Language is a set of strings.

Language on Σ is a subset Σ^*
member 2^{Σ^*}

left-linear grammar

$A \to aB \mid c$

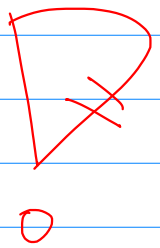
$A \to aB$

$A \to c$

3

A regular language is:

- DFA
- NFA
- left-linear grammar
- regular expression



2

Context-free Language (CFL)

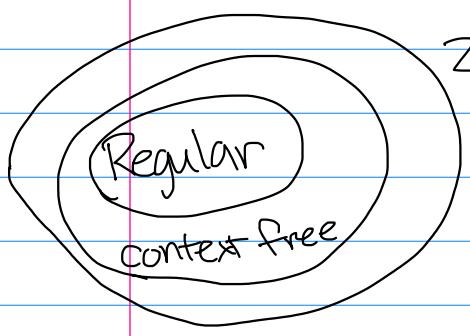
↳ Pushdown Automata (PDA)

↳ Context-free grammar (CFG)

subset of 2^{Σ^*} = set of language

reg. expr $\subset 2^{\Sigma^*}$

reg. lang. $\in 2^{\Sigma^*}$



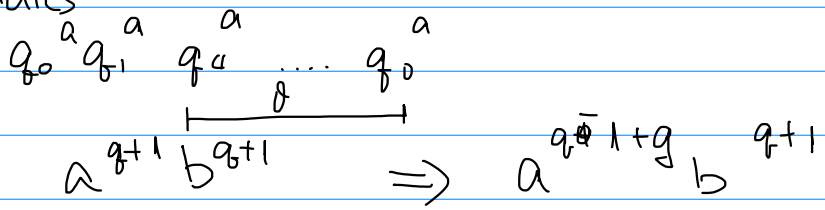
Reg $= \{ a^k \mid k \geq 0 \}$

$= \{ \varepsilon, a, aa, aaa, \ldots \}$

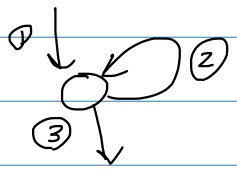
Context-free $= \{ a^k b^k \mid k \geq 0 \}$

$= \{ \varepsilon, ab, aabb, aaabbb, \ldots \}$

DFA q states



Weird proof with Pigeon Hole Principle that shows contradiction



Context free grammar says: $\{ S \to \varepsilon \mid aSB$

0 recursively enumerable

1 Context-free language (context-sensitive)

Context-Free Grammar

$G = (P, N, T, S)$

- G: grammar
- P: punctuation
- N: set of non-terminals
- T: set of terminals (bold)
- S: start symbol, $S \in N$

In a CFG production

$$A \Rightarrow \alpha$$

$A \in N$ $\quad \alpha \in (N \cup T \cup \{\varepsilon\})^*$

$G =$

$A \rightarrow B+B$
$A \rightarrow B$
$B \rightarrow C * C$
$B \rightarrow C$
$C \rightarrow d$
$C \rightarrow (A)$

list of productions where left is the start symbol,

N = set of capital letter $= \{A, B, C\}$
OR the LHS of the production

$T = \{+, *, (,)\}$
OR other stuff on RHS

Derivation using grammar G

start symbol

$A \Rightarrow B + B \Rightarrow C + B \Rightarrow d + B \Rightarrow d + C * C \Rightarrow d + C * d \Rightarrow d + d * d$

- $A \Rightarrow B + B$: replaced with RHS of an A-production
- $B \Rightarrow C$: choose one non-terminal and replace
- $d + B \Rightarrow d + C * C$: choose one non-terminal and replace
- $d + C * d \Rightarrow d + d * d$: no nonterminals

Process ends!!

Right-most Derivation:

Any string that appears in a leftmost derivation is a
left-sentential form ⇒ right most - right sentential

$A \overset{*}{\Rightarrow} d + (d*d) * d$

↑ sentences only contain terminals

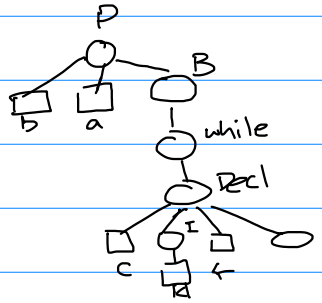
Top-down parsing is usually LL() parsing ← leftmost derivation

read input from left to right
(start) (end)

```
box main {
    const x ← 2;
     print x ;
}
```

```
ParseProgram ()
   expect ("box") ;
   expect ("main");
   parseBoxBody ();
parseBoxBody ()
   :
   while
     parseStatement ()
```

```
parseStatement ()
   ParseDecl ()
parseDecl ()
   expect ("const", "init")
ParseIdentifier ()
```



Bottom-up Parsing is usually LR (Rightmost derivation; from left to right)

Grammar

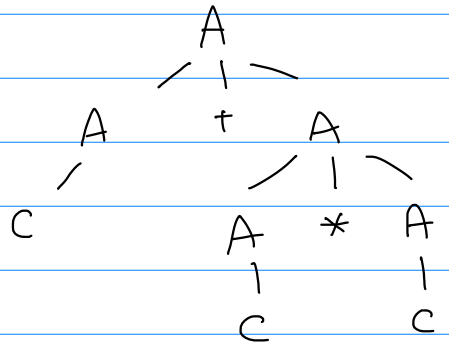
$A \to A + A$

$A \to A * A$

$A \to c$

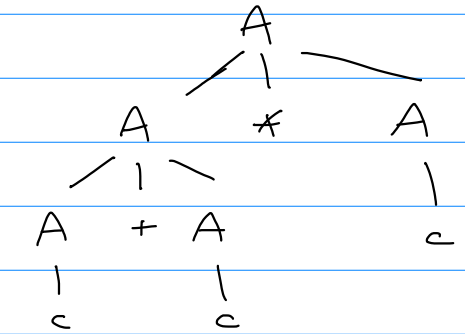
rightmost derivation

$A \to A + A \to A + A * A \overset{*}{\Rightarrow} c + c * c$



leftmost derivation.

$A \to A * A \to A + A * A \overset{*}{\Rightarrow} c + c * c$



2 distinct parse trees → AMBIGUOUS Grammar

Dangling "if"