

CS 273, Spring 2007

Exam 2 Solutions

Problem 1: Grammar design (4 points)

Give a CFG for the language

$$L = \{a^i b^j \# a^k b^l \mid i < j, k > l\}$$

Solution:

$$\begin{aligned} S &\rightarrow X \# Y \\ X &\rightarrow Xb \mid Eb \\ Y &\rightarrow aY \mid aE \\ E &\rightarrow aEb \mid \epsilon \end{aligned}$$

The variable E generates all strings of the form $a^n b^n$. X and Y add some (non-zero) number of extra b 's and a 's (respectively).

Problem 2: PDA design (6 points)

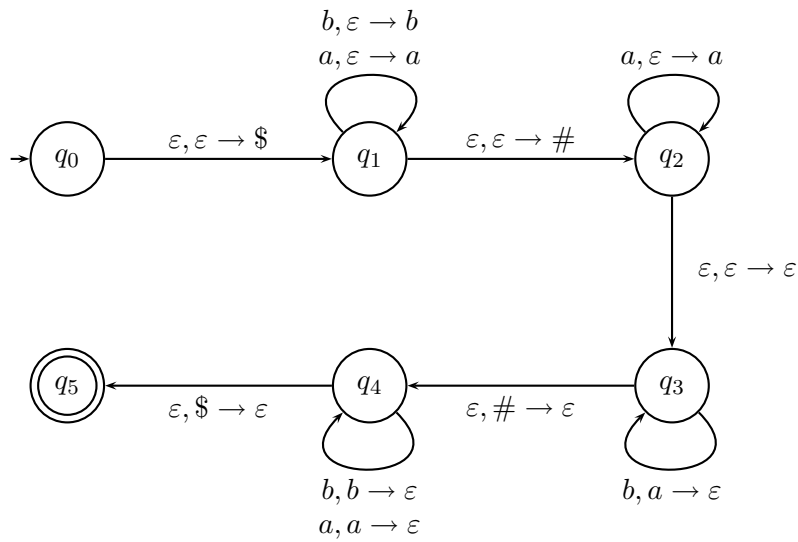
No proof or explanation is required for this problem. But, if your answer is incorrect, sensible explanations may increase your partial credit.

Give the state diagram for a PDA that recognizes the following language.

$$L = \{wa^n b^n w^R \mid w \in \{a, b\}^*, n \geq 0\}$$

Recall that x^R is the string reversal of x .

Solution:



Problem 3: Pumping lemma for regular languages (6 points)

Let $L = \{w \in \{a, b\}^* \mid \text{number of } a\text{'s in } w \text{ is greater than or equal to three times the number of } b\text{'s in } w\}$. Prove that L is not regular by filling in the missing parts of the following pumping lemma proof.

Suppose that L were regular. Let p be the constant given by the pumping lemma.

Consider the string

Solution: $w_p = b^p a^{3p}$

Because $w_p \in L$ and $|w_p| \geq p$, there must exist strings x , y , and z such that $w_p = xyz$, $|xy| \leq p$, $|y| > 0$, and $xy^i z \in L$ for every $i \geq 0$.

Solution: Since $|xy| \leq p$, y must consist entirely of b 's. Suppose that $y = b^k$.

Because $xy^i z \in L$ for every $i \geq 0$, we know in particular that $xy^2 z$ is in L .

$xy^2 z = b^{p+k} a^{3p}$. Since $|y| > 0$, $k > 0$. So $p + k > p$. But this means that $b^{p+k} a^{3p}$ can't be in L because it has too many b 's.

Since $b^{p+k} a^{3p}$ isn't in L , we have a contradiction. Therefore, L must not have been regular.

Comments: You can also use $w_p = a^{3p} b^p$. In this case, you need to pump down, i.e. set $i = 0$. You could write out more details about the form of the substrings, e.g. that x also consists entirely of b 's, $x = b^j$ and $z = b^{p-k-j} a^{3p}$.

Most people picked a reasonable string w_p . However, a number of people got into trouble because they didn't realize that y had to lie entirely within the initial string of b 's.

Common small mistakes included: claiming that z must consist entirely of a 's, not using the three pumping lemma conditions to justify your steps, e.g. claiming that y must consist entirely of b 's without referring to the fact that $|xy| \leq p$.

Problem 4: Induction and parse trees (6 points)

Let $G = (V, \Sigma, R, S)$ be a CFG whose rules all have one of these two forms

1. $A \rightarrow BC$
2. $A \rightarrow abc,$

where $A, B,$ and C are all variables from V and $a, b,$ and c are all terminals from Σ .

Prove that if w is a string in $L(G)$, then $|w|$ is a multiple of 3. **You must use strong induction.** Your induction variable can be either the number of nodes in the parse tree for w or the length of a derivation for w .

(a) State your inductive hypothesis precisely.

Solution: If w is a string in $L(G)$ which is derived from a variable V in $\leq k$ steps, $|w|$ is a multiple of 3.

