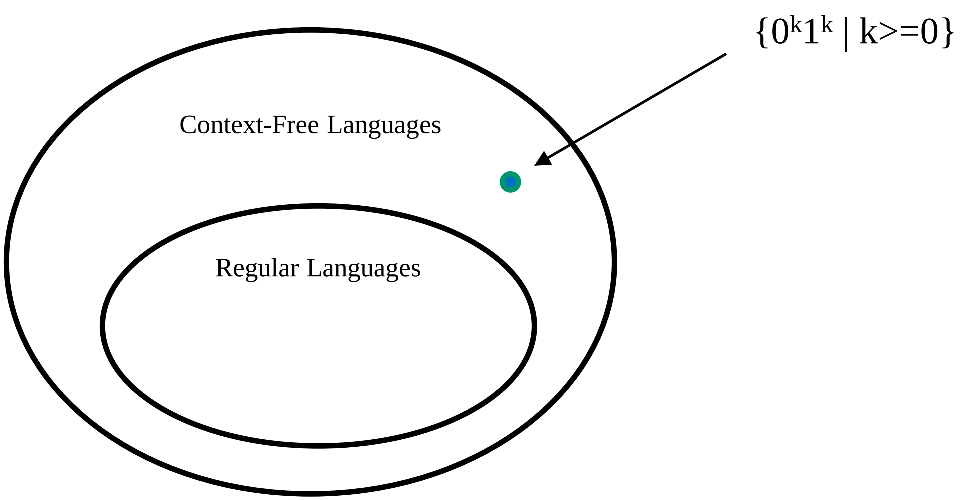
Context-Free Grammars and Languages

Reading: Chapter 5

Context-Free Languages

- The class of context-free languages generalizes the class of regular languages, i.e., every regular language is a context-free language.

- The reverse of this is not true,i.e., every context-free language is not necessarily regular. For example, as we will see $\{0^k1^k \mid k>=0\}$ is context-free but not regular.



Context-Free Languages

- Many issues and questions we asked for regular languages will be the same for context-free languages:

    Machine model – PDA (Push-Down Automata)

    Descriptor – CFG (Context-Free Grammar)

    Pumping lemma for context-free languages

    Closure of context-free languages with respect to various operations

    Algorithms and conditions for finiteness or emptiness

- Some analogies don't hold, e.g., non-determinism in a PDA makes a difference and, in particular, deterministic PDAs define a subset of the context-free languages.

- Informally a *Context-Free Language* (CFL) is a language generated by a *Context-Free Grammar* (CFG).

- What is a CFG?

- Informally, a CFG is a set of rules for deriving (or generating) strings (or sentences) in a language.

- **Example CFG:**

<sentence> –> <noun-phrase> <verb-phrase>	(1)
<noun-phrase> –> <proper-noun>	(2)
<noun-phrase> –> <determiner> <common-noun>	(3)
<proper-noun> –> John	(4)
<proper-noun> –> Jill	(5)
<common-noun> –> car	(6)
<common-noun> –> hamburger	(7)
<determiner> –> a	(8)
<determiner> –> the	(9)
<verb-phrase> –> <verb> <adverb>	(10)
<verb-phrase> –> <verb>	(11)
<verb> –> drives	(12)
<verb> –> eats	(13)
<adverb> –> slowly	(14)
<adverb> –> frequently	(15)

- **Example Derivation:**

<sentence>	=> <noun-phrase> <verb-phrase>	by (1)
	=> <proper-noun> <verb-phrase>	by (2)
	=> Jill <verb-phrase>	by (5)
	=> Jill <verb> <adverb>	by (10)
	=> Jill drives <adverb>	by (12)
	=> Jill drives frequently	by (15)

- Informally a CFG consists of:
  - A set of replacement rules.
  - Each will have a Left-Hand Side (LHS) and a Right-Hand Side (RHS).
  - Two types of symbols; *variables* and *terminals*.
  - LHS of each rule is a single variable (no terminals).
  - RHS of each rule consists of zero or more variables and terminals.
  - A “string” consists of only terminals.

