Today's learning goals  Sipser Ch 1.4, 2.1

- Context Free Grammars (CFG)
- Parse trees and ambiguity
- CFG Design
- Push Down Automata
# Arithmetic Expressions

- \((3+12)*(5+7+(5*63^2))\)
- **Grammar rules:**
  
  \[ S \rightarrow S+S \mid S*S \mid S^N \mid N \mid (S) \]
  
  \[ N \rightarrow D \mid DN \]
  
  \[ D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \]

- **Derivations**
  
  - \( S \rightarrow DN \rightarrow 1N \rightarrow 1DN \rightarrow 12N \rightarrow 12D \rightarrow 123 \)
  
  - \( S \rightarrow S+S \rightarrow 1+S \rightarrow 1+S*S \rightarrow 1+2*S \rightarrow 1+2*3 \)
Parse trees and Leftmost derivations

- $1 + 2 \times 3$
- $S \rightarrow S + S \rightarrow S + S \cdot S \rightarrow * S + 2 \times 3 \rightarrow * 1 + 2 \times 3$
- $S \rightarrow S + S \rightarrow * 1 + S \rightarrow 1 + S \cdot S \rightarrow * 1 + 2 \times 3$ (leftmost)
Parse trees and Leftmost derivations

- $1 + 2 \times 3$
- $S \rightarrow S + S \rightarrow S + S \times S \rightarrow S + 2 \times 3 \rightarrow 1 + 2 \times 3$
- $S \rightarrow S + S \rightarrow 1 + S \rightarrow 1 + S \times S \rightarrow 1 + 2 \times 3$ (leftmost)

- For each derivation, there is a unique parse tree.
- For each parse tree, there is a unique leftmost derivation.
Ambiguity

- $S \rightarrow S+S \rightarrow S+S+S \rightarrow 1+2+3$
- $S \rightarrow S+S \rightarrow 1+S \rightarrow 1+S+S \rightarrow 1+2+3$
Ambiguity

- $S \rightarrow S + S \rightarrow S + S + S \rightarrow 1 + 2 + 3$
- $S \rightarrow S + S \rightarrow 1 + S \rightarrow 1 + S + S \rightarrow 1 + 2 + 3$

$1 + (2 + 3)$

$(1 + 2) + 3$
Ambiguity

- \[ S \rightarrow S + S \rightarrow S + S + S \rightarrow 1 + 2 + 3 \]
- \[ S \rightarrow S + S \rightarrow 1 + S \rightarrow 1 + S + S \rightarrow 1 + 2 + 3 \]

\[(1 + 2) * 3 = 9\]

\[1 + (2 * 3) = 7\]
Unambiguous Grammars

- A CFG $G$ is unambiguous if every $w$ in $L(G)$ has a unique parse tree (or, equivalently, a unique leftmost derivation)

- A CFG is ambiguous if there exists a $w$ in $L(G)$ such that $w$ has two different parse trees (or, equivalently, two different leftmost derivations)
Test time

• Is the following grammar ambiguous?

\[ S \rightarrow S+S \mid S*S \mid 0 \mid 1 \]

A) Yes
B) No
C) It depends on the input string
D) There is no answer because the question is ambiguous
E) I don’t know
Unambiguous Grammar for Exp

- \( G = (\{E,T,F\}, \{0,...,9,+,*[^{\text{character}},(),}\}, R, E) \) where \( R \) is the set of rules:
  
  \[
  \begin{align*}
  E & \rightarrow E + T | T \\
  T & \rightarrow T * F | F \\
  F & \rightarrow F ^ N | N | (E) \\
  N & \rightarrow 0 | ... | 9
  \end{align*}
  \]

- Parse/Evaluate
  
  \[1+2*3^2+5\]
Recap: Context-free languages

**Context-free grammar**

\[ G = (V \text{ finite set of variables}, \Sigma \text{ finite set of terminals}, R \text{ finite set of rules}, S \text{ start variable}) \]

**One-step derivation**

\[ uAv \implies uwv \quad \text{where } u, v, w \in (\Sigma \cup V)^* \quad A \rightarrow w \in R \]

**Derivation**

\[ u \xrightarrow{*} v \quad u = v \text{ or } u \implies u_1 \implies \cdots \implies u_k \implies v \]

