Problem Set 7

How much firepower do context-free languages have? What are their limits? And just how awesome are PDAs? In this problem set, you'll get to find out!

Start this problem set early. It contains seven problems (plus one survey question and **three** extra credit problems), several of which require a fair amount of thought. I would suggest reading through this problem set at least once as soon as you get it to get a sense of what it covers.

As much as you possibly can, please try to work on this problem set individually. That said, if you do work with others, please be sure to cite who you are working with and on what problems. For more details, see the section on the honor code in the course information handout.

In any question that asks for a proof, you **must** provide a rigorous mathematical proof. You cannot draw a picture or argue by intuition. You should, at the very least, state what type of proof you are using, and (if proceeding by contradiction, contrapositive, or induction) state exactly what it is that you are trying to show. If we specify that a proof must be done a certain way, you must use that particular proof technique; otherwise you may prove the result however you wish.

If you are asked to prove something by induction, you may use weak induction, strong induction, the well-ordering principle, structural induction, or well-founded induction. In any case, you should state your base case before you prove it, and should state what the inductive hypothesis is before you prove the inductive step.

As always, please feel free to drop by office hours or send us emails if you have any questions. We'd be happy to help out.

This problem set has 125 possible points. It is weighted at 7% of your total grade. The earlier questions serve as a warm-up for the later problems, so do be aware that the difficulty of the problems does increase over the course of this problem set.

Good luck, and have fun!

Due Friday, November 18th at 2:15 PM

Problem One: Designing CFGs (20 Points)

Below are a list of alphabets and languages over those alphabets. For each language, design a context-free grammar that generates that language.

i. For the alphabet $\Sigma = \{$ `0`, `1`, `2` $\}$, write a CFG for the language $L = \{ w \mid w$ contains `00` as a substring $\}$

ii. For the alphabet $\Sigma = \{$ `0`, `1`, `2` $\}$, write a CFG for the language $L = \{ w \mid w$ does not contain both `0` and `1`. $\}$

iii. For the alphabet $\Sigma = \{$ `0`, `1`, `2` $\}$, write a CFG for $L = \{ \texttt{0}^i\texttt{1}^j\texttt{2}^k \mid i, j, k \in \mathbb{N} \wedge (i = j \vee i = k) \}$

iv. Suppose that you want to write a context-free grammar that describes function prototypes in C or C++. Let $\Sigma = \{$ `int`, `double`, `(`, `)`, `","`, `name`, `;` $\}$, where `name` is a symbol that represents the name of some function or variable. Let $L = \{ w \mid w$ is a valid function prototype $\}$. So, for example, the following would all be valid function prototypes:

```
int name();
double name(int name);
double name(double name, int name, double name);
```

Assuming that each argument to a function must have a name (though in C and C++ names are optional), write a CFG that generates L. Functions may take any number of arguments.

Problem Two: Designing PDAs (20 Points)

Below are a list of alphabets and languages over those alphabets. For each language, design a pushdown automaton that recognizes the given language. In each case, **please provide a brief description** of what each state in your PDA does, along with a short description of how the automaton works as a whole.

To specify your PDA, you may use a state-transition diagram (as done in lecture), a transition table (as found in Sipser), or a seven-tuple. You may use either the style of PDA covered in lecture (where the stack begins with an explicit start symbol and transitions may push strings of characters atop the stack), or the type covered in Sipser (which lacks these features).

i. Let $\Sigma = \{$ `1`, `≥` $\}$ and let $GE = \{ \texttt{1}^m \texttt{≥} \texttt{1}^n \mid m, n \in \mathbb{N} \wedge m \geq n \}$. That is, *GE* consists of strings of two numbers written in unary and separated by a `≥` character, where the first unary number is at least as large as the second unary number. In lecture, we saw a nondeterministic PDA for *GE*. Now, construct a **deterministic** PDA for *GE* (recall that a DPDA is a PDA in which for any combination of a state, input symbol, and stack symbol there is at most one transition that can be followed).

ii. Let $\Sigma = \{$ `0`, `1` $\}$ and let $THIRD = \{ \texttt{0}^n\texttt{1}^{3n} \mid n \in \mathbb{N} \}$. Design a (possibly nondeterministic) PDA that accepts *THIRD*.

iii. Let $\Sigma = \{$ `0`, `1` $\}$ and let $THRICE = \{ \texttt{0}^{3n}\texttt{1}^n \mid n \in \mathbb{N} \}$. Design a (possibly nondeterministic) PDA that accepts *THRICE*.

