Computer Science and Engineering, UCSD | Spring 04
CSE 105: Introduction to the Theory of Computation | **Instructor:** Mihir Bellare
Problem Set 2 Solutions | April 29, 2004

Problem Set 2 Solutions

These are practice problems on the subject of context-free grammars. The relevant chapter in Sipser's book is Chapter 2.

Problem 1 Let A be the language of Exercise 2.4 part (b), page 120 of the text. Give a context-free grammar that generates A and also give the state diagram of a PDA that recognizes A.

We let G be the grammar $G = (V, \Sigma, R, S)$ where

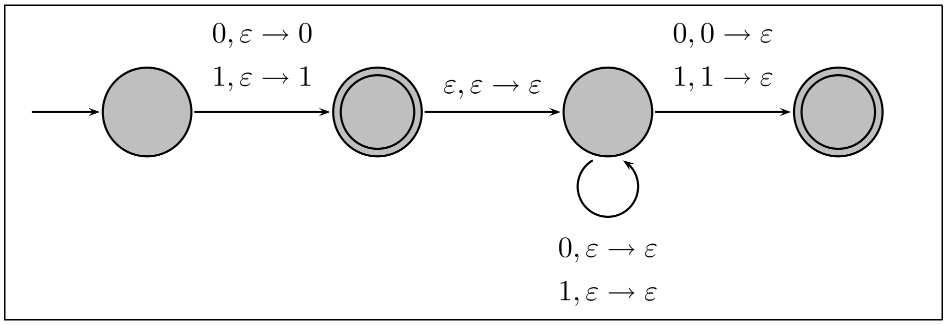
- $V = \{X, S\}$ (there is are two variables, one of which is the start variable S)
- $\Sigma = \{0, 1\}$ (the problem says to assume this for all parts)
- The set R of rules contains the following eight rules:

$$\begin{aligned} S &\rightarrow 0X0 \mid 1X1 \mid 0 \mid 1 \\ X &\rightarrow XX \mid 0 \mid 1 \mid \varepsilon \end{aligned}$$

How can we understand what this is doing? From the variable X, one can derive any string. From S, one can thus derive any string whose first and last symbols are the same. Note that the first and last symbols of the string 0 are the same, as also for the string 1, and thus we make these derivable. On the other hand, the empty string has no symbols and thus its first and last symbols are not the same and it should not be derivable.

When you provide grammers in your solutions in quizzes and exams, use the above template for the specification. Do not merely list the rules: you must also say what is the set V of variables, the set Σ of terminals, and the start variable, and say that the grammer is the 4-tuple of these. If not, you loose points.

Now we need to also provide a PDA that recognizes this language. This is easy, however, since this language is regular. We can design a DFA or NFA that recognizes it. A PDA is an NFA with a stack, so it is easy to convert a DFA or NFA to a PDA: we just program the PDA to ignore its stack. (It is not true that an NFA is PDA, but it is true that an NFA can be easily converted to a PDA that recognizes the same language as the original NFA.) However, this solution isn't much fun, so instead we will provide a PDA that does exploit its stack, and ends up being simpler than the PDA one obtains via a DFA or NFA. The idea is to push the first symbol of the input w on the stack, and then accept if the last symbol of w matches it. The state diagram of the resulting PDA is below.



$0, \varepsilon \to 0$
$1, \varepsilon \to 1$
$\varepsilon, \varepsilon \to \varepsilon$
$0, 0 \to \varepsilon$
$1, 1 \to \varepsilon$
$0, \varepsilon \to \varepsilon$
$1, \varepsilon \to \varepsilon$

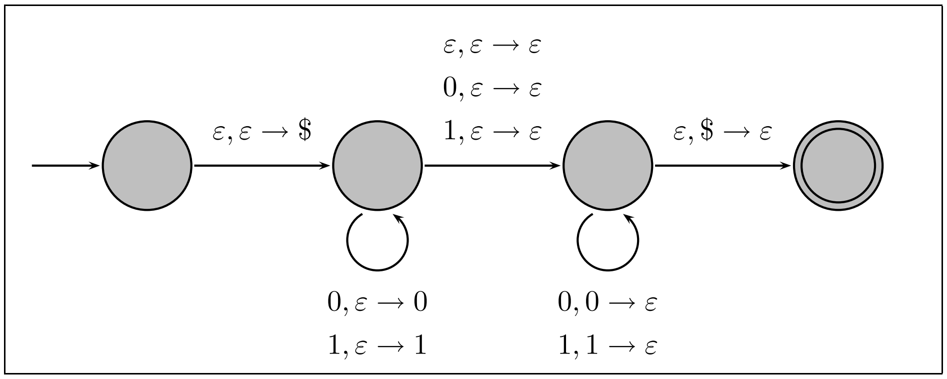
Problem 2 Let A be the language of Exercise 2.4 part (f), page 120 of the text. Give a context-free grammar that generates A and also give the state diagram of a PDA that recognizes A.