(b) Your proof:

Solution:

Base: Suppose that w is derived using only one step. Since rules of type (1) have variables on their righthand sides, the single step must expand the variable V using a rule of type (2). These rules produce exactly three terminals.

Induction: Suppose that the induction hypothesis is true for $k, k \geq 1$. Suppose that V is a variable and w is a string $L(G)$ which is derived from V in $k + 1$ steps. We need to show that $|w|$ is a multiple of 3.

The first step in the derivation must use a rule of type (1). So it expands the starting variable V into two other variables, call them B and C . The rest of the derivation expands B into some string x and C into some string y , using no more than k steps in each case. By the induction hypothesis, this means that $|x|$ and $|y|$ are multiples of 3.

But $w = xy$. So, since $|x|$ and $|y|$ are multiples of 3, $|w|$ must also be a multiple of 3.

Comments: There's several different theories about which bit is the "inductive hypothesis." I wasn't picky about this. Similarly, really nailing down the details of this proof requires being clear about what can be used as the starting point of a derivation, but I didn't expect to see this done right under test conditions.

Notice that the strong induction proof removes the first step in the derivation, extracting two shorter derivations. A similar proof using the parse tree would remove the top node of the tree, producing two smaller child trees. Most people lost points by trying to prove this using weak induction, e.g. by removing the last step in the derivation or adding stuff to the side or bottom of the parse tree. This works for binary storage trees but doesn't work for parse trees, because the new tree might not actually be consistent with the grammar, depending on what rules are in G .

Problem 5: Short explanation I (9 points)

The answers to these problems should be short and not complicated.

(a) Prove that if CFLs are closed under set difference then they are closed under complement. That is, if for every two CFLs L_1 and L_2 the language $L_1 - L_2$ is a CFL then for every CFL L its complement \bar{L} is also a CFL.

Solution: We want to show that given a CFL L , its complement \bar{L} is also a CFL. We can express the complement as:

$$\bar{L} = \Sigma^* - L$$

Σ^* is regular, and thus it is also a CFL. By our assumption, CFLs are closed under set difference, and thus \bar{L} is a CFL.

Note: It is irrelevant that CFLs aren't actually closed under set difference.

(b) State the pumping lemma for context-free languages.

Solution: See Theorem 2.34, p. 123 in Sipser.

(c) Eliminate unit productions from following grammar.

$$\begin{aligned}S &\rightarrow AB \mid BB \\A &\rightarrow DDF \mid B \\B &\rightarrow D \mid FF \mid A \\D &\rightarrow d \\F &\rightarrow f\end{aligned}$$

Solution:

The unit productions are $A \rightarrow B$, $B \rightarrow D$, and $B \rightarrow A$. Eliminating them, we get:

$$\begin{aligned}S &\rightarrow AB \mid BB \\A &\rightarrow DDF \mid d \mid FF \\B &\rightarrow d \mid FF \mid DDF \\D &\rightarrow d \\F &\rightarrow f\end{aligned}$$

Problem 6: Tuple notation (5 points)

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA. Using your best formal notation, explain precisely how to construct a new PDA N which recognizes the same language as M but empties its stack immediately before accepting.

Solution:

The procedure is one of the steps in converting a PDA into a “standard” format that shows the equivalence of PDAs and CFGs. We add three new states to M . We add a new start state q_s that pushes a new stack symbol $\$$ onto the stack and then passes control to M 's start state q_0 . From the final states of M we take an ϵ -transition to a new cleanup state q_c that pop's all stack symbols from Γ and then pops the $\$$ and moves to a new final state q_f . The construction in tuple notation is given below. $N = (Q', \Sigma, \Gamma', \delta', q_s, F')$ where

$$Q' = Q \cup \{q_s, q_c, q_f\}$$

$$\Gamma' = \Gamma \cup \{\$\}, \text{ assuming that } \$ \notin \Gamma.$$

$$F' = \{q_f\}.$$

The new transition function δ' is given below.

$$\begin{aligned} \delta'(q, a, b) &= \delta(q, a, b) & q \in Q, a \in \Sigma_\epsilon, b \in \Gamma_\epsilon \text{ and not } (q \in F, a = \epsilon, b = \epsilon) \\ \delta'(q, a, b) &= \delta(q, a, b) \cup \{(q_c, \epsilon)\} & q \in F, a = b = \epsilon. \\ \delta'(q_s, \epsilon, \epsilon) &= \{(q_0, \$)\} \\ \delta'(q_c, \epsilon, b) &= \{(q_c, \epsilon)\} & b \in \Gamma \\ \delta'(q_c, \epsilon, \$) &= \{(q_f, \epsilon)\}. \end{aligned}$$

Common mistakes:

- Some of you assumed that a $\$$ is always pushed on to the stack. Since M is a generic PDA you cannot assume this.
- Some of you did not add a new clean up state and cleaned the stack in the final states of M by adding transitions to pop symbols off. This can lead to incorrect behaviour. One can create examples of PDAs that have transitions out of final states. In such examples popping the stack in the final state can make the machine accept strings that are not accepted by the original machine M .
- Some of you added a cleanup state “before” the final states of M . This seems to have been guided by the wording of the problem. This is also incorrect because final states might have transitions out of them.