Problem Three: The Complexity of Addition (12 Points)

On the previous problem set, we began addressing the question

How hard is it to add two numbers?

We will now directly answer that question. Consider the language *ADD* = $\{ \mathtt{1}^m \mathtt{+} \mathtt{1}^n \mathtt{=} \mathtt{1}^{m+n} \mid m, n \in \mathbb{N} \}$ over the alphabet { **1**, **+** , **=** }. That is, *ADD* consists of strings encoding two unary numbers and their sum.

i. Write a context-free grammar for *ADD*. This proves that *ADD* is context-free.

ii. Design a **deterministic** PDA that recognizes *ADD* (recall that a DPDA is a PDA in which for any combination of a state, input symbol, and stack symbol there is at most one transition that can be followed). Provide a brief description of what each state in your DPDA does, along with a short description of how the automaton works as a whole. This proves that *ADD* is not only a context-free language, but also a deterministic context-free language.

Problem Four: The Complexity of Pet Ownership (12 Points)

On the previous problem set, we began answering the question

How hard is it to walk your dog without a leash?

Now, armed with context-free languages, we'll give a formal answer to this question.

Let Σ = { **Y**, **D** }, where **Y** represents you moving one unit forward and **D** represents your dog moving one step forward. Then a string of **Y**s and **D**s represents you and your dog going for a walk. For example, in the string **YYDY**, you end up two steps ahead of your dog, while in the string **DDDDYY**, your dog takes off and ends up two steps ahead of you (perhaps there was some other dog it wanted to meet).

If we consider the language of strings representing walks where you and your dog end up at the same location, we get the language *DOGWALK* = { w | w has the same number of **Y**s and **D**s }. This language is not regular, as you proved on the last problem set.

i. Write a context-free grammar that generates *DOGWALK*. This proves that *DOGWALK* is context-free.

ii. Design a **deterministic** PDA that recognizes *DOGWALK*. Provide a brief description of what each state in your DPDA does, along with a short description of how the automaton works as a whole. This proves that *DOGWALK* is not only a context-free language, but also as deterministic context-free language.

Problem Five: The Complexity of Exponentiation (16 Points)

On the previous problem set, we began addressing the question

How hard is it to check whether a number is a perfect power of two?

A number is a power of two if it can be written as 2^n for some natural number n. Consider the language *POWER2* = $\{ 1^{2^n} \mid n \in \mathbb{N} \}$ over the simple alphabet $\Sigma = \{ \texttt{1} \}$. That is, L contains all strings whose lengths are a power of two. For example, the smallest strings in L are `1`, `11`, `1111`, and `11111111`.

On the previous problem set, you proved that *POWER2* is not regular using the pumping lemma for *regular* languages. Now, using the pumping lemma for context-free languages, prove that it is not context-free either. *(Hint: As with last time, you may want to use the fact that $n < 2^n$ for all $n \in \mathbb{N}$)*

Problem Six: The Complexity of String Searching (24 Points)

On the previous problem set, we began addressing the question

How hard is it to search a string for a substring?

Given a string to search for (called the **pattern**) and a string in which the search should be conducted (called the **text**), we want to determine whether the pattern appears in the text. To encode this as a language problem, we let $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{?}\}$ and encoded questions of the form "does pattern string *p* appear in text *t*" as the string *p*`?`*t*. For example:

"Does `0110` appear in `1110110` ?"	would be encoded as	`0110?1110110`
"Does `11` appear in `0001` ?"	would be encoded as	`11?0001`
"Does ε appear in `1100` ?"	would be encoded as	`?1100`

Let the language *SEARCH* = $\{ p\texttt{?}t \mid p, t \in \{\texttt{0}, \texttt{1}\}^* \text{ and } p \text{ is a substring of } t \}$. On the last problem set, you proved that *SEARCH* is not regular using the pumping lemma for *regular* languages. Now, using the pumping lemma for context-free languages, prove that *SEARCH* is not context-free either.

