CS3102 Theory of Computation

Context Free Languages

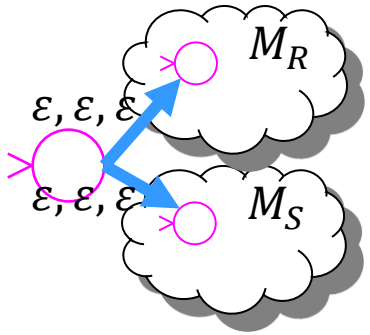
- For a PDA M, the language of M (denoted $L(M)$) refers to the set of strings accepted by the machine
  - $L(M) = \{s \in \Sigma^* | M \text{ accepts } s\}$
- The set of all languages decided by some PDA is call the **Context Free Languages**
  - Equivalent to the languages describable by Context Free Grammars
- A particular language decided by some FSA is called a **Context Free Language**
- All regular languages are context free (because if we choose not to use the stack, a PDA is a NFA)
- All context free languages can be decided by a Java program using only linear memory (relative to length of word)

Closure Properties of CFLs

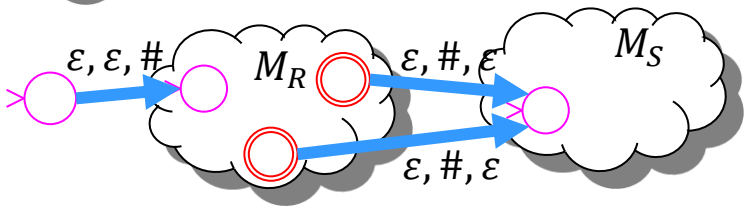
- Context Free Languages are closed under:
  - Union
  - Concatenation
  - Kleene Star
  - Intersection with Regular languages
- Context Free Languages are not closed under:
  - Complementation
  - Intersection with CFLs

CFLs closed under:

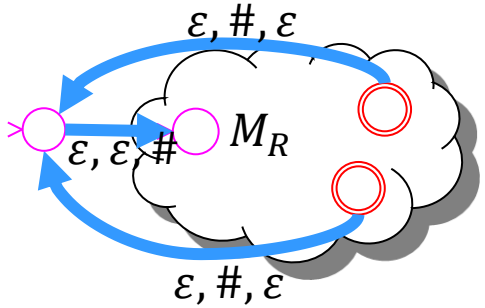
- Union



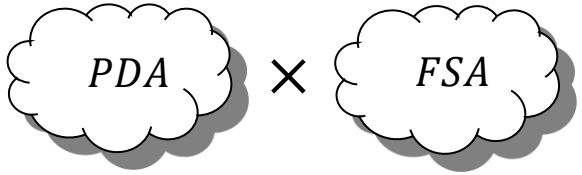
- Concatenation



- Kleene Star



- Intersection with regular languages



Non-Context-Free Languages

- Pumping Lemma for CFLs exists (but we won't cover it)
- $a^n b^n c^n$
  - Intuition: When deciding $a^n b^n$ we pushed for a's and popped for b's. Once we popped everything for b's, we "forgot" what n was.
- $a^n b^m c^n d^m$
  - Intuition: If I count the number of a's and b's using the stack, the b's are "blocking" the a's count from being checked against c's.
- $\{ww \mid w \in \Sigma^*\}$
  - Intuition: we could push the first half onto the stack, but when popping off it's in reverse order.

CFLs not closed under Intersection

- $a^n b^n c^k \cap a^j b^n c^n = a^n b^n c^n$
- Intersecting:
  - Some number of a's, same number of b's, any number of c's (a's match b's)
  - Any number of a's, some number of b's, some number of c's (b's match c's)
- Results in:
  - Some number of a's, same number of b's, same number of c's (a's match b's match c's)

CFLs not closed under Complement

- Prove this with the people around you

Context Free Grammars

- A way of describing context free languages
- Gives a list of subtitution rules you can apply to generate strings
- Any string that can be generated by some sequence of these substitutions is in the language of that grammar
- A language is generated by some CFG if and only if it is decided by some PDA

