

CS273: Theory of Computation, Summer 2008

Homework 5 Solutions

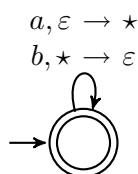
Due 3:00PM Monday, July 14, 2008 at SC 3303

Revision due 3:00PM Friday, July 18 2008 at SC 3303

1. [30 points; 10 per part] (Parts from Sipser.) Give state diagrams of PDAs for each of the following languages.

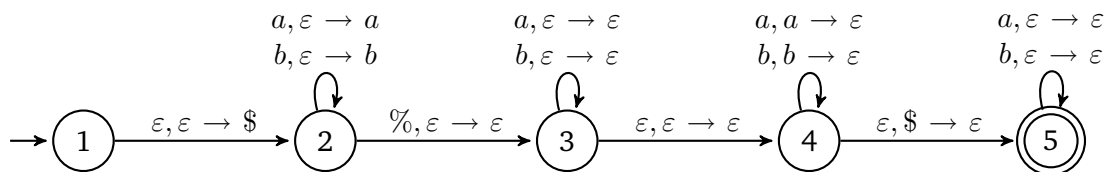
(a) $\{w \in \{a, b\}^* \mid \text{each prefix of } w \text{ has at least as many } a\text{'s as } b\text{'s}\}$. [Hint: In problem 3, you will want this PDA to have just 1 state.]

Solution: This PDA pushes an \star onto the stack each time it reads an a , and pops off a \star each time it reads a b . Thus, the number of \star 's on the stack is equal to the number of a 's read so far minus the number of b 's read so far. If this quantity ever becomes negative, there are not enough \star 's to pop and the PDA rejects as it has no legal way to continue the computation.



(b) $\{w\%x \mid w, x \in \{a, b\}^* \text{ and } w^R \text{ is a substring of } x\}$ (In this problem, the alphabet is $\Sigma = \{a, b, \%\}$.)

Solution: This PDA first pushes a $\$$ onto its stack. Next, in state 2, it reads and pushes w onto its stack. Once it reads $\%$, it transitions from state 2 to state 3, in preparation to read x . In state 3, the PDA reads characters of x . At some point, the PDA nondeterministically guesses that w^R is about to appear as a substring and transitions from state 3 to state 4. After taking this transition, the PDA verifies its guess in state 4 by popping w^R off the stack and comparing each symbol of w^R with a symbol of input. If there are any disagreements, then this particular branch of the nondeterministic computation rejects. Finally, if the PDA sees a $\$$ on the top of the stack, it knows it has finished verifying that w^R has occurred as a substring of x , so it transitions to state 5 and continues reading the remainder of the input.

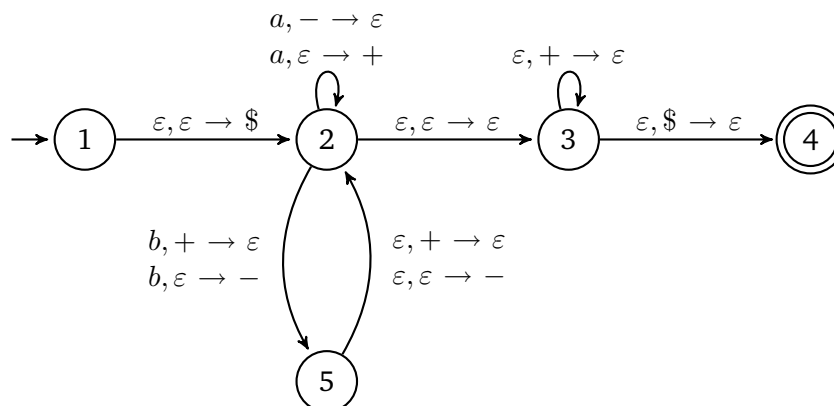


(c) $\{w \in \{a, b\}^* \mid \#a(w) \geq 2\#b(w)\}$

Solution: The PDA works by storing an integer n on its stack. The PDA's tape alphabet is $\{+, -\}$, and the value of n is simply the number of $+$'s on the stack minus the number of $-$'s on the stack. (The technique of using the stack to store a counter is helpful in designing PDAs for many CFLs; see, for example, the honors problem on homework 6.)

