Exercise C1

Part 1: Your Own Problems

Part 2: Grammar and TM Variants

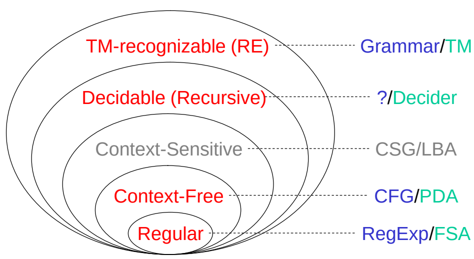
A. CFGs without empty productions
B. CFGs with rules such that |RHS| ≤ 2
C. TMs with 2 stacks
D. TMs with a queue

Unit C2: Overview

- Analyze a realistic example involving a mini language
- Understand CFGs/CFLs
  – Grammars, languages
- Understand how to process CFLs
- Understand the effect of determinism
- Preview Exercise C2 "Context-Free"

Review

Extended Chomsky Hierarchy



Example grammars?
Essential properties?

Demo

Graphing Tool

- Basic functions
  – Constant: e.g., **1** (integers) and **2.34** (**.5** must be written as **0.5**)
  – Factorial: only the fixed form of **n!**
  – Logarithm: **log** *OptBase NonConstantExp* (*OptBase*: const)
  – Power: *Exp*^*Exp* where *Exp* is a valid expression. *Exp* other than a constant or **n** requires parentheses.
- Complex expression
  – Complex expressions can be formed by using the operators *****, **/**, **+**, **-**, and parentheses **(** and **)**. ***** and **/** takes precedence over **+** and **-**. The operators are left associative.
- Notes
  – Spaces are ignored internally.
  – **sqrt(n)** must be entered as **n^0.5**.

How to specify?
How to process?

`http://www.tcnj.edu/~komagata/Graphing`

Group Exercise 1

- How would you implement the module to analyze the input expressions?
  – How to specify?
  – How to process?

Context-Free Grammar (CFG)

- **G** = (N, T, R, S)
  – N: finite set of nonterminals [upper case]
  – T: finite set of terminals [lower case]
  – R: finite set of rules $A \rightarrow \alpha$ where α is a string made up of the elements of N and T
  – S: start symbol $\in N$

Formal definition for a CFG for graphing tool?

Context-Free Language (CFL)

- Derivation (generation) of a string in a CFG, **G** (from the start symbol S)
  - $S \rightarrow^* w$ (i.e., zero or more rule application)
  - Also said: "**G** generates w" or "**G** accepts w"
- $L(\mathbf{G}) = \{w \mid S \rightarrow^* w$, a string made up of the elements of $\mathrm{T}\}$
- CFL = $\{L(\mathbf{G}) \mid \mathbf{G}$ is a CFG$\}$

Parsing

- Parsing: Process of analyzing how a particular string can be generated by a grammar
- Top-down: Start from the "start symbol"
- Bottom-up: Start from the string
- Hybrid: Combination of both

Shorthand

- Alternatives: $\mathrm{A} \rightarrow \alpha \mid \beta$
  - $\mathrm{A} \rightarrow \alpha$
  - $\mathrm{A} \rightarrow \beta$
- Optional element: $\mathrm{A} \rightarrow \alpha\ [\beta]$
  - $\mathrm{A} \rightarrow \alpha$
  - $\mathrm{A} \rightarrow \alpha\ \beta$

Demo

Top-down Parsing

- Example: Recursive-descent parsing
  - Keep the current input and the remaining part of the rule on stack
  - Expand nonterminals
  - Check terminals against the input

- $\mathrm{S} \rightarrow \mathrm{A}$
- $\mathrm{A} \rightarrow \mathrm{a\ A\ b} \mid \mathrm{a\ b}$

Demo

Bottom-up Parsing

- Example: Shift-reduce parsing
  - If the RHS of a rule matches a part of the input, reduce it to the LHS symbol
  - Otherwise, push the leftmost symbol onto the stack and repeat (shift)

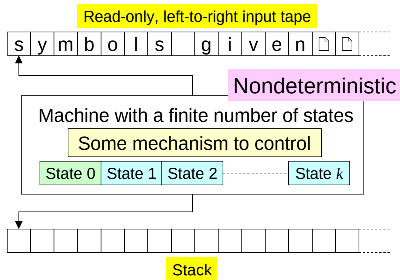
Essential component for processing?

Main CF Property

- Matching growth to the left and right
  - E.g., $\underbrace{(\ldots(}_{n}\ \underbrace{)\ldots)}_{n}$, 0^n1^n, $a_1 \ldots a_n a_n \ldots a_1$
- Characterized by *balanced* rules
  - E.g., $\mathrm{A} \rightarrow (\ \mathrm{A}\ )$, $\mathrm{A} \rightarrow 0\ \mathrm{A}\ 1$, $\mathrm{A} \rightarrow a_i\ \mathrm{A}\ a_i$
- Characterized by the use of stack
  - E.g., pushing '(' and later popping it and matching with ')'

$0^n1^n2^k3^k$, $0^n2^k3^k1^n$, $0^i2^k3^k0^{(n-i)}1^n$

Push-Down Automata (PDA)



PDA, Formally

- **M** = $(Q, \Sigma, \Gamma, \delta, q_0, F)$
  - Q: set of states
  - Σ: set of input symbols
  - Γ: set of stack symbols
  - δ: transition function — Note: ε as the empty string

    $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)$ — power set

    Γ^*: string of stack symbols
  - q_0: initial state $\in Q$
  - F: set of final states $\subseteq Q$

Equivalence of CFGs and PDAs

- Simulate a CFG with a PDA
- Simulate a PDA with a CFG

Group Exercise 2

- Are there any potential parsing problem the following grammars?

A	B	C
• $S \rightarrow A$	• $S \rightarrow A$	• $S \rightarrow A$
• $A \rightarrow a\ A\ b \mid a\ b$	• $A \rightarrow a\ A\ a$	• $A \rightarrow a\ A\ a$
• $A \rightarrow \varepsilon$	• $A \rightarrow b$	• $A \rightarrow a$

Ambiguity

- Multiple derivations of a single string
- A complete analysis requires nondeterminism.
- Problems with respect to:
  - Semantics
  - Efficiency

Deterministic PDA (DPDA)

- PDA where the range of the transition function is a set of singletons (unique state).
  - Equivalently,

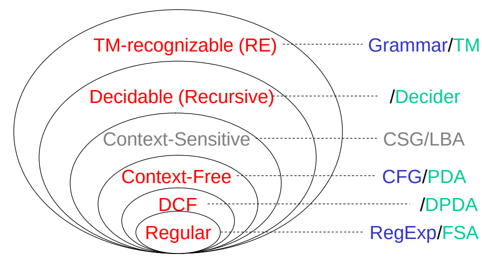
    PDA: $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma)$

    DPDA: $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma$
- Sufficiently powerful to characterize programming language core
- Efficient parsing algorithms are known.

Deterministic CFL (DCFL)

- A subset of CFLs that can be processed by a DPDA
- No easily identified class of grammars
  - Practical issue for specifying a programming language

Extended Chomsky Hierarchy



Unit Summary

- Many mini languages can be specified and processed by CFLs and PDAs.
- CFG: A single nonterminal on the LHS (e.g., $A \rightarrow \alpha$)
- CFL: Specified by CFGs
- PDA: Process CFLs
- DPDA/DCFL: Deterministic subset of PDAs/CFLs ~ backbone of programming languages

Summary Question

- Can you see the “context-free” property in some real-life phenomena? Explain.
- Questions/Comments/Suggestions