Context-Free Grammars:
- Terminal Vocabulary: (Terminals, Vt)
- Intermediate symbols: (non-terminals, variables, Vn)
- Start symbol (non-terminal) S
- Productions of the form $A \rightarrow X_1 X_2 ... X_m$
  - $A$ is a non-terminal
  - $X_1$ is a terminal or a non-terminal
  - valid: $A \rightarrow$

- Start with $S$ and rewrite using productions until all terminals
  - Derivation: $A \Rightarrow U_1...U_n \Rightarrow ... \Rightarrow a_1 ... a_m$
  - Sentential form: Any "string" along the derivation
  - Sentence: final form with no non-terminals
Grammars (page 2)

Conventions:
- $a, b, c \rightarrow V_t$
- $A, B, C \rightarrow V_n$
- $U, V, W \rightarrow V$ $(V_t + V_n)$
- $\text{Alpha, Beta, Gamma} \rightarrow V^*$
- $u, v, w \rightarrow V_t^*$

Leftmost derivation:
- $A \Rightarrow_{lm} X_1 X_2 \ldots X_m \Rightarrow_{lm} Y_1 \ldots Y_n X_2 \ldots X_m$
- All steps replace the left most non-terminal

Rightmost derivation
- $X_1 X_2 \ldots X_m \Rightarrow_{rm} X_1 \ldots X_{m-1} Y_1 \ldots Y_n$
- All steps replace the right most non-terminal
Example grammar 1

\[
\begin{align*}
S & \rightarrow a \ S \ b \\
S & \rightarrow a \ b \\
\end{align*}
\]

- Also written as: \( S \rightarrow a \ S \ b \mid a \ b \)
- Spaces not in final sentence
- Derivation: \( S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb \)
- Notice, this is both leftmost and rightmost

Example grammar 2

\[
\begin{align*}
S & \rightarrow AB \\
A & \rightarrow aAa \mid c \\
B & \rightarrow bBb \mid c \\
\end{align*}
\]

- Consider the word aacaabcb
  - Leftmost: \( S \Rightarrow AB \Rightarrow aAaB \Rightarrow aaAaaB \Rightarrow aacaabBb \Rightarrow aacaabcb \)
  - Rightmost: \( S \Rightarrow AB \Rightarrow AbBb \Rightarrow Abcb \Rightarrow aAabcb \Rightarrow aaAabcb \Rightarrow aacaabcb \)

Both grammars have the "self embedding property". It allows "primitive counting"
Parse tree:

A parse tree ... tree form of derivation
☐ A parse tree many have lots of derivations.
☐ Each parse tree has one leftmost derivation
☐ Each parse tree has one rightmost derivation
☐ Some sentences have more than one parse tree
  ☐ Grammars like this are ambiguous .... BAD grammar!
  ☐ Also have more than one leftmost and rightmost derivations.

Language:
☐ given grammar G, L(G) is the "language" generated by G.
☐ L(G) is the set of all possible terminal strings w where S =>* w.

☐ L(G) of S -> a S b | a b?
  ☐ a^n b^n | n >= 1

☐ L(G) of S -> AB, A -> aAa | c, B -> bBb | c
  ☐ a^n c a^n b^m c b^m | n, m >= 1
What is a PARSE?

Given a grammar and a string $x$, answer the question(s)

- Is $x$ a member of $L(G)$?
- If so, what is its parse tree?

Grammar is a generator but we want a recognizer. Such must exist!
(Or as the book says "a notational mechanism for specifying languages"

There is not one unique grammar for each language.
- A grammar may contain "useless symbols"
- A grammar may allow multiple, distinct derivations for some sentence
- A grammar may not generate the language you wanted to generate
Example grammar 3

\[ S \rightarrow aS \mid B \]
\[ B \rightarrow bB \mid b \]

- Language?
- \( a^*b^* \)
- Consider aabbb
- Derivation: \( S \Rightarrow aS \Rightarrow aaS \Rightarrow aaB \Rightarrow aabB \Rightarrow aabbB \Rightarrow aabbb \)
- This is both leftmost and rightmost.

- This grammar is actually a "regular grammar"
  - All productions of the form \( A \rightarrow a \) or \( A \rightarrow aB \)
  - "proof" variables are DFA states.
  - \( A \rightarrow a \) transition to a final state, reading a
  - \( A \rightarrow aB \) transition to another state, reading a

Other grammars?
- Context Sensitive:
  - \( \alpha A \beta \rightarrow \alpha \delta \beta \)
  - Not very useful for programming languages ... recognition is hard
General idea of a parser or recognizer:
- Given $G$, we need something that will answer the question:
  - Given input $x$, is $x$ a member of $L(G)$?
- For language translation, we need the parse tree also!

Consider a "Push Down Automata" aka PDA
- $M = (Q, T, S, D, q_0, Z, F)$
  - $Q$ is a set of states, $T$ is a set of "terminals"
  - $S$ is the stack alphabet
  - $D$ is a transition function
  - $q_0$ (a member of $Q$) is the start state
  - $Z$ (a member of $S$) is the initial stack symbol
  - $F$ (a subset of $Q$) are the final states
- $D(Q,T$ or epsilon,$S) \rightarrow Q \times S^*$

- $M$ can accept via "final state" or "empty stack"
- Note, as soon as the stack is empty, the machine halts
- This machine takes as input $X$ and says Yes or No.
- This machine is going to the basis for all our recognizers
A PDA for $a^n b a^n$

- $Q = \{q_0, q_1, q_2\}$, $T = \{a, b\}$
- $S = \{B, a\}$, $Z = B$, $F = \{q_2\}$
- $D$: the delta function
  - $D(q_0, a, B) \rightarrow q_0, aB$
  - $D(q_0, a, a) \rightarrow q_0, aa$
  - $D(q_0, b, B) \rightarrow q_2$
  - $D(q_0, b, a) \rightarrow q_1, a$
  - $D(q_1, a, a) \rightarrow q_1$
  - $D(q_1, B) \rightarrow q_2$

While we won’t do them here, there are theorems like:
- Acceptance by final state or empty stack are equivalent
- PDAs and CFGs are equivalent for language definition.