Programming Languages and Formal Language Theory

We ask the question: *Does a particular formal language describe some key aspect of a programming language*?

Then we find out if that language *isn’t* in a particular language class.
For example, if we abstract some aspect of the programming language structure to the formal language:
\{ww^R \mid \text{where } w \in \{a, b\}^*, w^R \text{ is the reverse of } w\} we can then ask if this language is a regular language.

If this is false, i.e. the language is not regular, then we have to go beyond regular languages.
Consider a regular expression for matching arithmetic expressions:

\[2 + 3 \times 4\]
\[8 \times 10 + 24\]
\[2 + 3 \times 2 + 8 + 10\]

\textbf{num} [0-9]+  
\textbf{op} (\+|\-|\\*|\/)  
\textbf{ws} [ \t\]*  

\%

\{ws\} {   }  
\{num\}\{ws\}\{op\}\{ws\}\{num\}\}*  \{ printf("yes\n"); \}  
.  \{ printf("no\n"); \}

Can we compute the \textit{meaning} of these expressions?
Recursion in Regular Languages

- Construct the finite state automata and associate the meaning with the state sequence
- However, this solution is missing something crucial about arithmetic expressions – *what is it?*
Consider the following arithmetic expressions

- $(((2) + (3)) \times (4))$
- $((8) \times ((10) + (−24)))$

Map $(→a$ and $)→b$. Map everything else to $\epsilon$ (keep only the tree structure)

This results in strings like $aaababbab$ and $aabaababbab$.

So the language is a set $L = \{\epsilon, ab, aabb, abab, \ldots\}$

What is a good description of this language?

Consider the intersection of $L$ with the language of the regexp $a^*b^*$. If $L$ is regular then the intersection is also regular.

Let’s call it $L_{\text{new}} = \{a^n b^n : n \geq 0\}$ or simply $a^n b^n$ for short.
Pumping Lemma proofs

- Is $L$ a regular language?
- For any infinite set of strings generated by a finite-state machine if you consider a string that is long enough from this set, there has to be a loop which visits the same state at least twice (from the pigeonhole principle)
- Thus, in a regular language $L$, there are strings $x, y, z$ such that $xy^iz \in L$ for $i \geq 0$ where $y \neq \epsilon$
- We can use this basic characteristic of regular languages to show that $a^n b^n$ cannot be regular
The Chomsky Hierarchy

- **unrestricted** or **type-0** grammars, generate the *recursively enumerable* languages, automata equals *Turing machines*

- **context-sensitive** or **type-1** grammars, generate the *context-sensitive* languages, automata equals *Linear Bounded Automata*

- **context-free** or **type-2** grammars, generate the *context-free* languages, automata equals *Pushdown Automata*

- **regular** or **type-3** grammars, generate the *regular* languages, automata equals *Finite-State Automata*
The Chomsky Hierarchy

- A system of grammars $G = (N, T, P, S)$
- $T$ is a set of symbols called terminal symbols. Also called the alphabet $\Sigma$
- $N$ is a set of non-terminals, where $N \cap T = \emptyset$
  Some notation: $\alpha, \beta, \gamma \in (N \cup T)^*$
  $N$ is sometimes called the set of variables $V$
- $P$ is a set of production rules that provide a finite description of an infinite set of strings (a language)
- $S$ is the start non-terminal symbol (similar to the start state in a FSA)
Languages

- Language defined by $G$: $L(G)$
  - $L(G)$: set of strings $w \in T^*$ derived from $S$
  - $S \Rightarrow^+ w$ (derives in 1 or more steps using rules in $P$)
  - $w$ is a sentence of $G$
  - Sentential form: $S \Rightarrow^+ \alpha$ and $\alpha$ contains a mix of terminals and non-terminals

- Two grammars $G_1$ and $G_2$ are equivalent if $L(G_1) = L(G_2)$
The Chomsky Hierarchy: \( G = (N, T, P, S) \) where,
\( \alpha, \beta, \gamma \in (N \cup T)^* \)

- **unrestricted** or **type-0** grammars: \( \alpha \rightarrow \gamma \), such that \( \alpha \neq \epsilon \)
- **context-sensitive** or **type-1** grammars: \( \alpha \rightarrow \gamma \), where \( |\gamma| \geq |\alpha| \)
  CSG Normal Form: \( \alpha A \beta \rightarrow \alpha \gamma \beta \), such that \( \gamma \neq \epsilon \) and \( S \rightarrow \epsilon \)
  if \( \epsilon \in L(G) \)
- **context-free** or **type-2** grammars: \( A \rightarrow \gamma \)
- **regular** or **type-3** grammars: \( A \rightarrow a B \) or \( A \rightarrow a \)
Examples of Languages in the Chomsky Hierarchy

- **context-sensitive** grammars: $0^i$, $i$ is a prime number
- **indexed** grammars: $0^n1^n2^n \ldots m^n$, for any fixed $m$ and $n \geq 0$
- **context-free** grammars: $0^n1^n$ for $n \geq 0$; also $\{0^n1^n2^m\} \cup \{0^m1^n2^n\}$ which is inherently ambiguous, i.e. no unambiguous CFG exists!
- **deterministic context-free** grammars: $S' \rightarrow S \ c$, $S \rightarrow S \ A | A$, $A \rightarrow a \ S \ b | ab$: the language of ”balanced parentheses”
- **regular** grammars: $(0|1)^*00(0|1)^*$
<table>
<thead>
<tr>
<th>Language</th>
<th>Automaton</th>
<th>Grammar</th>
<th>Recognition</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursively Enumerated Languages</td>
<td>Turing Machine</td>
<td>Unrestricted</td>
<td>Undecidable</td>
<td>Arbitrary</td>
</tr>
<tr>
<td></td>
<td>![Turing Machine Diagram]</td>
<td><em>Baa → A</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context-Sensitive Languages</td>
<td>Linear-Bounded Automaton</td>
<td>Context-Sensitive</td>
<td>NP-Complete</td>
<td>Crossing</td>
</tr>
<tr>
<td></td>
<td>![Linear-Bounded Automaton Diagram]</td>
<td><em>At → aA</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context-Free Languages</td>
<td>Pushdown (stack)</td>
<td>Context-Free</td>
<td>Polynomial</td>
<td>Nested</td>
</tr>
<tr>
<td></td>
<td>![Pushdown Automaton Diagram]</td>
<td><em>S → gSc</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regular Languages</td>
<td>Finite-State Machine</td>
<td>Regular</td>
<td>Linear</td>
<td>Strictly Local</td>
</tr>
<tr>
<td></td>
<td>![Finite-State Machine Diagram]</td>
<td><em>A → cA</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Given grammar $G$ and input $x$, provide algorithm for: Is $x \in L(G)$?

- **unrestricted**: undecidable
- **context-sensitive**: NSPACE($n$) – linear non-deterministic space
- **indexed** grammars: NP-Complete
- **context-free**: $O(n^3)$
- **deterministic context-free**: $O(n)$
- **regular** grammars: $O(n)$
Aspects of PL structure cannot be represented by FSAs
We can show that a language is not regular.
If such a language is needed for our programming language then we have to use something more powerful than a regular language
Chomsky hierarchy: from FSAs to Turing machines
Context-free grammars (seems sufficient for PLs) but problems with ambiguity