NULLABLE(terminal) = false

NULLABLE(sequence of symbols) = true iff none of the symbols are NOT NULLABLE

NULLABLE(production) = NULLABLE(RHS)

NULLABLE(nonterminal) = any of its productions are NULLABLE

FIRST(terminal) = { terminal }

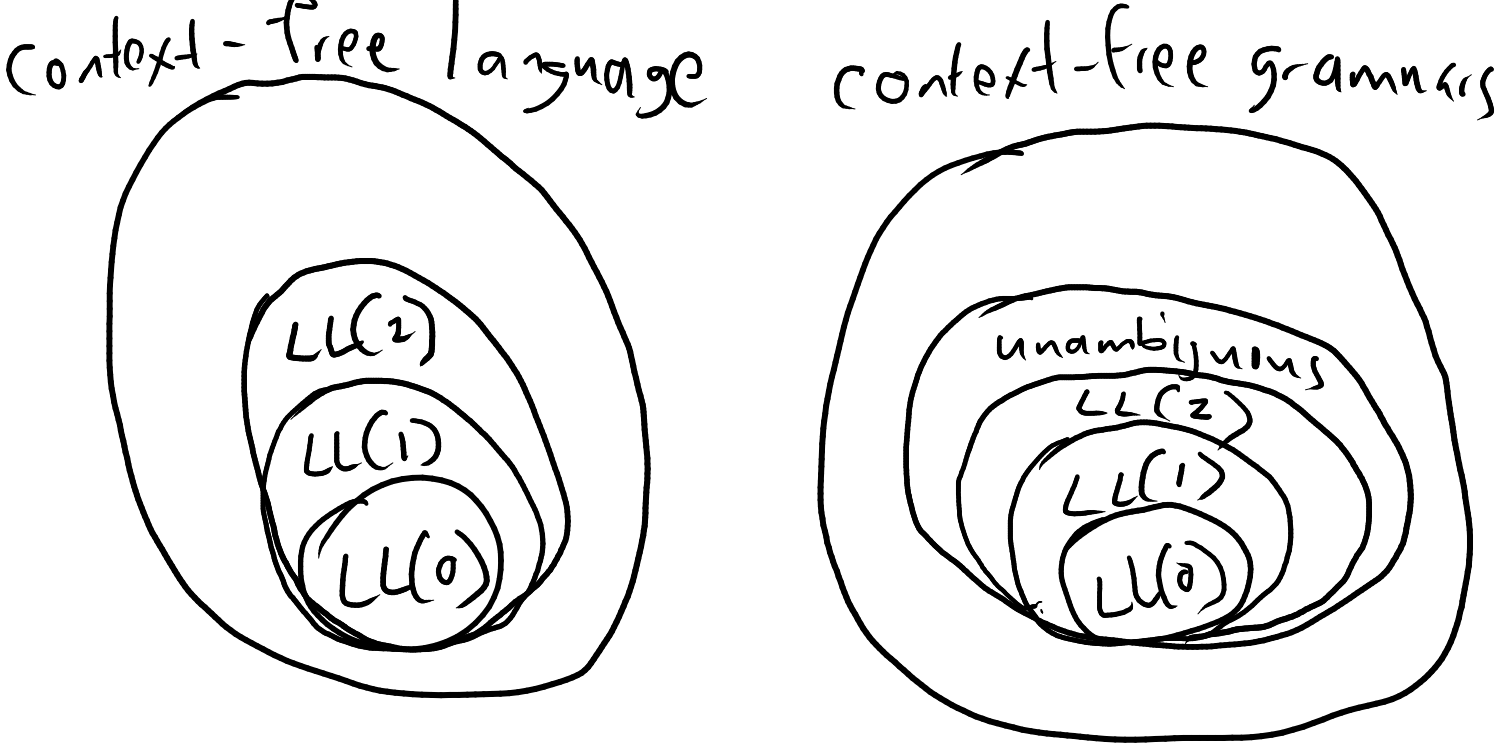
FIRST(X1 X2 ... Xn) = union of FIRST(X1), FIRST(X2), ..., FIRST(Xk)
where X1, X2, ... X_{k - 1} are NULLABLE and Xk is NOT NULLABLE

FIRST(production) = FIRST(RHS)

FIRST(nonterminal) = union of FIRST of RHS of its productions

grammar is LL(k) iff
we can create an LL(k) parser

language is LL(k) iff
∃ an LL(k) grammar for it



output is a 2d table M

```
for each production P in the grammar {
    A = lhs(P)
    for each terminal a in FIRST(P) {
        M[A, a] = P
        if NULLABLE(P) { // equivalently, epsilon in FIRST(P)
            for each terminal b in FOLLOW(A) {
                M[A, b] = P
```

1 E -> T E'
2 E' -> + T E' | epsilon 3
4 T -> F T'
5 T' -> * F T' | epsilon 6
7 F -> (E) | id 8

nonterminal	FIRST	FOLLOW
E	(, id	), $
E'	+, epsilon	), $
T	(, id	+,), $
T'	*, epsilon	+,), $
F	(, id	+, *,), $

	(	)	+	*	id	$
E	1				1	
E'		3	2			3
T	4				4	
T'		6	6	5		6
F	7				8	

```
// X refers to the top of the stack
// a refers to the next input token
while X != $ {
    if X is a terminal {
        if X == a {
            pop_stack();
            advance_input();
        } else {
            error();
        }
    } else { // X is a nonterminal
        if M[X, a] is empty {
            error();
        } else {
            P = M[X, a];
            output(P);
            pop_stack();
            for Yi in reverse(rhs(P)) {
                push(Yi);
            }
        }
    }
}
return a == $;
```

stack

top → S

$

a b c ... $

↑

A → X Y Z

top → X

Y

Z

$

Z

$

if_stmt → if (expr) {

statements

} else {

statements

}