Problem 7: CFG to PDA and PDA to CFG (6 points)

(a) Give a PDA for the grammar below, using the automatic conversion procedure in the book and lecture. Your machine is allowed to push strings of length greater than 1 onto the stack in one transition.

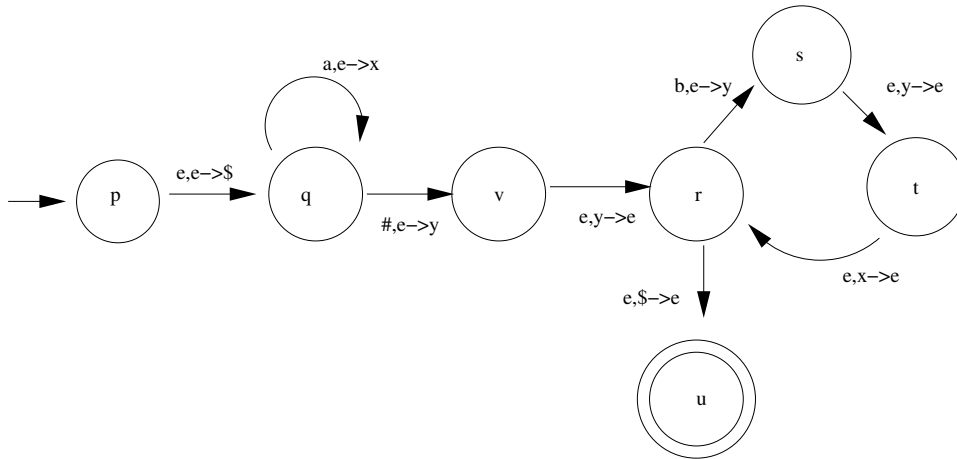
$$S \rightarrow ABa \mid bAA \mid \epsilon$$

$$A \rightarrow aA \mid B \mid a$$

$$B \rightarrow b$$

Solution: This is a mechanical problem to test knowledge of a given algorithm. The grammar is not really relevant. Some of you tried to interpret the grammar or simplify it but that was not necessary. You simply had to follow the construction given in Sipser, pages 115-118.

(b) For the PDA whose state diagram is shown below (ϵ on the transitions means ϵ), give all rules with left hand side A_{qr} that would be produced by the automatic procedure for converting to a CFG. (Notice that this PDA is already in standard format.)



Solution: Note that the PDA was modified during the exam as shown above. As we stressed in class review, the procedure is mechanical and follows the rules given in Sipser, pages 119-120. The rules for A_{qr} are the following:

$$A_{qr} \rightarrow \#A_{qv} \mid aA_{qt}$$

$$A_{qr} \rightarrow A_{qx}A_{xr}, \text{ for each } x \in \{p, q, r, s, t, u, v\}.$$

Problem 8: True/false (8 points)

Label each of the following claims as true or false. No explanation is required but your true/false markings must be clearly legible.

- (a) Every finite language is context free.

Solution: True. In fact, all finite languages are regular.

- (b) Given two context-free grammars G_1 and G_2 we can produce another CFG G such that $L(G) = L(G_1) - L(G_2)$.

Solution: False. Context-free languages aren't closed under subtraction because they aren't closed under set complement. Set up G_1 so it generates all of Σ^* .

- (c) If a grammar G is unambiguous, each string has exactly one derivation.

Solution: False. There can be several derivations corresponding to a single parse tree (though only one leftmost derivation).

- (d) The transition function for a PDA is a function from $Q \times \Sigma \times \Gamma$ to $\mathbb{P}(Q \times \Gamma)$.

Solution: False. The Σ 's and Γ 's need subscript epsilons.

- (e) This language is context-free: $L = \{wxw \mid w, x \in \{a, b\}^*\}$

Solution: True. If you set w to be the empty string, you can see that this language includes all of Σ^* .

- (f) There are two infinite context free languages L_1 and L_2 which have no bijection between them.

Solution: False. All context-free languages are either finite or countable. There's a bijection between any two countable sets.

- (g) If M is a PDA, then you can make a new PDA accepting the complement of $L(M)$ by making all of M 's final states non-final and its non-final states final.

Solution: False. This trick doesn't work for non-deterministic machines. In fact, context-free languages aren't even closed under complementation.

- (h) If L_1 is a context-free language and L_2 is a regular language, then $L_1 - L_2$ is a context-free language.

Solution: True. You can rewrite this as the intersection of L_1 with a regular language, because regular languages are closed under complementation. Context-free languages are closed under intersection with regular languages.