As a hint, the pattern and text string you show cannot be pumped must use both `0`s and `1`s. In fact, if you restrict the pattern and text strings to strings consisting solely of `0`s, you get the language

$$LE = \{ \texttt{0}^n\texttt{?}\texttt{0}^m \mid n, m \in \mathbb{N} \wedge n \leq m \}$$

which *is* context-free.

Problem Seven: Uncertainty about Ambiguity (16 Points)

In this question, you'll explore some of the properties of ambiguous grammars. To begin with, consider the following (ambiguous) grammar that describes all strings of balanced parentheses:

- $P \rightarrow (P)$
- $P \rightarrow PP$
- $P \rightarrow \varepsilon$

You may want to play around with this grammar a bit before answering these questions.

i. Show that this grammar is ambiguous by providing a string and two different parse trees for it.

ii. Rewrite this grammar so that it is not ambiguous. Explain, but do not formally prove, why your new grammar is unambiguous.

iii. Prove or disprove: If a grammar G is ambiguous, then there is no DPDA that accepts $L(G)$.

Problem Eight: Course Feedback (5 Points)

We want this course to be as good as it can be, and we'd really appreciate your feedback on how we're doing. For a free five points, please answer the following questions. We'll give you full credit no matter what you write (as long as you write something!), but we'd appreciate it if you're honest about how we're doing.

i. How hard did you find this problem set? How long did it take you to finish?

ii. Does that seem unreasonably difficult or time-consuming for a five-unit class?

iii. Did you attend Monday's problem session? If so, did you find it useful?

iv. How is the pace of this course so far? Too slow? Too fast? Just right?

v. Is there anything in particular we could do better? Is there anything in particular that you think we're doing well?

Submission Instructions

There are four ways to submit this assignment:

1. Hand in a physical copy of your answers at the start of class.

2. Submit a physical copy of your answers in the filing cabinet in the open space near the handout hangout in the Gates building. If you haven't been there before, it's right inside the entrance labeled "Stanford Venture Fund Laboratories." There will be a clearly-labeled filing cabinet into which you can submit your homework.

3. Submit a physical copy of your answers in the drop-off box in the Gates basement. We'll send out an email announcement about the location of this box on the night of Friday, November 18.

4. Send an email with an electronic copy of your answers to **cs103@cs.stanford.edu**

Extra Credit Problem: Infinite PDAs (5 Points Extra Credit)

In our definition of PDAs, one of the restrictions is that the transition function δ never maps any combination of a state, input symbol, and stack symbol to an infinite set of transitions (where each transition is a combination of a state and a string of stack symbols). It turns out that if we relax this restriction and allow PDAs that have infinitely many transitions defined for some state/input/stack combinations, the resulting PDA (which we'll call an *IPDA* for *infinite PDA*) can accept *any* language.

Let Σ be an alphabet and $L \subseteq \Sigma^*$ be an arbitrary language over Σ. Prove that there is some IPDA that accepts L by defining a seven-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ for it. In addition to the seven-tuple, provide an intuitive explanation for how your IPDA works. Note that in an IPDA there may be infinitely many transitions defined for some state/input/stack combination, but $|Q|$ and $|\Gamma|$ must be finite.

Extra Credit Problem: Crossover Closure Properties (5 Points Extra Credit)

As mentioned in lecture, CFLs are not closed under intersection, meaning that if L_1 and L_2 are CFLs, then $L_1 \cap L_2$ is not necessarily a CFL. However, if L_1 is a CFL and L_2 is a regular language, then $L_1 \cap L_2$ is a CFL. That is, if we begin with a CFL and filter the language down to just those strings that are contained in a regular language, the result is context-free. Prove this.

Extra Credit Problem: Palindromes are Nondeterministic (10 Points Extra Credit)

In lecture we mentioned that the language $L = \{ w \mid w \text{ is a palindrome} \}$ over the alphabet $\Sigma = \{ 0, 1 \}$ is context-free, but not **deterministic** context-free. That is, there is an NPDA that recognizes L, but there is no DPDA that recognizes L. Formally prove this result.