Pre-lecture brain teaser

Is the following language regular? Either way, prove it.
$L = \{\text{strings of properly matched open and closing parentheses}\}$

$E_x: \{ ()(), ((()))(), \ldots, 3 \}$
CS/ECE-374: Lecture 8 - Context-Free languages and Turing Machines

Lecturer: Nickvash Kani
Chat moderator: Samir Khan
February 16, 2021

University of Illinois at Urbana-Champaign
Pre-lecture brain teaser

Is the following language regular? Either way, prove it.

$L = \{ \text{strings of properly matched open and closing parentheses} \}$

Not regular

Prove via Fooling Set

$F = \{ x^n \mid n > 0 \}$

$i \neq j \implies \exists w = x^i x^j$ where $x^i \in L$ and $x^j \notin L$

Since $F$ is infinite, DFA $M$ representing $L$ is also infinite.
Larger world of languages!
Chomsky Hierarchy

Non-recursively-enumerable

Recursively-enumerable

Context-sensitive

Context-Free

Regular

Remember our hierarchy of languages
You’ve mastered regular expressions.
Chomsky Hierarchy

Non-recursively-enumerable

Recursively-enumerable

Context-sensitive

Context-Free

Regular

Now what about the next level up?
Context-Free Languages
Example

- $V = \{S\}$
- $T = \{a, b\}$
- $P = \{S \to \epsilon \mid a \mid b \mid aSa \mid bSb\}$
  (abbrev. for $S \to \epsilon, S \to a, S \to b, S \to aSa, S \to bSb$)
Example

\[ L = \{ \text{palindromes} \} \]

- \( V = \{ S \} \)
- \( T = \{ a, b \} \)
- \( P = \{ S \rightarrow \varepsilon \mid a \mid b \mid aSa \mid bSb \} \)
  (abbrev. for \( S \rightarrow \varepsilon, S \rightarrow a, S \rightarrow b, S \rightarrow aSa, S \rightarrow bSb \))

\[ w, w_2 \]

\[ [A, B] \]

\[ S \rightarrow aSa \rightarrow abSba \rightarrow abbSbba \rightarrow abb b bba \]

CF lang includes any string that can be derived from these production rules.
• $V = \{S\}$
• $T = \{a, b\}$
• $P = \{S \to \epsilon \mid a \mid b \mid aSa \mid bSb\}$
  (abbrev. for $S \to \epsilon, S \to a, S \to b, S \to aSa, S \to bSb$)

$S \rightsquigarrow aSa \rightsquigarrow abSba \rightsquigarrow abbSbba \rightsquigarrow abb\ b\ bba$

What strings can $S$ generate like this?
Context Free Grammar (CFG) Definition

Definition
A CFG is a quadruple \( G = (V, T, P, S) \)

- \( V \) is a finite set of non-terminal symbols

\[
G = \left( \text{Variables}, \quad \text{Terminals}, \quad \text{Productions}, \quad \text{Start var} \right)
\]
Context Free Grammar (CFG) Definition

Definition
A CFG is a quadruple $G = (V, T, P, S)$

- $V$ is a finite set of non-terminal symbols
- $T$ is a finite set of terminal symbols (alphabet) $\Sigma$

$$G = \left( \text{Variables, Terminal, Productions, Start var} \right)$$
Context Free Grammar (CFG) Definition

Definition
A CFG is a quadruple \( G = (V, T, P, S) \)

- \( V \) is a finite set of non-terminal symbols
- \( T \) is a finite set of terminal symbols (alphabet)
- \( P \) is a finite set of productions, each of the form
  \[ A \rightarrow \alpha \]
  where \( A \in V \) and \( \alpha \) is a string in \((V \cup T)^*\).

Formally, \( P \subset V \times (V \cup T)^* \).

Left side is always a single variable

\[
G = \left( \text{Variables, Terminal, Productions, Start var} \right)
\]
Context Free Grammar (CFG) Definition

**Definition**
A CFG is a quadruple $G = (V, T, P, S)$

- $V$ is a finite set of **non-terminal symbols**
- $T$ is a finite set of **terminal symbols** (alphabet)
- $P$ is a finite set of **productions**, each of the form $A \rightarrow \alpha$
  where $A \in V$ and $\alpha$ is a string in $(V \cup T)^*$. Formally, $P \subseteq V \times (V \cup T)^*$.
- $S \in V$ is a **start symbol**

$$G = (\text{Variables, } \text{Terminals, } \text{Productions, } \text{Start var})$$
Example formally...

\[ L = \{ \text{palindromes} \} \]

