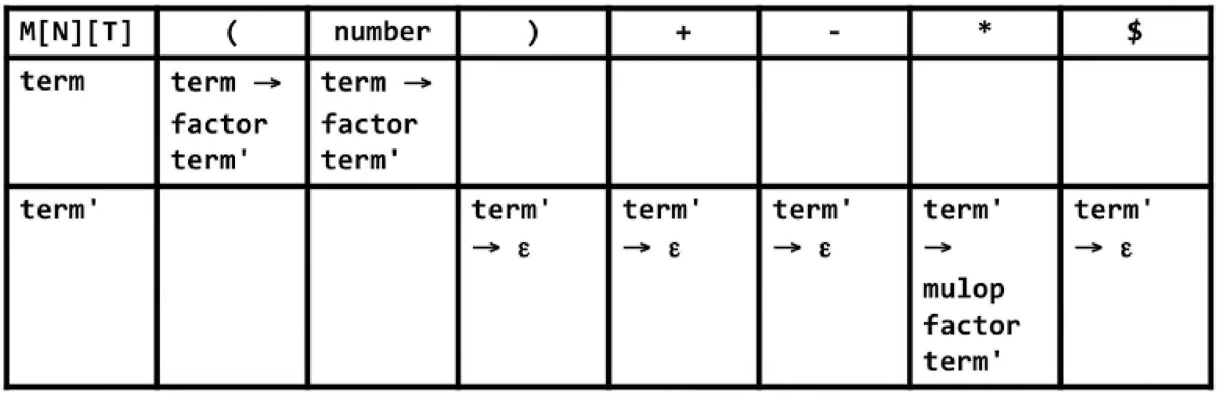
1. Grammar
    - exp -> term exp’
    - exp’ -> addop term exp’ | ’ ’

    - addop -> + | -
    - term -> factor term’
    - term’ -> mulop factor term’ | ’ ’
    - mulop -> *
    - factor -> (exp) | num
2. First sets
    - First(exp) = { (number }
    - First(exp’) = { + - e }
    - First(addop) = { + - }
    - First(term) = { (number }
    - First(term’) = { * e }
    - First(mulop) = { * }
    - First(factor) = { (number }
3. Follow sets
    - Follow(exp) = { $) }
    - Follow(exp’) = { $ }
    - Follow(addop) = { (number }
    - Follow(term) = { + - $) }
    - Follow(term’) = { + - $) }
    - Follow(mulop) = { (number }
    - Follow(factor) = { * + - $) }
4. Predict sets
    - Predict(A -> a) = First(a) if First(a) does not contain epsilon
    - Predict(A -> a) = First(a) - {e} U Follow(A) otherwise
5. How to generate parsing table from predict sets?
    - If a token t appears in Predict(A -> a), put rule A -> a in entry M[A} [t]



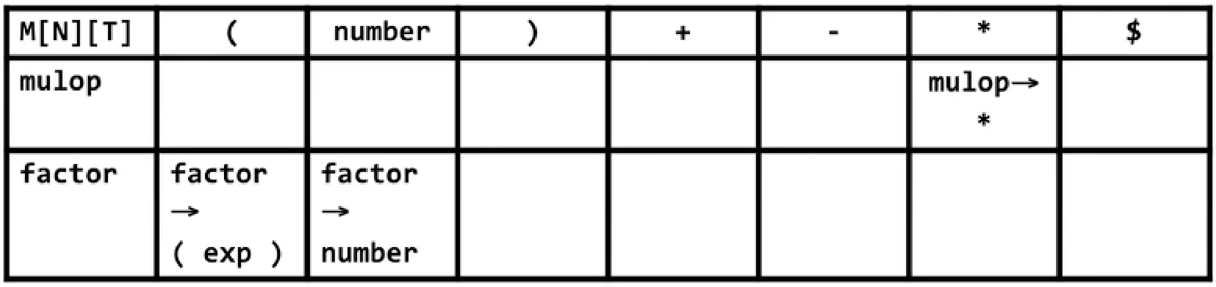
M[N][T]	(	number	)	+	-	*	$
exp	exp→ term exp'	exp→ term exp'					
exp'			exp' → ε	exp' → addop term exp'	exp' → addop term exp'		exp' → ε
addop				addop → +	addop → -		

Final Parse Table Part 1



M[N][T]	(	number	)	+	-	*	$
term	term → factor term'	term → factor term'					
term'			term' → ε	term' → ε	term' → ε	term' → mulop factor term'	term' → ε

Final Parse Table Part 2



M[N][T]	(	number	)	+	-	*	$
mulop						mulop→ *	
factor	factor → (exp)	factor → number					

Final Parse Table Part 3