Context Free Grammar

- Basic idea: Apply substitutions to construct strings
- Finite set of variables/non-terminals: $V = \{V_1, V_2, \ldots, V_k\}$
- Finite set of terminals: $\Sigma \cup \{\varepsilon\}$
- Finite set of productions/subtitutions: $R: V \rightarrow (\Sigma \cup V \cup \{\varepsilon\})^*$
- Start symbol: S
- To produce a string:
  - Start with the start symbol,
  - As long as your string still contains non-terminals,
  - Subtitute a non-terminal using a production rule

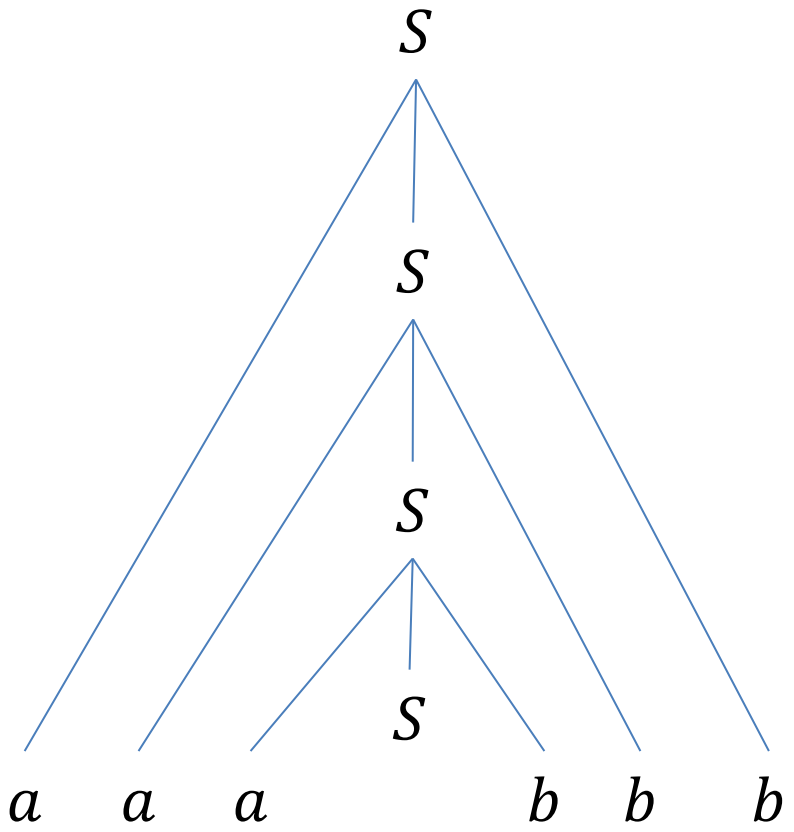
CFG Example: $a^n b^n$

$$S \rightarrow aSb \mid \varepsilon$$

To produce $aaabbb$

Current String	Production rule applied
S	$S \rightarrow aSb$
aSb	$S \rightarrow aSb$
$aaSbb$	$S \rightarrow aSb$
$aaaSbbb$	$S \rightarrow \varepsilon$
$aaabbb$	All terminals