Formal Definition of Context-Free Grammar

- A Context-Free Grammar (CFG) is a 4-tuple:

  G = (V, T, P, S)

  V - A finite set of variables or *non-terminals*

  T - A finite set of *terminals* (V and T do not intersect)

  P - A finite set of *productions*, each of the form A –> α, where A is in V and α is in (V U T)* // Note that α may be ε

  S - A starting non-terminal (S is in V)

- **Example CFG #1:**

G = ({A, B, C, S}, {a, b, c}, P, S)

P:

(1)	S –> ABC	
(2)	A –> aA	A –> aA | ε
(3)	A –> ε	
(4)	B –> bB	B –> bB | ε
(5)	B –> ε	
(6)	C –> cC	C –> cC | ε
(7)	C –> ε	

- **Example Derivations:**

S	=> ABC	(1)	S	=> ABC	(1)
	=> BC	(3)		=> aABC	(2)
	=> C	(5)		=> aaABC	(2)
	=> ε	(7)		=> aaBC	(3)
				=> aabBC	(4)
				=> aabC	(5)
				=> aabcC	(6)
				=> aabc	(7)

- Note that G generates the language a*b*c*

- **Example CFG #2:**

  G = ({S}, {0, 1}, P, S)

  P:

  (1) S –> 0S1 or just simply S –> 0S1 | ε

  (2) S –> ε

- **Example Derivations:**

  S => 0S1 (1)
  => 01 (2)

  S => ε (2)

  S => 0S1 (1)
  => 00S11 (1)
  => 000S111 (1)
  => 000111 (2)

- Note that G "generates" the language $\{0^k 1^k \mid k >= 0\}$

Formal Definitions for CFLs

- Let G = (V, T, P, S) be a CFG.

- **Definition:** Let X be in V, Y be in (V U T)*, X –> Y be in P, and let α and β be in (V U T)*. Then:

$$\alpha X\beta \Rightarrow \alpha Y\beta$$

In words, $\alpha X\beta$ *directly derives* $\alpha Y\beta$, or rather $\alpha Y\beta$ follows from $\alpha X\beta$ by the application of exactly one production from P.

- **Example:** (for grammar #1)

aaab**B**ccC => aaab**bB**ccC
aAb => ab
aAb => aaAb
aaAbBcccC => aaAbBccc
S => ABC

~~aAbBcC => abAbBcC~~
~~aAbbbC => aAbbbCB~~
~~S => aaabbbc~~

S -> ABC
A –> aA | ε
B –> bB | ε
C –> cC | ε

- **Definition:** Suppose that $\alpha_1, \alpha_2, \ldots, \alpha_m$ are in (V U T)*, m>=1, and

$$\alpha_1 \Rightarrow \alpha_2$$
$$\alpha_2 \Rightarrow \alpha_3$$
$$\vdots$$
$$\alpha_{m-1} \Rightarrow \alpha_m$$

Then $\alpha_1 \Rightarrow^* \alpha_m$

In words, α_1 *derives* α_m, or rather, α_m follows from α_1 by the application of *zero or more* productions. Note that: $\alpha \Rightarrow^* \alpha$.

- **Example:** (for grammar #1)

S -> ABC
A –> aA | ε
B –> bB | ε
C –> cC | ε

aAbBcC =>* aaabbccccC
aAbBcC =>* abBc
S =>* aabbbc

~~aAbBcC =>* bac~~
~~aabbccc =>* aAbBc~~
~~S=>* CaAB~~

- **Definition:** Let α be in (V U T)*. Then α is a *sentential form* if and only if $S \Rightarrow^* \alpha$.

- **Definition:** Let G = (V, T, P, S) be a context-free grammar. Then the *language generated* by G, denoted L(G), is the set:

  {w | w is in T* and S=>* w}

- **Definition:** Let L be a language. Then L is a *context-free language* if and only if there exists a context-free grammar G such that L = L(G).

- **Definition:** Let G_1 and G_2 be context-free grammars. Then G_1 and G_2 are *equivalent* if and only if $L(G_1) = L(G_2)$.

- **Observations:** (we won't use these, but food for thought…)

  –> forms a relation on V and (V U T)*

  => forms a relation on (V U T)* and (V U T)*.

  =>* forms a relation on (V U T)* and (V U T)*.

- **Exercise:** Give a CFG that generates the set of all strings of 0's and 1's that contain the substring 010.

- **Exercise:** Give a CFG that generates the set of all strings of *a*'s, *b*'s and *c*'s where every *a* is immediately followed by a *b*.

- **Exercise:** Give a CFG that generates the set of all strings of 0's and 1's that contain an even number of 0's.

- **Note –** as with the states in a DFA, non-terminals in a CFG have "assertions" associated with them.