- \( V = \{ S \} \)
- \( T = \{ a, b \} \)
- \( P = \{ S \to \epsilon, a, b, aSa, bSb \} \) (abbrev. for \( S \to \epsilon, S \to a, S \to b, S \to aSa, S \to bSb \))

\[
G = \left( \begin{array}{c}
\{ S \}, \\
\{ a, b \}, \\
\{ S \to \epsilon, \\
S \to a, \\
S \to b, \\
S \to aSa, \\
S \to bSb \}
\end{array} \right) \]

\( S \)
Examples

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

\[ V = \{s^3\} \quad T = \{0, 1\} \quad S = S \]

\[ P = \{ s \to 0s11s \} \]
Examples

\[ L = \{0^n1^n \mid n \geq 0\}\]

\[ S \rightarrow \epsilon \mid 0S1 \]
Context Free Languages

Definition
The language generated by CFG $G = (V, T, P, S)$ is denoted by $L(G)$ where $L(G) = \{ w \in T^* | S \Rightarrow^* w \}$. 

$L = \{ \epsilon, 01, 0011, \ldots \}$
Definition
The language generated by \textbf{CFG} $G = (V, T, P, S)$ is denoted by $L(G)$ where $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$.

Definition
A language $L$ is \textbf{context free (CFL)} if it is generated by a context free grammar. That is, there is a \textbf{CFG} $G$ such that $L = L(G)$. 
Example

\[ L = \{0^n1^n \mid n \geq 0\} \]

\[ S \to \epsilon | 0S1 \]

\[ L = \{0^n1^m \mid m > n\} \]

\[ L = \left\{ w \in \{(,\}\}^* \mid w \text{ is properly nested string of parenthesis} \right\} \]

\[ S \to \epsilon | (S) S \]

recursive - regex
Context-Sensitive Languages
Chomsky Hierarchy

Non-recursively-enumerable

Recursively-enumerable

Context-sensitive

Context-Free

Regular

Now that we mastered acknowledged Context-Free Languages.....
Chomsky Hierarchy

Non-recursively-enumerable

Recursively-enumerable

Context-sensitive

Context-Free

Regular

On to the next one.....
Example

The language $L = \{a^n b^n c^n | n \geq 1\}$ is not a context free language.

Prove via pumping lemma
The language $L = \{a^n b^n c^n | n \geq 1\}$ is not a context free language. \textit{but it is a context-sensitive language!}

- $V = \{S, A, B\}$
- $T = \{a, b, c\}$
- $P = \begin{cases} 
S \rightarrow abc | aAbc, \\
    Ab \rightarrow bA, \\
    Ac \rightarrow Bb \quad & \text{\textit{not shown, but exists}} \\
    bB \rightarrow Bb \\
    aB \rightarrow aa | aaA \\
\end{cases}$
The language \( L = \{a^n b^n c^n | n \geq 1\} \) is not a context free language. *but it is a context-sensitive language!* 

\[
\begin{align*}
V &= \{S, A, B\} \\
T &= \{a, b, c\} \\
P &= \left\{ \begin{array}{l}
S \to abc | aAbc, \\
\quad Ab \to bA, \\
\quad Ac \to BbCc \\
\quad bB \to Bb \\
\quad aB \to aa | aaA \\
\end{array} \right\
\end{align*}
\]

\[
S \leadsto aAbc \leadsto abAc \leadsto abBbCc \leadsto aBbCc \leadsto aaAbbCc \leadsto aabAAbCc \\
\leadsto aabbAcc \leadsto aabbBbCc \leadsto aabBbCcCc \leadsto aaBbBbCcCc \\
\leadsto aaabbbCcCc
\]
Context-Free Grammar (CFG) Definition

Definition
A CSG is a quadruple \( G = (V, T, P, S) \)

- \( V \) is a finite set of **non-terminal symbols**
- \( T \) is a finite set of **terminal symbols** (alphabet)
- \( P \) is a finite set of **productions**, each of the form
  \[ \alpha \rightarrow \beta \]
  where \( \beta \) and \( \alpha \) are strings in \((V \cup T)^*\).
- \( S \in V \) is a **start symbol**

\[
G = \left( \text{Variables, Terminal, Productions, Start var} \right)
\]
Example formally...

\[ L = \{a^n b^n c^n | n \geq 1 \} \]

- \( V = \{S, A, B\} \)
- \( T = \{a, b, c\} \)
- \( P = \begin{cases} 
  S \rightarrow abc|aAbc, \\
  Ab \rightarrow bA, \\
  Ac \rightarrow BbCc \\
  bB \rightarrow Bb \\
  aB \rightarrow aa|aaA 
\end{cases} \)

\[ G = \left( \begin{array}{c} 
  \{S, A, B\}, \\
  \{a, b, c\}, \\
  \{S \rightarrow abc|aAbc, \\
  Ab \rightarrow bA, \\
  Ac \rightarrow BbCc \\
  bB \rightarrow Bb \\
  aB \rightarrow aa|aaA \} \end{array} \right) \]
Turing Machines
“Most General” computer?