Parse Tree



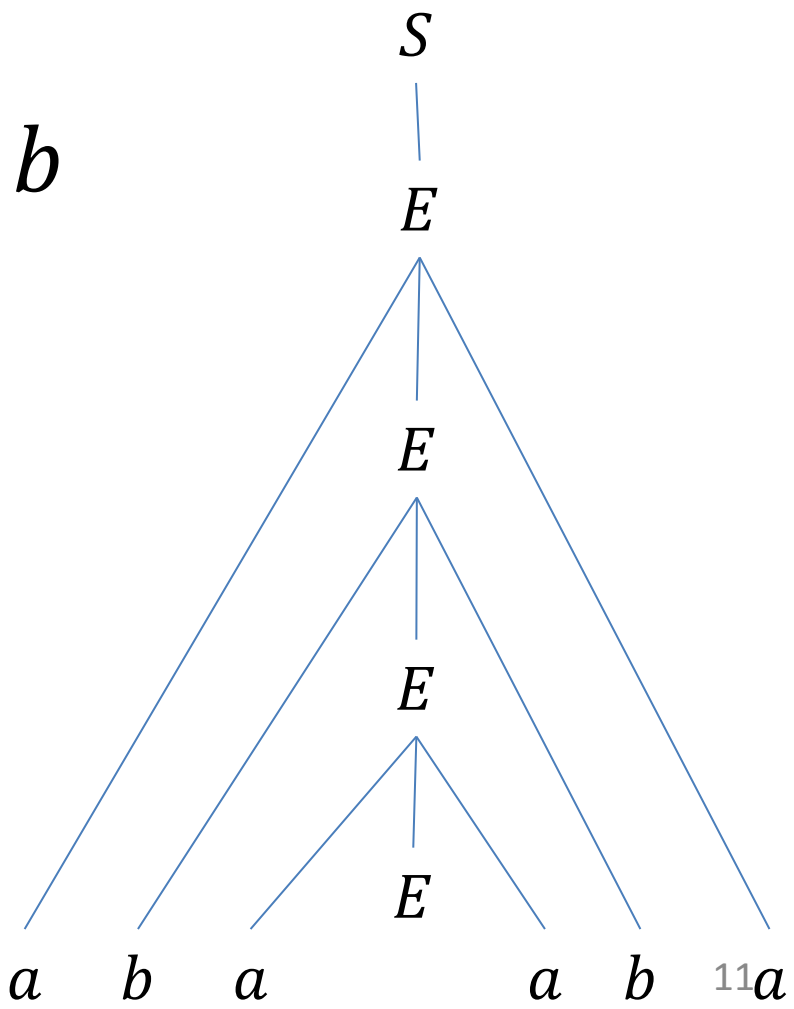
CFG for palindromes ($w = w^R$)

$$S \rightarrow E|O$$
$$E \rightarrow aEa \mid bEb \mid \varepsilon$$
$$O \rightarrow aOa \mid bOb \mid a \mid b$$

To produce: abaaba

Current String	Production rule applied
S	$S \rightarrow E$
E	$E \rightarrow aEa$
aEa	$E \rightarrow bEb$
$abEba$	$E \rightarrow aEa$
$abaEaba$	$E \rightarrow \varepsilon$
$abaaba$	All terminals