- **Question:** Is the following a valid CFG?

  S -> 0A
  A -> 1B
  B -> 0S1

- **Keep in mind the smaller, "toolkit" grammers:**

  S->0S|e $\quad$ 0*

  S->0S1|e $\quad$ 0^n1^n

  S->AB $\quad$ *Something from A followed by something from B*

  A->aA|e, B->bB|e $\quad$ a*b*

  S->AS|e $\quad$ *Zero or more occurrences of something from A*

  A->0A1|e $\quad$ $(0^n1^n)^*$

- **Sometimes it's helpful to start with a simpler language, and then modify the grammar:**

  $0^i1^j, \quad j \geq i \geq 0$

So what is the relationship between the regular and context-free languages?

- **Theorem:** Let L be a regular language. Then L is a context-free language.

- **Proof:** (by induction)

  We will prove that if r is a regular expression then there exists a CFG G such that L(r) = L(G). The proof will be by induction on the number of operators in r.

- **Basis:** Op(r) = 0

Then r is either Ø, ε, or **a**, for some symbol **a** in Σ.

For Ø:

Let G = ({S}, {}, P, S) where P = {}

For ε:

Let G = ({S}, {}, P, S) where P = {S –> ε}

For **a**:

Let G = ({S}, {a}, P, S) where P = {S –> **a**}

Inductive Hypothesis:

Suppose there exists a $k>=0$ such that for any regular expression r, where $0<=op(r)<=k$, that there exists a CFG G such that $L(r) = L(G)$.

Inductive Step:

Let r be a regular expression with $op(r)=k+1$. Since $k>=0$, it follows that $k+1>=1$, i.e., r has at least one operator. Therefore $r = r_1 + r_2$, $r = r_1r_2$ or $r = r_1*$.

Case 1) $r = r_1 + r_2$

Since r has k+1 operators, one of which is +, it follows that $0<=op(r_1)<=k$ and $0<=op(r_2)<=k$.

From the inductive hypothesis it follows that there exist CFGs $G_1 = (V_1, T_1, P_1, S_1)$ and $G_2 = (V_2, T_2, P_2, S_2)$ such that $L(r_1) = L(G_1)$ and $L(r_2) = L(G_2)$.

Assume without loss of generality that V_1 and V_2 have no non-terminals in common, and construct a grammar $G = (V, T, P, S)$ where:

$V = V_1 \text{ U } V_2 \text{ U } \{S\}$
$T = T_1 \text{ U } T_2$
$P = P_1 \text{ U } P_2 \text{ U } \{S \text{ –> } S_1, S \text{ –> } S_2\}$

Clearly, $L(r) = L(G)$.

Case 2) $r = r_1 r_2$

Let $G_1 = (V_1, T_1, P_1, S_1)$ and $G_2 = (V_2, T_2, P_2, S_2)$ be as in Case 1, and construct a grammar $G = (V, T, P, S)$ where:

$V = V_1 \cup V_2 \cup \{S\}$
$T = T_1 \cup T_2$
$P = P_1 \cup P_2 \cup \{S \rightarrow S_1S_2\}$

Clearly, $L(r) = L(G)$.

Case 3) $r = (r_1)^*$

Let $G_1 = (V_1, T_1, P_1, S_1)$ be a CFG such that $L(r_1) = L(G_1)$ and construct a grammar $G = (V, T, P, S)$ where:

$V = V_1 \cup \{S\}$
$T = T_1$
$P = P_1 \cup \{S \rightarrow S_1S, S \rightarrow \varepsilon\}$

Clearly, $L(r) = L(G)$.•

- The preceding theorem is constructive, in the sense that it shows how to construct a CFG from a given regular expression.

- **Example #1:**

  $r = a^*b^*$

  $r = r_1r_2$

  $r_1 = r_3^*$

  $r_3 = a$

  $r_2 = r_4^*$

  $r_4 = b$

- **Example #1:** a*b*

$r_4 = b$	$S_1 \to b$
$r_3 = a$	$S_2 \to a$
$r_2 = r_4^*$	$S_3 \to S_1S_3$ $S_3 \to \varepsilon$
$r_1 = r_3^*$	$S_4 \to S_2S_4$ $S_4 \to \varepsilon$
$r = r_1r_2$	$S_5 \to S_4S_3$

