CS 4810 Midterm 2 Key Ideas

October 2019

1 Context-Free Languages (CFL)

Section 5.1.2 of Introduction to Automata Theory, Languages, and Computation gives a formal definition of Context-Free Grammars. We define context-free languages to be all sets of strings that are accepted by a context-free grammar (see section 5.1.5 of Introduction to Automata Theory, Languages, and Computation for additional discussion).

1.1 CFLs properly contains regular sets

This is the idea that the set of regular sets is properly contained by the set of Context-Free Languages. This means that if a set is regular, then it is also a context free language, and there exists a context-free language that is not regular. Likely, you should look at how to prove both of those claims.

1.2 CFLs defined by error in string

This type of approach is often used if one wanted to create a CFL for the complement of a set. You can start by embedding the error that is needed to make the string not in the set, then build up all options around it. See problem 3 in Homework 7.

1.3 Pushdown automata (PDA)

Our formal notation for pushdown automata (PDA) involves seven components. We write the specification of a PDA P as follows:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

The components have the following meanings:
Q : A finite set of states, like the states of a finite automata.

Σ : A finite set of input symbols, also analogous to the corresponding component of a finite automaton.

Γ : A finite stack alphabet. This component, which has no finite-automaton analog, is the set of symbols that we are allowed to push onto the stack.

δ : The transition function. As for a finite automaton, δ governs the behavior of the automaton. Formally, δ takes as argument a triple $\delta(q, a, X)$, where:

1. q is a state in Q
2. a is either an input symbol in Σ or $a = \varepsilon$, the empty string, which is assumed not be an input symbol.
3. X is a stack symbol, that is, a member of Γ

The output of δ is a finite set of pairs (p, γ), where p is the new state, and γ is the string of stack symbols that replaces X at the top of the stack. For instance, if $\gamma = \varepsilon$, then the stack is popped, if $\gamma = X$, then the

stack is unchanged, and if $\gamma = YZ$, then X is replaced with Z, and Y is pushed onto the stack.

q_0: the start state. The PDA is in this state before making any transitions.

Z_0: The start symbol. Initially, the PDA's stack consists of one instance of this symbol, and nothing else.

F : The set of accepting states, or final states.

1.4 Acceptance by final state and by empty store define same class of languages

The high level idea here is that we can adjust the semantics of a pushdown automata such that 'accepting' a string is either done by final state (similar to DFA acceptance), or by empty store (e.g., when the stack is empty and the string has been fully read). The following sections are from Introduction to Automata Theory, Languages, and Computation.

Section 6.2.1 describes acceptance by final state.

Section 6.2.2 describes acceptance by empty stack.

Section 6.2.3 describes going from empty stack to final state.

Section 6.2.4 describes going from final state to empty stack.

1.5 Reduction of many state to one state

This is the idea that given a Pushdown automaton P_1 with many states, once can construct an equivalent Pushdown automaton P_2 that accepts the same language ($L(P_1) = L(P_2)$) such that P_2 only holds a single state. Intuitively, this is done by storing statefulness on stack variables instead of as states.

The following is one possible construction:

Suppose $P_0 = (Q, \Sigma, \Gamma_0, \delta_0, q_0, Z)$ is a multi-state pushdown automata that accepts by empty store. Then define $P := (\{q_0\}, \Sigma, \Gamma, \delta, q_0, Z)$ as a single-state pushdown automaton such that

$$\Gamma := \{[pXq] \mid p, q \in Q \text{ and } X \in \Gamma\}$$

and

If $\delta_0(p, a, X) = \{(q, \epsilon)\}$, then $\delta(q_0, a, [pXq]) = \{(q_0, \epsilon)\}$

If $\delta_0(p, a, X) = \{(q, Y)\}$, then $\delta(q_0, a, [pXb]) = \{(q_0, [qYb])\}$ for all $b \in Q$