Parse Tree



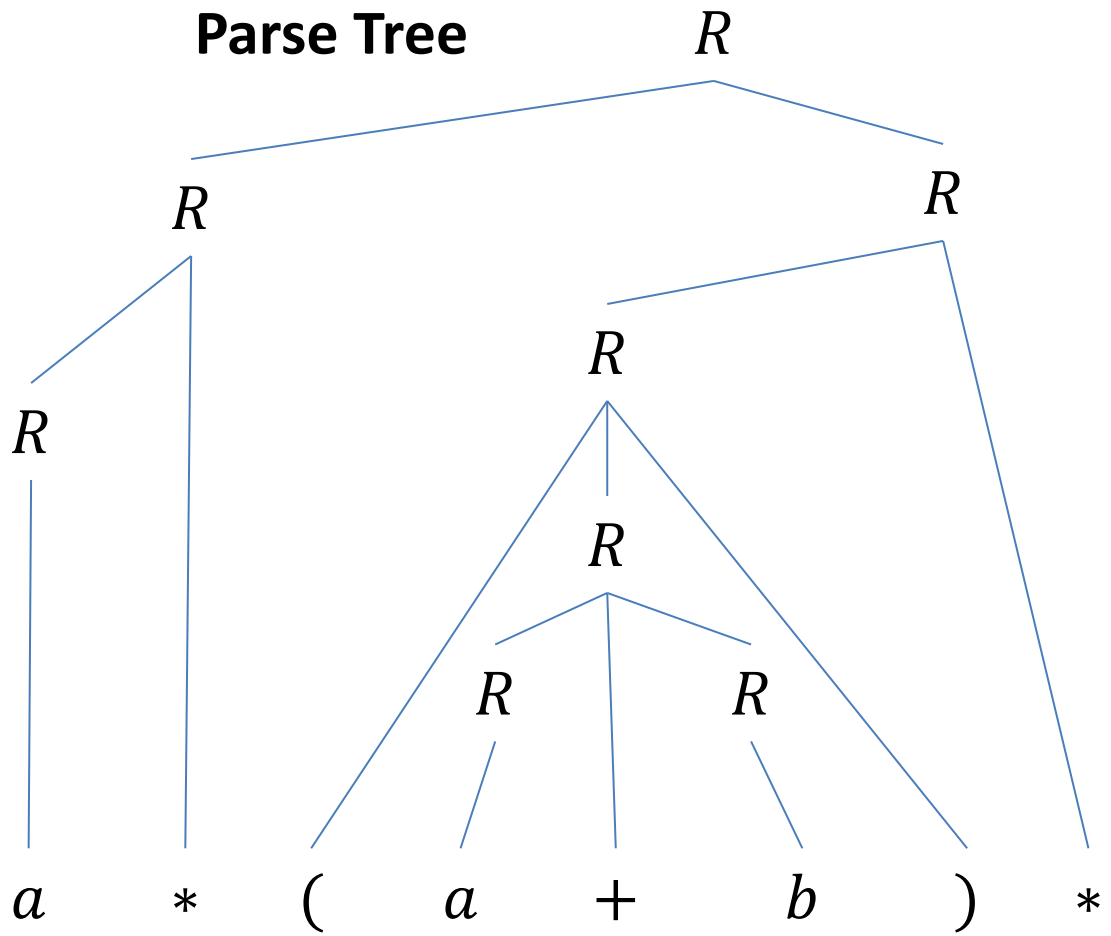
CFG for Regular Expressions

$$R \rightarrow a \mid b \mid \varepsilon \mid R + R \mid RR \mid R^* \mid (R)$$

To produce: $a^*(a+b)^*$

Current String	Production rule applied
R	$R \rightarrow RR$
RR	$R \rightarrow R^*$
R^*R	$R \rightarrow R^*$
R^*R^*	$R \rightarrow a$
a^*R^*	$R \rightarrow (R)$
$a^*(R)^*$	$R \rightarrow R + R$
$a^*(R+R)^*$	$R \rightarrow a$
$a^*(a+R)^*$	$R \rightarrow b$
$a^*(a+b)^*$	All terminals