**Language generated by grammar**

\[ L(G) = \{ w \in \Sigma^* | S \xrightarrow{*} w \} \]
CFGs and Automata

- L is a context free language (CFL) if L=L(G) for some CFG G
- What is the relation between CFL and regular languages?
  - Can any DFA/Regex be transformed into an equivalent CFG?
  - Not all CFL can be recognized by a DFA/NFA!
  - How can we extend DFAs/NFAs to make them as powerful as CFGs?
From Regex to CFG

- Approach via closure properties:
  - Show that \{a\}, \{\varepsilon\} and {} are context free languages
  - Show that the class of context free languages is closed under union, concatenation and star
  - See Haskell 3 for details

- It follows that for any regular expression E, there is a CFG for L(E)
From DFA to CFG

• **Claim:** For any DFA $M = (Q, \Sigma, \delta, s, F)$ there is a CFG $G = (V, \Sigma, R, S)$ such that $L(M) = L(G)$

• **Proof:**

  Idea: trace computation using variables to denote state
  - $V = Q$
  - $S = s$
  - $R = \{ q \rightarrow a \delta(q, a) \mid q \in Q, a \in \Sigma \} \cup \{ q \rightarrow \varepsilon \mid q \in F \}$
Regular languages vs. CFL

Context-free languages

Regular languages
An alternative ... 

- NFA + stack

Sipser p. 109

State control (NFA)

Read tape head

Push/pop to stack

Stack

Input
Pushdown automata

- NFA + stack

At each step
1. **Transition** to new state based on current state, letter read, and top letter of stack.
2. (Possibly) **push or pop** a letter to (or from) top of stack.
Pushdown automata

• NFA + stack

Accept a string if there is some sequence of states and some sequence of stack contents which processes the entire input string and ends in an accepting state.
State diagram for PDA

If hand-drawn or in Sipser
State transition labelled $a, b \rightarrow c$ means
"when machine reads an $a$ from the input and the top symbol of the stack is a $b$, it may replace the $b$ with a $c$.”

In JFLAP: use $;$ instead of $\rightarrow$
If hand-drawn or in Sipser

State transition labelled $a, b \rightarrow c$ means

"when machine reads an $a$ from the input and the top symbol of the stack is $a \ b$, it may replace the $b$ with a $c$.

What edge label would indicate “Read a 0, don’t pop anything from stack, don’t push anything to the stack”?

A) 0, $\varepsilon \rightarrow \varepsilon$
B) $\varepsilon$, 0→$\varepsilon$
C) $\varepsilon, \varepsilon \rightarrow 0$
D) $\varepsilon \rightarrow \varepsilon, 0$
E) I don’t know
Useful trick

- What would $\varepsilon, \varepsilon \rightarrow \$ \text{ mean?}$

  A) Without reading any input or popping any symbol from stack.
  B) If the input stack is empty, push $\$.$
  C) At the end of reading the input.
  D) I don’t know.

Why is this useful?

Commonly used from initial state (at start of computation) to record bottom of stack with a special symbol. .... Useful to check if stack becomes empty again.
Formal definition of PDA  

A PDA is a 6-tuple \((Q, \Sigma, \Gamma, \delta, q_0, F)\) where \(Q, \Sigma, \Gamma, F\) are all finite sets and

1. \(Q\) is the set of states
2. \(\Sigma\) is the input alphabet
3. \(\Gamma\) is the stack alphabet
4. \(\delta : Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)\) is the transition function
5. \(q_0 \in Q\) is the start state
6. \(F \subseteq Q\) is the set of accept states.
Designing a PDA

\[ L = \{ 0^i 1^{i+1} \mid i \geq 0 \} \]

*Informal description of PDA:*

Read symbols from the input. As each 0 is read, push it onto the stack. As soon as 1s are seen, pop a 0 off the stack for each 1 read. If the stack becomes empty and there is exactly one 1 left to read, read that 1 and accept the input. If the stack becomes empty and there are either zero or more than one 1s left to read, or if the 1s are finished while the stack still contains 0s, or if any 0s appear in the input following 1s, reject the input.
Designing/Tracing a PDA

$L = \{ 0^i 1^{i+1} \mid i \geq 0 \}$
PDAs and CFGs are equivalently expressive

**Theorem 2.20**: A language is context-free if and only some nondeterministic PDA recognizes it.

**Consequences**
- Quick proof that every regular language is context free
- To prove closure of class of CFLs under a given operation, can choose two modes of proof (via CFGs or PDAs) depending on which is easier
For next time

- Read Sipser 2.1, 2.2
- Design a few CFGs and PDAs on your own