Automata Theory (CS1.302)

Monsoon 2021, IIIT Hyderabad
21 October, Thursday (Lecture 5)

Models of Computation

Grammars

Chomsky Normal Form

A CFG G is said to be in Chomsky Normal Form (CNF) if every rule has one of the following forms:

$$\begin{aligned} V &\to VV \\ V &\to T \quad , \\ S &\to \varepsilon \end{aligned}$$

where V is any variable, T any terminal, and S the start variable.

The CNF of a grammar is useful when deciding whether it generates a string w or not. If the grammar is not in CNF, an arbitrary number of steps might be needed to generate w, making the question undecidable.
However, a grammar in CNF takes at most $2n - 1$ steps to derive a string of length n. This bound makes the membership problem decidable.

To prove this claim, we will use induction. If $|w| = 1$, then one application of a rule of the form $V \to T$ suffices to derive w.
Now, assuming that all strings of length k take at most $2k - 1$ steps, we need to show that a string w of length $k + 1$ takes at most $2k + 1$ steps. First, note that a derivation of w must start with a rule of the form $V \to V_1V_2$, so w can be written as xy, where x can be derived from V_1 and y from V_2, and $x, y \neq \varepsilon$. Since $|x|, |y| < w$, we can say that x and y take at most \$2|x| -1 \$ and $2|y| - 1$ steps to be derived. Thus w takes at most

$$1 + (2|x| - 1) + (2|y| - 1) = 2(|x| + |y|) - 1 = 2|w| - 1$$

steps, QED.

We can also show that any CFL can be generated by a CFG in CNF, via a constructive proof. Suppose that G is an arbitrary CFG. Then to construct G' in CNF,

- Add a new start variable $S' \to S$.
- Remove all rules of the form $V \to \varepsilon$. For every rule with V on the RHS, add new rules with V deleted. For example, after $A \to \varepsilon$ is deleted, $B \to uAvAw$ generates $B \to uAvAw \mid uAvw \mid uvAw \mid uvw$. In case there is a rule $B \to A$, replace it with $B \to \varepsilon$ and repeat.
- Remove all unit rules $V \to V'$. For all rules of the form $V' \to u$, add a rule $V \to u$ (if this rule was not already removed).
- Remove long rules $A \to u_1 u_2 \ldots u_k$ (where the u_i are variables or terminals). Replace such rulses with the rules $A \to u_1 A_1, A_1 \to u_2 A_2, \cdots A_{k-2} u_{k-1} u_k$. For all u_i that are variables, replace $A_{i-1} \to u_i A_i$ with $A_{i-1} \to U_i A_i$ and $U_i \to u_i$.

Pushdown Automata

The way finite automata recognise the same set of languages as regular expressions and linear grammars, pushdown automata can recognise context-free languages.

We note first, however, that an FSM that recognises all CFLs would need unbounded memory. It may not use the memory (depending on the language and/or the string), in which case it could behave like an NFA or a DFA, but it needs to make of it to recognise non-regular CFLs.

A pushdown automaton, therefore, uses a *stack* as an unbounded memory device. New symbols can be *pushed* onto the stack, and the most recently pushed symbol can be *popped* off it. A stack is thus a LIFO structure. We allocate a special symbol `$` to demarcate the bottom of the stack and always pushed before anything else to the stack.

A PDA makes transitions based on the input symbol and the symbol on top of the stack. Along with transitions, at every read, it may manipulate the stack (by pushing or popping).