- **Example #2:**

$r = (0+1)^*01$

$r = r_1 r_2$

$r_1 = r_3^*$

$r_3 = (r_4 + r_5)$

$r_4 = 0$

$r_5 = 1$

$r_2 = r_6 r_7$

$r_6 = 0$

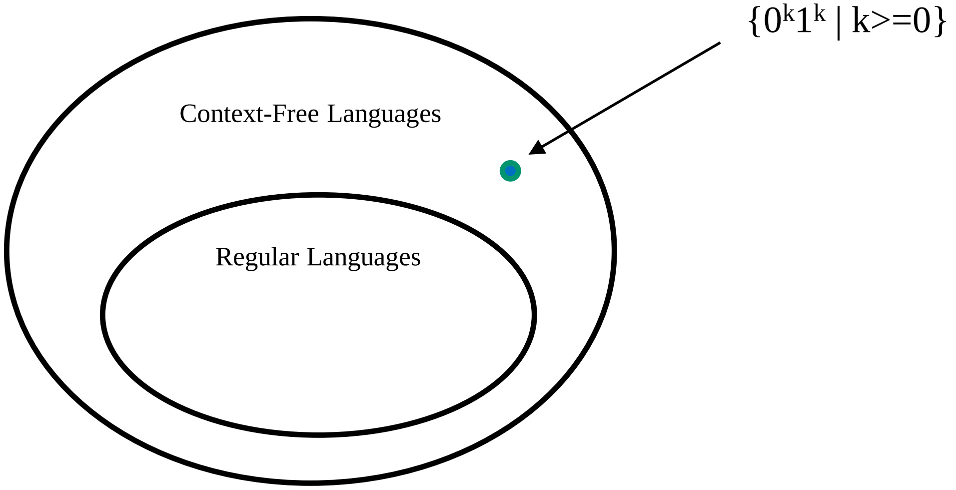
$r_7 = 1$

- **Example #2:** (0+1)*01

$r_7 = 1$	$S_1 \rightarrow 1$
$r_6 = 0$	$S_2 \rightarrow 0$
$r_2 = r_6 r_7$	$S_3 \rightarrow S_2 S_1$
$r_5 = 1$	$S_4 \rightarrow 1$
$r_4 = 0$	$S_5 \rightarrow 0$
$r_3 = (r_4 + r_5)$	$S_6 \rightarrow S_4$, $S_6 \rightarrow S_5$
$r_1 = r_3^*$	$S_7 \rightarrow S_6 S_7$ $S_7 \rightarrow \varepsilon$
$r = r_1 r_2$	$S_8 \rightarrow S_7 S_3$

- **Note:** Although every regular language is a CFL, the reverse is not true. In other words, there exist CFLs that are not regular languages, i.e., $\{0^n1^n \mid n>=0\}$.

  => Therefore the regular languages form a proper subset of the CFLs.



- By the way, note that it is usually very easy to construct a CFG for a given regular expression, even without using the previous technique.

- **Examples:**
  - 1(0+01)*0
  - (0+1)*0(0+1)*0(0+1)*

- **Definition:** A CFG is a regular grammar if each rule is of the following form:

  - A –> a <non-terminal> -> terminal-symbol
  - A –> aB <non-terminal> -> terminal-symbol <non-terminal>
  - A –> ε <non-terminal> -> epsilon

  where A and B are in V, and a is in T

- **Regular Grammar:**

  S –> aS | ε

  S –> aB

  B –> bB

  B –> b

- **Non-Regular Grammar:**

  S –> 0S1 | ε

- **Theorem:** A language L is a regular language iff there exists a regular grammar G such that L = L(G).

- **Proof:** Exercise.•

- **Observation:** A language may have several CFGs, some regular, some not
  - Recall that S –> 0S1 | ε is not a regular grammar.
  - The fact that this grammar is not regular does not in and of itself prove that 0^n1^n is not a regular language.
  - Similarly S -> S0 | ε is not a regular grammar.