Parse Tree



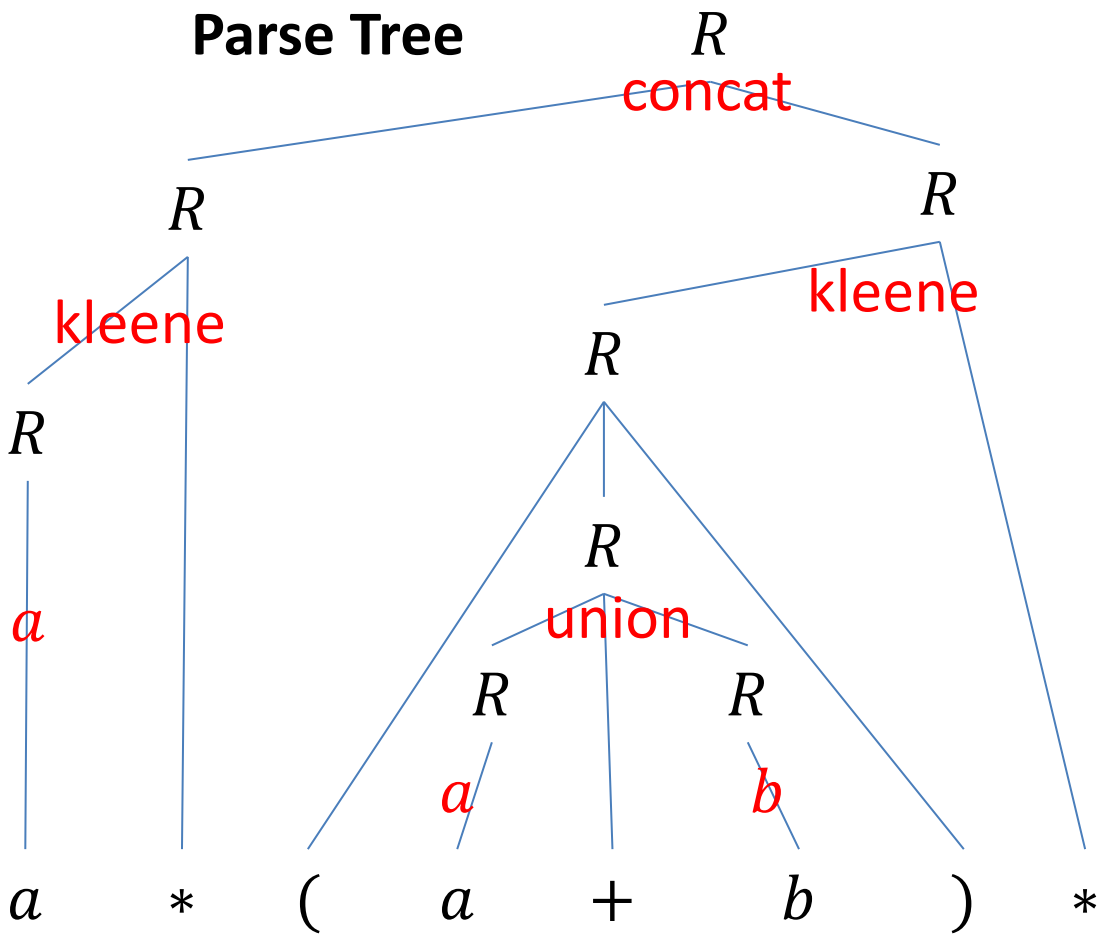
CFL Closure Redux

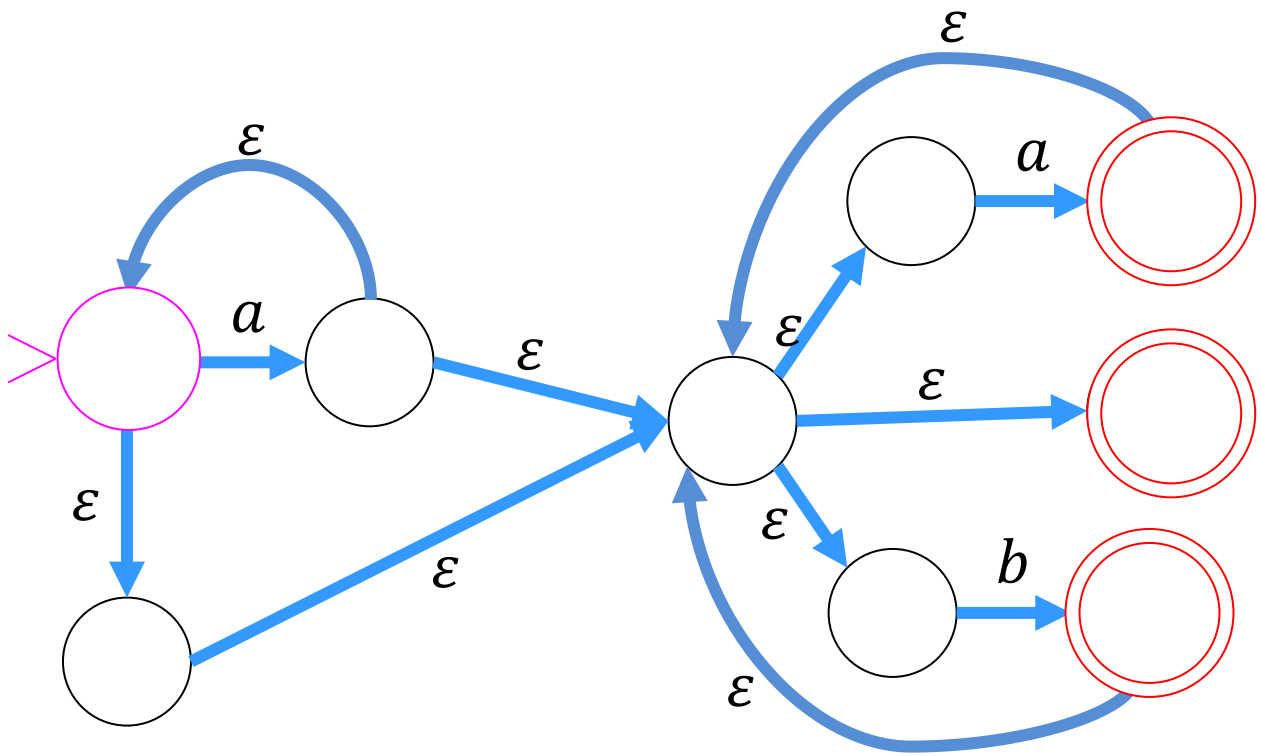
- We can also show closure of CFLs using CFGs
  - If we have CFGs for languages L_1, L_2 with start symbols S_1, S_2 respectively
- Union
  - Take production rules for both grammars, add $S \rightarrow S_1 | S_2$
- Concatenation
  - Take production rules for both grammars, add $S \rightarrow S_1 S_2$
- Kleene
  - Add to production rules of S_1, $S \rightarrow \varepsilon \mid SS \mid S_1$

