What is the context-free grammar of the following push-down automata:

\[
\begin{align*}
S & \rightarrow S \\
S & \rightarrow T \\
T & \rightarrow 0 \\
T & \rightarrow 1 \\
0 & \rightarrow S \\
1 & \rightarrow T \\
\end{align*}
\]
ECE-374-B: Lecture 7 - Context-sensitive and decidable languages

Instructor: Nickvash Kani
September 14, 2023

University of Illinois at Urbana-Champaign
What is the context-free grammar of the following push-down automata:

$$S \rightarrow OT1$$

$$T \rightarrow TO1\epsilon$$
Larger world of languages!
Remember our hierarchy of languages
You’ve mastered regular expressions.
Now what about the next level up?
Chomsky Hierarchy

Non-recursively-enumerable

Recursively-enumerable

Context-sensitive

Context-Free

Regular

On to the next one.....
Context-Sensitive Languages
Example

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

- $V = \{S, A, B\}$
- $T = \{a, b, c\}$
- $P = \{S \rightarrow abc|aAbc,$
  $\quad Ab \rightarrow bA,$
  $\quad Ac \rightarrow Bb$,
  $\quad bB \rightarrow Bb$
  $\quad aB \rightarrow aa|aaA\}$

CFG:

\[
 V \rightarrow \alpha - \\
 \alpha = \text{EVOFT3}
\]
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!*

- $V = \{S, A, B\}$
- $T = \{a, b, c\}$
- $P = \begin{align*}
  S & \rightarrow abc | aAbc, \\
  A b & \rightarrow bA, \\
  A c & \rightarrow B b c c \\
  b B & \rightarrow B b \\
  a B & \rightarrow a a | a a A
\end{align*}$

$S \rightarrow aAbc \rightarrow abAc \rightarrow abBbCc \rightarrow aBbCc \rightarrow aaA.bbCc \rightarrow aabAbCc \rightarrow aabbbA.CC \rightarrow aabbbBbCcC \rightarrow aabBbbCcCc \rightarrow aaBbbCcCc \rightarrow aaabbbCcC$
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 $\alpha$ and $\beta$ are strings in $(V \cup T)^*$.
- $S \in V$ is a start symbol

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

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

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

\[ G = \begin{cases} 
\{S, A, B\}, & \{a, b, c\}, \\
S \rightarrow abc|aAbc, & S \\
& Ab \rightarrow bA, \\
& Ac \rightarrow Bbb, \\
& bB \rightarrow Bb, \\
& aB \rightarrow aa|aaA 
\end{cases} \]
Other examples of context-sensitive languages

\[ L_{Cross} = \{ a^m b^n c^m d^n | m, n \geq 1 \} \] (1)
Turing Machines
“Most General” computer?

- DFA
- 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.
- Conclusion: There are languages for which there are no programs.
“Most General” computer?

- DFA’s are a 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
- Non-recursively-enumerable
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
Examples of Turing
- binary increment
A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$

- $Q$: finite set of states.
- $\Sigma$: finite input alphabet.
- $\Gamma$: finite tape alphabet.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$: Transition function.
- $q_0 \in Q$ is the initial state.
- $q_{\text{acc}} \in Q$ is the accepting/final state.
- $q_{\text{rej}} \in Q$ is the rejecting state.
- $\square$ or $\blacksquare$: Special blank symbol on the tape.

Implicit
The transition function \( \delta \) of a Turing machine is given by the following:

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

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.

Can also be written as

\[
c \rightarrow d, L
\]
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
• equal strings TM
• palindrome TM
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**
  \[ L = \{ L(M) \mid M \text{ some Turing machine that halts on all inputs} \} . \]

Fundamental questions:
- What languages are RE?
- Which are recursive?
- What is the difference?
- What makes a language decidable?
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 \((\text{bad})\)

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

• Recursive / decidable languages \((\text{good})\)

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

• Fundamental questions:
  • What languages are RE?
  • Which are recursive?
  • What is the difference?
  • What makes a language decidable?
What is Decidable?
Decidable vs recursively-enumerable

A semi-decidable problem (equivalent of recursively enumerable) could be:

- **Decidable** - equivalent of recursive (TM always accepts or rejects).
- **Undecidable** - Problem is not recursive (doesn’t always halt on negative)

There are undecidable problem that are not semi-decidable (recursively enumerable).
Infinite Tapes? Do we need them?
Let’s look at the TM that recognizes \( L = \{a^n b^n c^n | n \geq 0\} \):
• (Nondeterministic) Linear bounded automata can recognize all context sensitive languages.

• Machine can non-deterministically apply all production rules to input in reverse and see if we end up with the start token.
Well that was a journey....
### Grammar

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Languages</th>
<th>Production Rules</th>
<th>Automation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type-0</td>
<td>Turing machine</td>
<td>γ → α</td>
<td>Turing machine</td>
<td>(L = {w \mid w \text{ is a TM which halts}})</td>
</tr>
<tr>
<td>Type-1</td>
<td>Context-sensitive</td>
<td>αAβ → αγβ</td>
<td>Linear bounded Turing machine</td>
<td>(L = {a^n b^n c^n \mid n &gt; 0})</td>
</tr>
<tr>
<td>Type-2</td>
<td>Context-free</td>
<td>A → α</td>
<td>Non-deterministic Push-down automata</td>
<td>(L = {a^n b^n \mid n &gt; 0})</td>
</tr>
<tr>
<td>Type-3</td>
<td>Regular</td>
<td>A → aB</td>
<td>Finite State Machine</td>
<td>(L = {a^n \mid n &gt; 0})</td>
</tr>
</tbody>
</table>

### Meaning of symbols:

- \(a\) = terminal
- \(A, B\) = variables
- \(\alpha, \beta, \gamma\) = string of \(\{a \cup A\}^*\)
- \(\alpha, \beta\) = maybe empty —— \(\gamma\) = never empty