Derivation Trees

- **Definition:** Let G = (V, T, P, S) be a CFG. A tree is a derivation (or parse) tree if:

  - Every vertex has a label from V U T U {ε}
  - The label of the root is S
  - If a vertex with label A has children with labels X_1, X_2,…, X_n, from left to right, then

    $$A \rightarrow X_1, X_2, \ldots, X_n$$

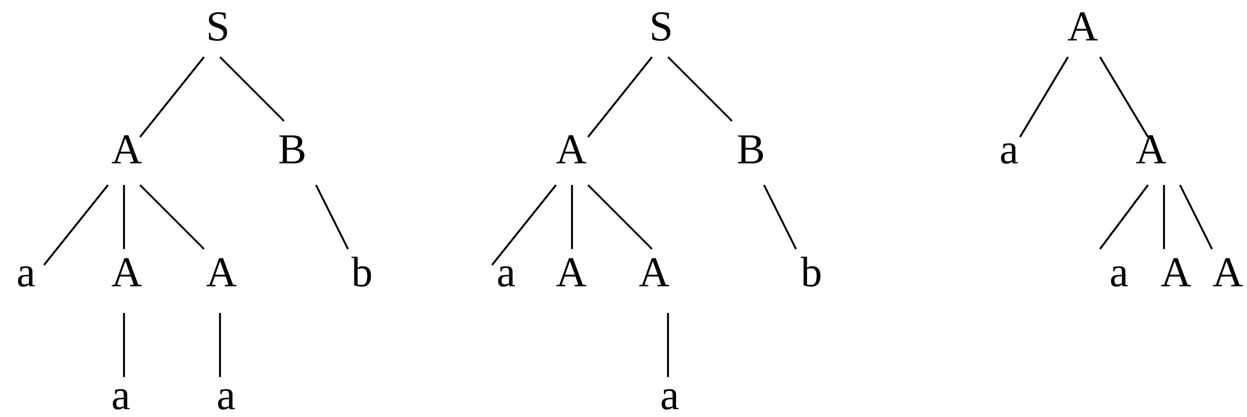
    must be a production in P
  - If a vertex has label from T, then that vertex is a leaf
  - If a vertex has label ε, then that vertex is a leaf and the only child of its' parent

- More Generally, a derivation tree can be defined with any non-terminal as the root.

- A derivation tree is basically another way of conveying a (part of a) derivation.

- **Example:**

S –> AB
A –> aAA
A –> aA
A –> a
B –> bB
B –> b

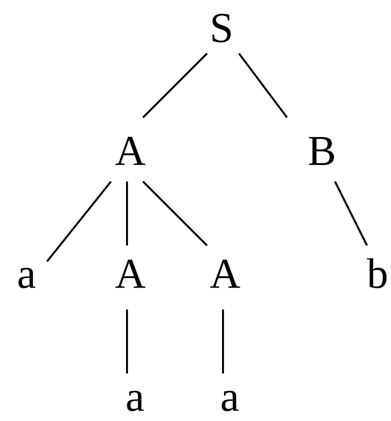


yield = aaab   yield = aAab   yield = aaAA

- **However:**
  - Root can be any non-terminal
  - Leaf nodes can be terminals or non-terminals
  - A derivation tree with root S shows the productions used to obtain a sentential form

- **Observation:** Every derivation corresponds to <u>one</u> derivation tree.

S => AB
=> aAAB
=> aaAB
=> aaaB
=> aaab

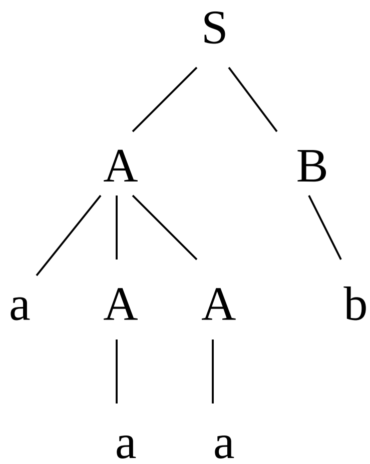


- **Observation:** Every derivation tree corresponds to <u>one or more</u> derivations.

S => AB	S => AB	S => AB
=> aAAB	=> Ab	=> Ab
=> aaAB	=> aAAb	=> aAAb
=> aaaB	=>aAab	=> aaAb
=> aaab	=> aaab	=> aaab

- **Definition:** A derivation is *leftmost (rightmost)* if at each step in the derivation a production is applied to the leftmost (rightmost) non-terminal in the sentential form.
  - The first derivation above is leftmost, second is rightmost, the third is neither.

- **Observation:** Every derivation tree for a string x in L(G) corresponds to <u>exactly one</u> leftmost (and rightmost) derivation.