At any point in time, our PDA's counter stores $\#a(x) - 2\#b(x)$, where x is the part of the input that the PDA has read so far. Each time the PDA reads an input symbol, it updates the stack accordingly. That is, upon reading an a , the PDA either adds a $+$ to the stack or erases a $-$ from the stack. Similarly, upon reading a b , the PDA either erases two $+$'s from the stack, adds two $-$'s to the stack, or erases one $+$ and adds one $-$. We allow the PDA to perform any of these updates to the stack, and some of the nondeterministic choices may lead the PDA to mix $+$'s and $-$'s on the stack. The key point is that no matter how the PDA changes its stack, the value of the counter is updated correctly.

The PDA nondeterministically guesses when it has reached the end of the input and transitions to a part of the machine that accepts the input provided that there are no $-$'s on the stack. The state diagram follows.



Note that if the PDA accepts a string w , then after reading w , the stack consists of zero or more $+$'s, so that $\#a(w) - 2\#b(w) \geq 0$, which of course implies that $\#a(w) \geq 2\#b(w)$.

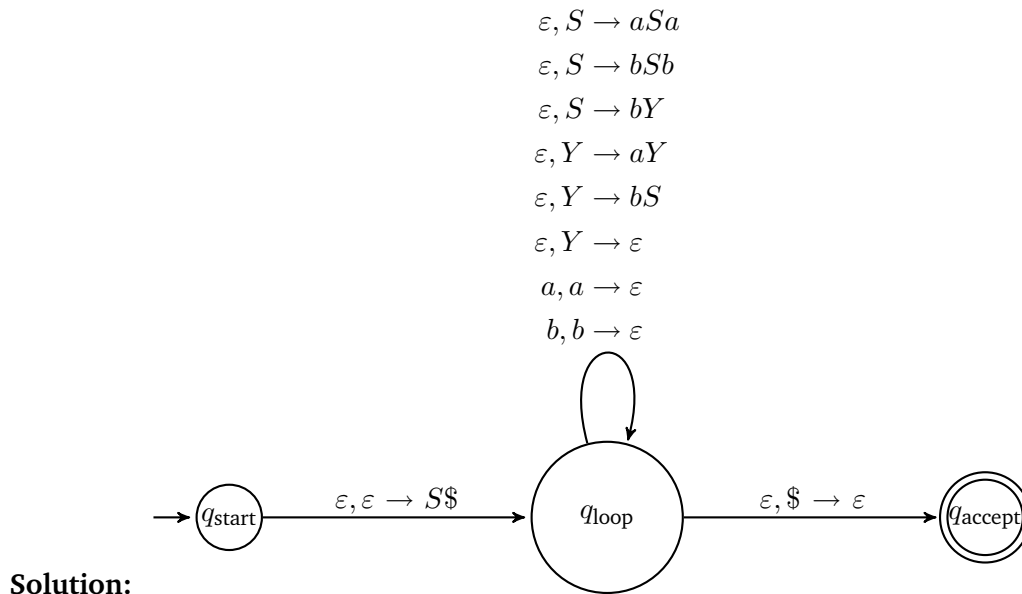
Conversely, if w has twice as many a 's as b 's, then there are choices which cause the PDA to accept. In particular, if the PDA chooses to update its stack so that it does not mix both $+$'s and $-$'s, then after reading w , because $\#w(a) \geq 2\#w(b)$, there will be only $+$'s left on the stack. Thus, the PDA accepts under this branch of the nondeterminism. ■

2. [15 points] Let G be the following CFG, with start symbol S :

$$S \rightarrow aSa \mid bSb \mid bY$$

$$Y \rightarrow aY \mid bS \mid \varepsilon$$

Using the CFG to PDA conversion algorithm, convert G to a GPDA (generalized PDA). You do not need to convert your GPDA to a PDA, and you may express your GPDA with a state diagram. (Please be sure to use the conversion algorithm; no credit for an ad hoc GPDA.)