Usefulness of Parse Tree

- Typically with CFGs, each production rule as some “meaning” associated
- E.g. in Regex, each rule was an operation
- From the parse tree, we know which operations to apply to which languages in what order to get the describe language

Regex parse tree to NFA



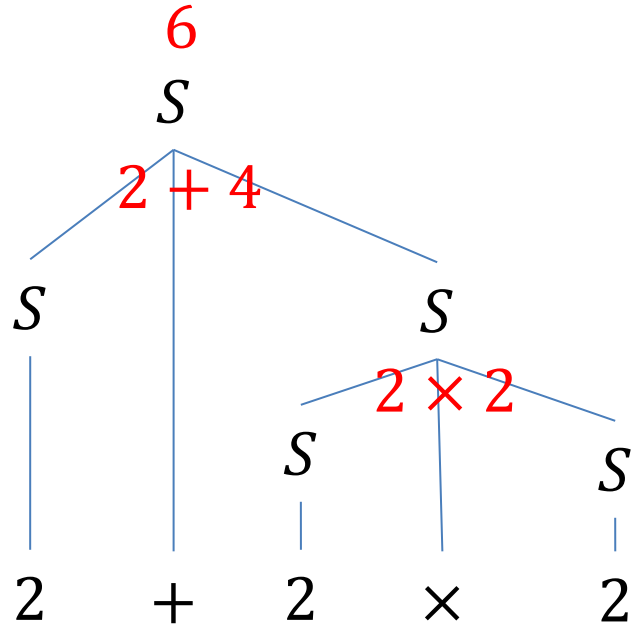


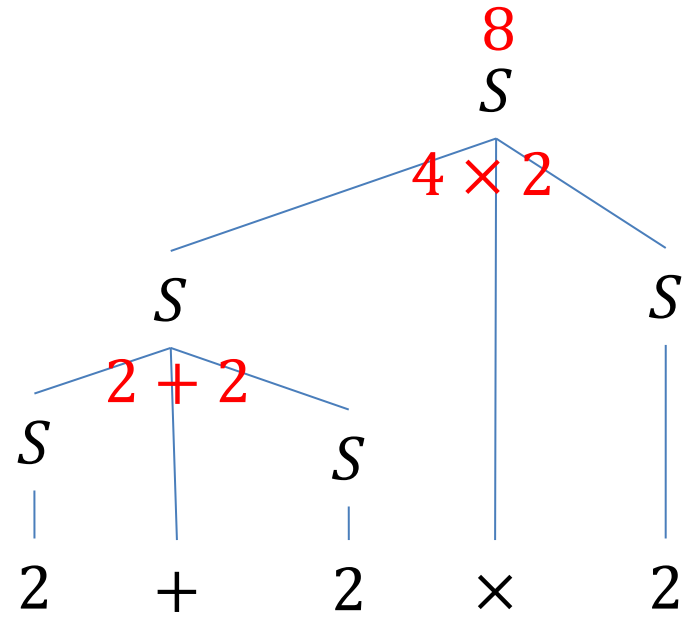
CFGs and Smug People on Social Media

- Consider this CFG for arithemetic expressions:

$$S \to S + S \mid S \times S \mid (S) \mid 2$$

How could we get $2 + 2 \times 2$?





Ambiguous Grammars

- A CFG is ambiguous if there is at least 1 string that can be generated by at least 2 different parse trees
- Generally less desirable (harder to assign meaning to each production rule)
- Some languages have no non-ambiguous CFGs
- Determining whether a CFG is ambiguous is not computable