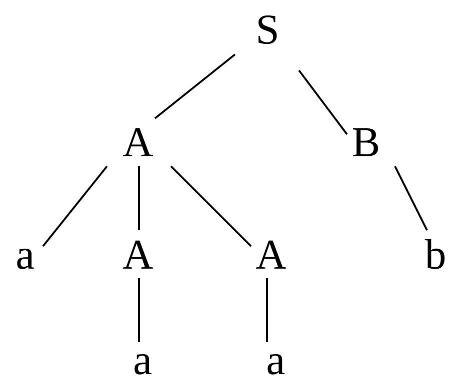
S => AB
=> aAAB
=> aaAB
=> aaaB
=> aaab

- **Observation:** Let G be a CFG. Then there <u>may</u> exist a string x in L(G) that has more than 1 leftmost (or rightmost) derivation. Such a string will also have more than 1 derivation tree.

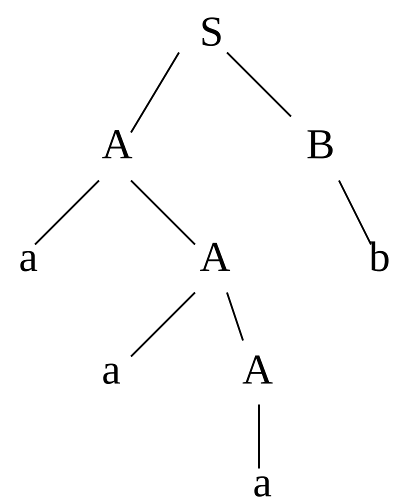
- **Example:** Consider the string aaab and the preceding grammar.

S –> AB
A –> aAA
A –> aA
A –> a
B –> bB
B –> b

S => AB
=> aAAB
=> aaAB
=> aaaB
=> aaab



S => AB
=> aAB
=> aaAB
=> aaaB
=> aaab



- The string has two left-most derivations, and therefore has two distinct parse trees.

- **Definition:** Let G be a CFG. Then G is said to be <u>ambiguous</u> if there exists an x in L(G) with >1 leftmost derivations.

- Equivalently, G is ambiguous if there exists an x in L(G) with >1 rightmost derivations.

- Equivalently, G is ambiguous if there exists an x in L(G) with >1 parse trees.

"So," the rabbit asked the frog, *"why is ambiguity such a bad thing?"*

- Consider the following CFG, and the string 3+4*5:

E -> E + E
E -> E * E
E -> (E)
E -> number

- A parsing algorithm is based on a grammar.

- The parse tree generated by a parsing algorithm determines how the algorithm interprets the string.

- If the grammar allows the algorithm to parse a string in more than one way, then that string could be interpreted in more than one way…not good!

* *In other words, the grammar should be designed so that it dictates exactly one way to parse a given string.*

“Oh, now I understand,” said the rabbit…

- And there is some good news!

- **Observation:** Given a CFL L, there may be more than one CFG G with L = L(G). Some ambiguous and some not.

- For example, a non-ambiguous version of the previous grammar:

  E -> T | E+T
  T -> F | T*F
  F -> (E) | number

- Note that 3+4*5 has exactly one leftmost derivation, and hence, parse tree.

"So from this day forward, I will only write non-ambiguous CFGs," said the rabbit…

"But there is just one more problem," said the frog…

- **Definition:** Let L be a CFL. If every CFG G with L = L(G) is ambiguous, then L is inherently ambiguous.

- And yes, there do exist inherently ambiguous languages…

$$\{a^n b^n c^m d^m \mid n>=1, m>=1\} \cup \{a^n b^m c^m d^n \mid n>=1, m>=1\}$$

"Oh, s@#t!," said the rabbit…

"Don't worry," said the frog…

"Inherently ambiguous CFLs hide deep in the forest, so you won't see them very often."

- Exercise – try writing a grammar for the above language, and see how any string of the form $a^n b^n c^n d^n$ has more than one leftmost derivation.

- Many, potential algorithmic problems exist for context-free grammars.

- Imagine developing algorithms for each of the following problems:

  - Is L(G) empty?
  - Is L(G) finite?
  - Is L(G) infinite?
  - Is L(G) = T*
  - Is $L(G_1) = L(G_2)$?
  - Is G ambiguous?
  - Is L(G) inherently ambiguous?
  - Given ambiguous G, construct unambiguous G' such that L(G) = L(G')
  - Given G, is G "minimal?"

- Most of the above problems are "undecidable," i.e., there is no algorithm, or they are computation difficult, i.e. NP-hard or PSPACE-hard.

S -> A

A -> S

B -> b