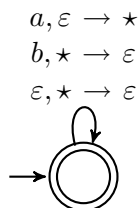
3. PDA to CFG conversion.

(a) [5 points] Convert your PDA from 1(a) to an NPDA that has at most 3 states.

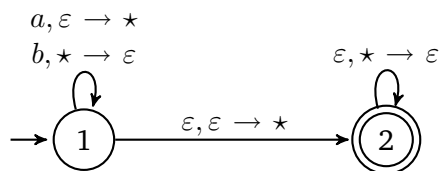
Solution: Recall that an NPDA is a PDA with three special properties:

- i. There is one accept state,
- ii. If the PDA accepts a string w , it can do so with an empty stack, and
- iii. Each transition either pushes or pops, but not both.

In fact, the PDA presented in 1(a) already has properties (1) and (3). It can be given property (2) by also allowing the PDA to pop off excess \star 's on the stack.



Unfortunately, this NPDA is too simple to provide for a very instructive example in part (b), so we will use the following NPDA instead.



- (b) [10 points] Using the NPDA to CFG conversion algorithm, convert your NPDA from part (a) to an equivalent CFG. (Please be sure to use the conversion algorithm; no credit for an ad hoc CFG.)

Solution: The rules from case (1) are as follows.

$$A_{11} \rightarrow A_{11}A_{11} \mid A_{12}A_{21}$$

$$A_{12} \rightarrow A_{11}A_{12} \mid A_{12}A_{22}$$

$$A_{21} \rightarrow A_{21}A_{11} \mid A_{22}A_{21}$$

$$A_{22} \rightarrow A_{21}A_{12} \mid A_{22}A_{22}$$

The rules from case (2) are as follows.

$$A_{11} \rightarrow aA_{11}b$$

$$A_{12} \rightarrow aA_{12}\varepsilon \mid \varepsilon A_{22}\varepsilon$$

$$A_{21} \rightarrow$$

$$A_{22} \rightarrow$$

Finally, the rules from case (3) are as follows.

$$A_{11} \rightarrow \varepsilon$$

$$A_{12} \rightarrow$$

$$A_{21} \rightarrow$$

$$A_{22} \rightarrow \varepsilon$$

Putting all three cases together and removing extraneous occurrences of ε , we obtain our grammar. The start variable is A_{12} .

$$A_{11} \rightarrow A_{11}A_{11} \mid A_{12}A_{21} \mid aA_{11}b \mid \varepsilon$$

$$A_{12} \rightarrow A_{11}A_{12} \mid A_{12}A_{22} \mid aA_{12} \mid A_{22}$$

$$A_{21} \rightarrow A_{21}A_{11} \mid A_{22}A_{21}$$

$$A_{22} \rightarrow A_{21}A_{12} \mid A_{22}A_{22} \mid \varepsilon$$

4. [15 points] (From Sipser.) Let $C = \{x\%y \mid x, y \in \{a, b\}^* \text{ and } x \neq y\}$. Show that C is a context-free language. (In this problem, the alphabet is $\Sigma = \{a, b, \%\}$.)

Solution: We begin by expressing C as the union of two languages that are more easily seen to be context free. Let

$$C_1 = \{x\%y \mid |x| \neq |y|\}$$

$$C_2 = \{x_1x_2x_3 \cdots x_m\%y_1y_2 \cdots y_n \mid x_i, y_j \in \{a, b\} \text{ and } x_j \neq y_j \text{ for some } j\}.$$

Note that $C = C_1 \cup C_2$, even though there are some strings (such as $aab\%ba$) that are in both C_1 and C_2 .