- DFA\(^s\) are simple model of computation.
- Accept only the regular languages.
- Is there a kind of computer that can accept any language, or compute any function?
- Recall counting argument. Set of all languages:
  \( \{ L \mid L \subseteq \{0,1\}^* \} \) is countably infinite / uncountably infinite
“Most General” computer?

- DFA s are simple model of computation.
- Accept only the regular languages.
- Is there a kind of computer that can accept any language, or compute any function?
- Recall counting argument. Set of all languages: \( \{ L \mid L \subseteq \{0, 1\}^* \} \) is countably infinite / uncountably infinite.
- Set of all programs: \( \{ P \mid P \text{ is a finite length computer program} \} \): is countably infinite / uncountably infinite.
“Most General” computer?

- DFAs are simple model of computation.
- Accept only the regular languages.
- Is there a kind of computer that can accept any language, or compute any function?
- Recall counting argument. Set of all languages: \( \{L \mid L \subseteq \{0, 1\}^*\} \) is countably infinite / uncountably infinite.
- Set of all programs: \( \{P \mid P \text{ is a finite length computer program}\} \): is countably infinite / uncountably infinite.
- **Conclusion:** There are languages for which there are no programs.
Chomsky Hierarchy

- Regular
- Context-Free
- Context-sensitive
- Recursively enumerable
- Countably infinite
- Non-recursively enumerable
- Uncountably infinite
Onto our final class of languages - recursively enumerable (aka Turing-recognizable) languages.
What is a Turing machine
Turing machine

- Input written on (infinite) one sided tape.
- Special blank characters.
- Finite state control (similar to DFA).
- Every step: Read character under head, write character out, move the head right or left (or stay).
High level goals

- Church-Turing thesis: TMs are the most general computing devices. So far no counter example.
- Every TM can be represented as a string.
- Existence of Universal Turing Machine which is the model/inspiration for stored program computing. UTM can simulate any TM
- Implications for what can be computed and what cannot be computed
Turing machine: Formal definition

A *Turing machine* is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

- $Q$: finite set of states.
- $\Sigma$: finite input alphabet.
- $\Gamma$: finite tape alphabet. *includes "\_" = blank space*
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$: Transition function.
- $q_0 \in Q$ is the initial state.
- $q_{acc} \in Q$ is the *accepting/final* state.
- $q_{rej} \in Q$ is the *rejecting* state.
- □ or : Special blank symbol on the tape.
Turing machine: Transition function

\[ \delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\} \]

As such, the transition

\[ \delta(q, c) = (p, d, L) \]

• \( q \): current state.
• \( c \): character under tape head.
• \( p \): new state.
• \( d \): character to write under tape head
• \( L \): Move tape head left.
### Turing machine: Transition function

\[ \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\} \]

As such, the transition

\[ \delta(q, c) = (p, d, L) \]

- **q**: current state.
- **c**: character under tape head.
- **p**: new state.
- **d**: character to write under tape head
- **L**: Move tape head left.

Missing transitions lead to hell state.
“Blue screen of death.”
“Machine crashes.”
Some examples of Turing machines
Example: Turing machine for $a^n b^n c^n$

Can view this Turing machine in action on turingmachine.io!
Languages defined by a Turing machine
Recursive vs. Recursively Enumerable

- *Recursively enumerable* (aka *RE*) languages

  \[ L = \{ L(M) \mid M \text{ some Turing machine} \} . \]

- *Recursive / decidable* languages

  \[ L = \{ L(M) \mid M \text{ some Turing machine that halts on all inputs} \} . \]
Recursive vs. Recursively Enumerable

• *Recursively enumerable* (aka RE) languages *(bad)*

\[ L = \{L(M) \mid M \text{ some Turing machine}\}. \]

• *Recursive / decidable* languages *(good)*

\[ L = \{L(M) \mid M \text{ some Turing machine that halts on all inputs}\}. \]
Recursive vs. Recursively Enumerable

- *Recursively enumerable* (aka RE) languages *(bad)*

\[ L = \{ L(M) \mid M \text{ some Turing machine} \} \]

- *Recursive / decidable* languages *(good)*

\[ L = \{ L(M) \mid M \text{ some Turing machine that halts on all inputs} \} \]

- Fundamental questions:
  - What languages are RE?
  - Which are recursive?
  - What is the difference?
  - What makes a language decidable?