We let G be the grammar $G = (V, \Sigma, R, S)$ where

- $V = \{S\}$ (there is only one variable, namely the start variable S)
- $\Sigma = \{0, 1\}$ (the problem says to assume this for all parts)
- The set R of rules contains the following five rules:

$$S \quad \to \quad 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$$

Now we need to also provide a PDA that recognizes this language. Since this language is not regular, our PDA will have to make use of its stack. The idea is that the PDA will push symbols on its stack as it reads the first half of the input string, and then, in reading the second half of the input string, pop the stack symbols and compare them to the read symbols as it goes along. One tricky issue is how the PDA knows when it has reached the middle of the input and should start popping rather than pushing. It will use its non-determinism to guess this. The state diagram of the PDA appears below.



$\varepsilon, \varepsilon \to \$$
$\varepsilon, \varepsilon \to \varepsilon$
$0, \varepsilon \to \varepsilon$
$1, \varepsilon \to \varepsilon$
$\varepsilon, \$ \to \varepsilon$
$0, \varepsilon \to 0$
$1, \varepsilon \to 1$
$0, 0 \to \varepsilon$
$1, 1 \to \varepsilon$

Problem 3 Prove that the class of context-free languages is closed under the concatenation operation.

In answering a question like this you should use the template that we have always used for proving closure properties and was exemplified many times when we studied regular languages.

Given: Languages A_1 and A_2 that are context-free. So there is a CFG $G_1 = (V_1, \Sigma_1, R_1, S_1)$ that generates A_1 and a CFG $G_2 = (V_2, \Sigma_2, R_2, S_2)$ that generates A_2.

Want: To construct a CFG $G = (V, \Sigma, R, S)$ that generates the language $A_1 \cdot A_2$, showing that $A_1 \cdot A_2$ is a context-free language.

Construction: Grammer $G = (V, \Sigma, R, S)$ is defined as follows:

- $V = V_1 \cup V_2 \cup \{S\}$ where S is a new variable
- $\Sigma = \Sigma_1 \cup \Sigma_2$
- $R = R_1 \cup R_2 \cup \{S \rightarrow S_1 S_2\}$ (the set of rules contains the rules of G_1, the rules of G_2, and one new rule, namely the rule $S \rightarrow S_1 S_2$)

In this construction, we assume that $V_1 \cap V_2 = \emptyset$, meaning that the two given grammars have no variables in common. This can always be made true by renaming variables in one of the grammars if necessary.

Correctness of construction: We claim that $L(G) = L(G_1) \cdot L(G_2)$. (Since $L(G_1) = A_1$ and $L(G_2) = A_2$ this means that $L(G) = A_1 \cdot A_2$, so we are done.) In other words, we claim that $S \Rightarrow^* w$ in G if and only if w has the form $w_1 w_2$ for some w_1, w_2 such that $S_1 \Rightarrow^* w_1$ in G_1 and $S_2 \Rightarrow^* w_2$ in G_2. This is true because the first rule applied in a derivation of any w via G is $S \rightarrow S_1 S_2$, and after that the only way the derivation can proceed is to expand S_1 according to G_1 and S_2 according to G_2. The fact that $V_1 \cap V_2 = \emptyset$ is important to ensure that a derivation starting from, say, S_1, can only use rules in G_1 and not use rules in G_2, and vice-versa.

Problem 4 Let A be the language of Exercise 2.6 part (b), page 120 of the text. Give a context-free grammar that generates A and also give the state diagram of a PDA that recognizes A.

