CS 121 Section 5

Harvard University

October 20 2012

Overview

This week we will focus on reviewing the core concepts involved with PDAs, Pumping lemma for CFL, and closure properties of context-free language.

1 Concept Review

1.1 PDA

Intuitively a Pushdown Automata (PDA) represents a NFA with an additional (unbounded size) stack. The addition of the stack makes PDAs much powerful than NFA. For example, the language $L = \{a^n b^n : n \in \mathbb{N}\}$ is recognized by a PDA, but not by any NFA. Intuitively, PDAs the stack gives PDAs the ability to perform simple counting tasks.

1.2 PDAs v.s. CFGs

PDAs and Context-Free Grammars are equivalent in power.

Theorem 1.1. *The CFLs are the languages accepted by PDAs.*

1.3 Pumping lemma for Context-Free Language

Lemma 1.1. *If L is context-free, then there is a number p such that any $s \in L$ of length at least p can be divided into $s = uvxyz$, where*

- $v \neq \epsilon$ *or* $y \neq \epsilon$,
- $|vxy \leq p|$ *,and*
- $uv^i xy^i z \in L$ *for every* $i \geq 0$.

1.4 Closure properties of Context-Free Language

The Context-Free Languages are closed under:

- Union
- Concatenation
- Kleene $*$
- Intersection with a regular set

The Context-Free Languages are **not** closed under:

- Complement
- Intersection

2 Exercises

Exercise 2.1. *Explain and justify the following statement: "Almost all languages are not context-free."*
We mean that, over a given alphabet, the set of all languages is uncountable, but the set of context-free languages is countable. One could show (don't worry about doing it) that if one were to pick a language at random from the set of all languages, the probability that it would be context-free would be smaller than any positive number.

For any CFL, we can describe it in a finite string (the way we usually do). A lexicographical ordering of those strings yields an enumeration of CFLs.

Exercise 2.2. *Show that $L = \{a^n b^{2n}\}$ is context-free by giving a PDA that accepts it. Draw the state diagram and write the 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$*
Push a's for a while, then pop an a every time you read two b's.

Exercise 2.3. *Determine, with proof, whether or not each of the following languages is context-free.*

1. $\{ww : w \in \Sigma^*\}$
   No. Pump $a^p b^p a^p b^p$. It is casework from there. The key is that, looking at the string in four "sections" (a^p, b^p, a^p, and b^p), the parts we are pumping must be in either one section or two adjacent sections.

2. $L = \{w$ *is not of the form* $a^n b^n\}$
   Yes. Write grammars for $a^i b^j$ $i > j$, $i < j$. Then write a grammar for the language where there is some a following a b. Unioning these three grammars gives the result. These grammars should be

$S -> aSb|aS|A$
$S -> aSb|Sb|b$
$S -> XbXaX, X -> aX|bX|\epsilon$

3. $\{w \in \{(,)\}^* : w$ *is not properly parenthesized*$\}$
Yes. I would think it's easiest with a PDA. Push (*when you encounter them and pop them when you see*). *If you have ever seen more*) *than* (, *move to a looping accept state. If you are done reading your string and have a non-empty stack, you should be able to* ϵ*-transition to an accept state as well.*

4. $L = \{a^i b^j c^k | 0 \leq i \leq j \leq k\}$
No. Suppose it were. Pump $a^p b^p c^p$*. If our pumpable sections contain a's, they cannot contain c's, so pump up for the contradiction. It they contain c's they cannot contain a's, so pump down for the contradiction. If they contains neither, they just contains b's. Pump either way.*

Exercise 2.4. *DPDAs*

1. *Give a formal definition of a deterministic pushdown automaton (DPDA). (Note, there are a few different ways of doing this. Some defintions are equivalent, but some are not. However, any defintion that eliminates nondeterminism from PDAs will yield less powerful machines.)*

2. *Now show that the DPDAs defined in part 1 are weaker than PDAs.*

Solution.

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

where:

- Q is a set of states.
- Σ is the input alphabet.
- Γ is the tape alphabet.
- $\delta : Q \times \Sigma \times \Gamma_\varepsilon \to Q \times T$ is the transition function, where $T \subseteq \Gamma^*$ such that, if any string $w \in T$, then there is no string $wx : |x| \geq 1$ in T.
- $q_0 \in Q$ is the start state.
- $F \subseteq Q$ is the set of accept states.

where $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$.

What's the difference from PDAs? In a PDA,

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$$

This DPDA computes by reading the next character in the input and then, based on the current state and possibly the top symbol on the stack, transitioning to one particular state and writing a string from T to the stack. By the definition of T, no string in T can be a prefix of another. The DPDA accepts if it halts in a final state.

Proof:

1. DPDAs are special cases of PDAs, so not more powerful. (PDAs can only push symbols, not strings, but by using multiple states, we could achieve the same effect.)

2. Closed under complement: Flip the final states. By definition, every string we read in only has one path it can possibly go through in the DPDA. So it always ends in the same state. Noting that CFLs are not closed under complement gives the result.