Because the context-free languages are closed under union, it is enough to show that C_1 and C_2 are both context-free. It is relatively straightforward to construct a PDA that recognizes C_1 ; the PDA uses its stack to count $|x|$ and compares it with $|y|$. However, it is even simpler to construct a CFG that generates C_1 :

$$S \rightarrow ZSZ \mid ZA \mid BZ$$

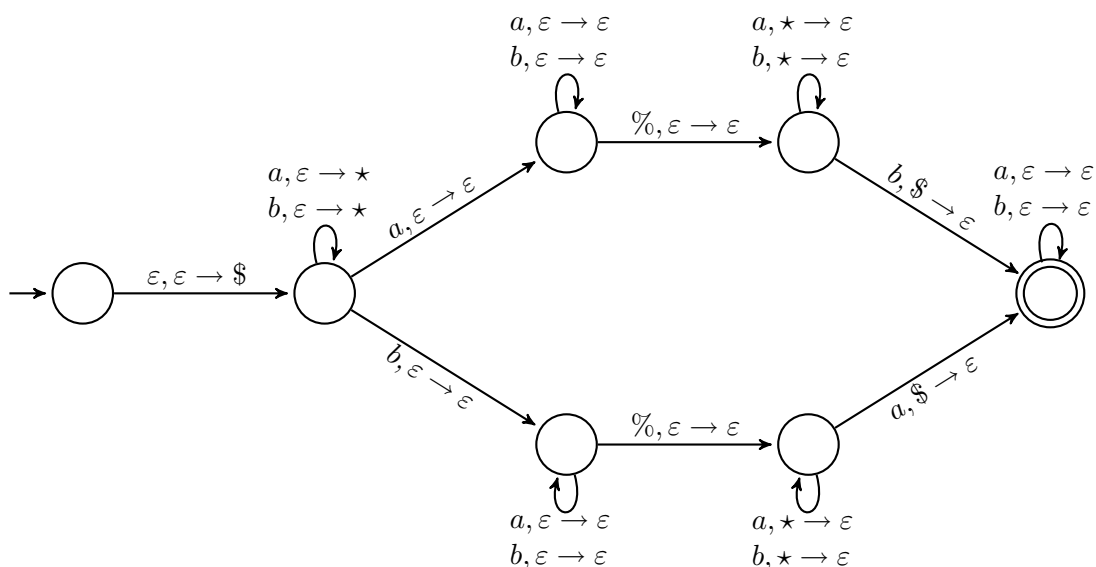
$$A \rightarrow ZA \mid \%$$

$$B \rightarrow BZ \mid \%$$

$$Z \rightarrow a \mid b$$

First note that Z simply generates a single character (a or b). The grammar works by first generating equal number of characters to the left and right from the middle, followed by one or more characters to the left (if the derivation uses $S \rightarrow ZA$) or one or more characters to the right (if the derivation uses $S \rightarrow BZ$). Finally, the middle location from which characters are generated is replaced by $\%$. Because there is a CFG that generates C_1 , we know that C_1 is context-free.

We must show that C_2 is also context free. This time, we design a PDA¹ The PDA works by reading input symbols and pushing \star 's on the stack. It nondeterministically guesses when it is about to read x_j . It uses its states to remember the value of x_j . It continues to read input characters until it reads a $\%$, at which time it begins counting to j by reading input characters and popping from the stack. The PDA compares y_j with x_j and accepts if they are different characters.



¹Here also, a CFG for C_2 would not be too complex — but perhaps the task of constructing a CFG for C_2 is more difficult than constructing a PDA.

Because there is a PDA that recognizes C_2 , we know that C_2 is context-free. Because CFLs are closed under union, $C = C_1 \cup C_2$ is context-free. ■

5. [15 points] (From Sipser.) Let G be a CFG in Chomsky normal form that contains b variables. Show that if G generates any string of length more than² 2^b , then $L(G)$ is infinite.

Solution: This proof copies many of the ideas in the proof of the pumping lemma for context free languages.

Let s be a string of length more than 2^b generated by G , and consider a parse tree for s . Because G is in CNF, each node in the parse tree has at most two children. It follows that the parse tree must have depth larger than b . (Indeed, if the parse tree had depth at most b , then there could be at most 2^b leaves, contradicting that s has length more than 2^b .)