We let G be the grammar $G = (V, \Sigma, R, S)$ where

- $V = \{S, T_1, T_2, T_3, D\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- The set R of rules contains the following 14 rules:

$$\begin{aligned}
S &\rightarrow T_1 \mid T_2 \mid T_3 \\
T_1 &\rightarrow \mathtt{a} \mid \mathtt{a}T_1 \mid \mathtt{a}T_1\mathtt{b} \\
T_2 &\rightarrow \mathtt{b} \mid T_2\mathtt{b} \mid \mathtt{a}T_2\mathtt{b} \\
T_3 &\rightarrow D\mathtt{ba}D \\
D &\rightarrow DD \mid \mathtt{a} \mid \mathtt{b} \mid \varepsilon
\end{aligned}$$

Here's how this works. We want the grammar to generate all strings *not* consisting of some number of as followed by the same number of bs. To simplify the task, we break it up into parts depending

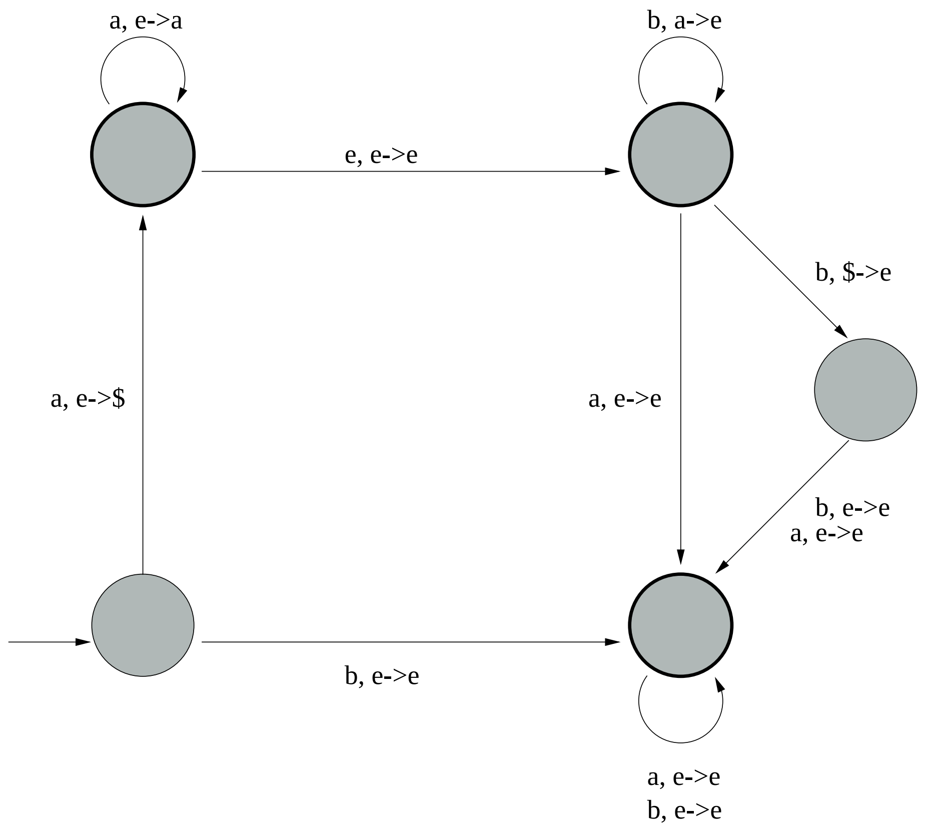


Figure 1: PDA for Problem 4

on the structure of w. We note that w is not of the form $\mathtt{a}^n\mathtt{b}^n$ if and only if one of the following is true: (1) w is of the form $\mathtt{a}^i\mathtt{b}^j$ for some $i > j \geq 0$, or (2) w is of the form $\mathtt{a}^i\mathtt{b}^j$ for some $j > i \geq 0$, or (3) w contains `ba` as a substring. In other words, A is the union of the three languages A_1, A_2, A_3, where

(1) $A_1 = \{\, \mathtt{a}^i\mathtt{b}^j \,:\, i > j \geq 0 \,\}$

(2) $A_2 = \{\, \mathtt{a}^i\mathtt{b}^j \,:\, j > i \geq 0 \,\}$

(3) $A_3 = \{\, w \,:\, \mathtt{ba}$ is a substring of $w \,\}$.