If $\delta_0(p, a, X) = \{(q, YZ)\}$, then $\delta(q_0, a, [pXb]) = \{q_0, [qYg][gZb]\}$ for all $g, b \in Q$

Then the constructed PDA P should accept the same strings at P_0. The main idea behind this construction is that we keep the current state with the topmost stack symbol on the left so if $[pAq]$ is the top stack then we're in state p. Also, if we pop $[pAq]$, we must still know what state we're in. That's where q comes in. Before we push a new symbol onto the stack, we guess what state we'll be in when the stack returns to the same height. That guess is on the right. We verify the guess later since we can only delete $[pAq]$ if there is a transition in δ_0 that deletes A and moves from p to q.

1.6 One state equivalent to CFL

Let $G = (V, T, P, S)$ be a Context-Free grammar. So V is a set of symbols, T is set of terminal symbols, P is the set of productions, and S is the start symbol. We can then construct a PDA

$$P := (\{q\}, T, V \cup T, \delta, q, S)$$

with a transition function δ such that

$$\begin{aligned} \delta(q, \epsilon, A) &= \{(q, \beta) \mid A \to \beta \text{ is a production in } P\} && \forall A \in V \\ \delta(q, a, a) &= \{(q, \epsilon)\} && \forall a \in T \end{aligned}$$

Then the PDA P accepts $L(G)$ by empty store.

This describes a one state equivalent to a context-free language, and is proved in section 6.3 of Introduction to Automata Theory, Languages, and Computation.

1.7 Machine construction for intersection with regular sets

If L is a context-free language and R is a regular language, then $L \cap R$ is a context-free language. See page 286 of Introduction to Automata Theory, Languages, and Computation for the proof.

1.8 Normal forms: Eliminate useless variables, epsilon productions, singleton productions

Largely, this information is covered in section 4.1 of *Formal Languages and their relation to automata*. Here are a few of the relevant theorems. You should also try and understand the proofs of these.

Theorem 4.1 There is an algorithm for determining if the language generated by a context-free grammar is empty.

Theorem 4.2 Given any context-free grammar $G = (V_N, V_T, P, S)$, generating a nonempty language, it is possible to find an equivalent grammar G_1, such that for every variable A of G_1 there is a terminal string w, such that $A \implies * w$.

Theorem 4.3 Given any context-free grammar generating a nonempty cfl L, its is possible for find a grammar G, generating L, such that for each variable A there is a derivation

$$S \implies * w_1 A w_3 \implies * w_1 w_2 w_3$$

where w_1, w_2 and w_3 are in V_T^*.

Theorem 4.4 Given a context-free grammar G, we can find an equivalent grammar G_1 with no productions of the form $A \to B$, where A and B are variables.

1.9 Chomsky Normal form

Any context-free language can be generated by a grammar in which all productions are of the form $A \to BC$ or $A \to a$. Here A, B, and C are variables and a is a terminal. This is Theorem 4.5 on page 52 of Formal Languages and their relation to automata. The proof is on the same page.

1.10 Pumping lemma for context-free languages

See section 7.2 of Introduction to Automata Theory, Languages, and Computation.

1.11 Closure properties of context-free languages

See section 7.3 of Introduction to Automata Theory, Languages, and Computation.

1.12 Decision properties

See section 7.4 of Introduction to Automata Theory, Languages, and Computation.

1.13 Dynamic programming applied to membership problem

The name of the game here is to determine if a word $w \in \Sigma^*$ where Σ is the alphabet in question, belongs to some Context-free language L. This can be done via dynamic programming (which is an algorithmic technique covered in CS 4820).

https://courses.cs.washington.edu/courses/cse322/05au/cky.pdf

2 Work cited

1. Formal Languages and their relation to automata, John E. Hopcroft.
2. Introduction to Automata Theory, Languages, and Computation, John E. Hopcroft.
3. https://courses.cs.washington.edu/courses/cse322/05au/cky.pdf