Because the parse tree has depth at least $b+1$, it contains a path with at least $b+1$ variable nodes. Because G has only b variables, some variable must be repeated along this path. This repeated variable allows us to split s into 5 parts $s = uvxyz$ with $vy \neq \varepsilon$ and $uv^i xy^i z \in L(G)$ for each i , just as in the proof of the CFL pumping lemma. Because $vy \neq \varepsilon$, the strings $\{uv^i xy^i z \mid i \geq 0\}$ are all distinct and in $L(G)$, and so $L(G)$ is infinite. ■

6. [15 points] (From Sipser.) For any language A , let $\text{SUFFIX}(A) = \{v \mid uv \in A \text{ for some string } u\}$. Show that the class of context-free languages is closed under the SUFFIX operation.

Solution: Let A be a context-free language. We show that $\text{SUFFIX}(A)$ is also a CFL. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA that recognizes A . We show how to use M to construct a PDA N that recognizes $\text{SUFFIX}(A)$. On an input string v , our PDA N must determine if there exists another string u such that uv is in A . The key idea is that N contains two copies of M , although one copy is modified. The first copy of M is modified so that it does not read any symbols of input; the effect is that this modified copy of M nondeterministically guesses a prefix string u and manipulates the stack as though M were reading u . After pretending to read u , the PDA N then ε -transitions to the second copy of M which actually does read the input v . Our PDA N accepts if the second copy of M accepts.

Formally, we define $N = (Q_N, \Sigma, \Gamma, \delta_N, q_N, F_N)$ as follows.

(a) $Q_N = Q \times \{\text{U-PART}, \text{V-PART}\}$

(b)

$$\delta_N((p, \text{U-PART}), \varepsilon, b) = \{((q, \text{U-PART}), c) \mid \text{for some } a \in \Sigma_\varepsilon, (q, c) \in \delta(p, a, b)\} \quad b \in \Gamma$$

$$\delta_N((p, \text{U-PART}), \varepsilon, \varepsilon) = \{((q, \text{U-PART}), c) \mid \text{for some } a \in \Sigma_\varepsilon, (q, c) \in \delta(p, a, \varepsilon)\} \\ \cup \{((p, \text{V-PART}), \varepsilon)\}$$

$$\delta_N((p, \text{V-PART}), a, b) = \{((q, \text{V-PART}), c) \mid (q, c) \in \delta(p, a, b)\} \quad a \in \Sigma_\varepsilon, b \in \Gamma_\varepsilon$$

(c) $q_N = (q_0, \text{U-PART})$

(d) $F_N = \{(q, \text{V-PART}) \mid q \in F\} = F \times \{\text{V-PART}\}$

■

²In fact, the conclusion holds as long as G generates a string of length more than 2^{b-1} .

7. **[Honors; 15 points]** Let M be a PDA with the following special property: there is some constant c such that whenever M is computing, M 's stack contains at most c symbols. Prove that $L(M)$ is regular.

Solution: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA with the listed property. We show that we can simulate M with an NFA. Because the stack of M contains a limited number of symbols, we can simulate the stack with a finite number of states. Let $\mathcal{S} = \{x \in \Gamma^* \mid x \text{ has size at most } c\}$. We construct an NFA whose states are $Q \times \mathcal{S}$. Thus, the current state of the NFA stores the entire configuration of the PDA — both the PDA's current state as well as its stack contents.

Formally, we define an NFA $N = (Q_N, \Sigma, \delta_N, q_N, F_N)$ as follows.

(a) $Q_N = Q \times \mathcal{S}$

(b)

$$\delta_N((p, x), a) = \{(q, x') \in Q_n \mid x = by \text{ and } x' = cy \text{ for some } b, c \in \Gamma_\varepsilon \text{ and some } y \in \Gamma^* \\ \text{and } (q, c) \in \delta(p, a, b)\}$$

(c) $q_N = (q_0, \varepsilon)$

(d) $F_N = F \times \mathcal{S}$

■