The grammar G above was designed so that we have:

$$\begin{aligned} \{\, w : T_1 \Rightarrow^* w \,\} &= A_1 \\ \{\, w : T_2 \Rightarrow^* w \,\} &= A_2 \\ \{\, w : T_3 \Rightarrow^* w \,\} &= A_3 \,. \end{aligned}$$

You should make sure you understand what the above statements mean and why they are true. Now, the first rule of the grammar tells us that $L(G) = A_1 \cup A_2 \cup A_3$.

We remark that the high-level idea of the construction follows the idea behind the proof of the closure of the class of CFLs under union given in lecture. Also notice that the last rule of this grammar is the same (up to a change in variable name) as the last rule of the grammar of Problem 1 above, and has the same functionality, namely to generate any string. This shows how we can reuse pieces of old grammars in building new ones.

Now we need to also provide a PDA that recognizes this language. Here is a tempting thought. Take the PDA for the language $\{ \mathtt{a}^n\mathtt{b}^n \ : \ n \geq 0 \}$ as given in Figure 2.6, page 104 of the text, and change final states to non-final states and vice versa. Meaning, the PDA we suggest is that of Figure 2.6 except that now states q_2, q_3 are final while states q_1, q_4 are not. This does *not* work. You should check that the PDA we have just described does not recognize A. Instead, we have to design a different PDA. Its state diagram appears in Figure 1.

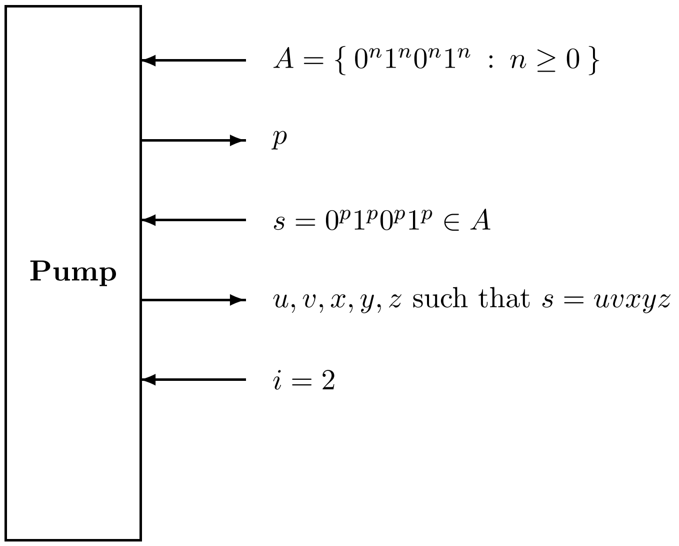
Problem 5 Problem 2.18 part (a), page 121 of the text.

We have seen numerous examples of proving non-regularity of a given language via the pumping lemma for regular languages, and will use the same template here. You should use the same template in your own solutions.

The proof is by contradiction.

Assume: $A = \{ 0^n1^n0^n1^n \ : \ n \geq 0 \}$ is a CFL.

The assumption means that the pumping lemma (Theorem 2.19, page 115 of the text) applies to A. We imagine ourselves "interacting" with the lemma as follows:


Pump
$A = \{ 0^n1^n0^n1^n \ : \ n \geq 0 \}$
p
$s = 0^p1^p0^p1^p \in A$
u, v, x, y, z such that $s = uvxyz$
$i = 2$

We give it A, and it returns a pumping length p. Now, we choose a string $s \in A$ of length greater than p, and return it to the lemma. The choice of string is important for the rest of the argument, and we set it to $s = 0^p1^p0^p1^p$. Because s is in A and has a length greater than p, the pumping lemma says that s can be split into $uvxyz$ which obey the three conditions of the pumping lemma. The lemma returns u, v, x, y, z to us. We then choose $i = 2$ and return it to the lemma. At this point, the lemma guarantees that

(1) $uv^2xy^2z \in A$

(2) $|vy| > 0$

(3) $|vxy| \le p$

The interaction emphasizes what we can choose and what we cannot choose. We are allowed to choose s and i, but, having chosen s, we have no control over u, v, x, y, z. But we know they satisfy the three conditions.

Now our goal is to show that conditions (2) and (3) imply that uv^2xy^2z cannot be in A, contradicting condition (1) above. This means our assumption that A was a CFL is false.

How can we argue that $uv^2xy^2z \notin A$? This argument is harder than the ones we have seen for regular languages because in this case we are not given information on the length of u. So we do not know where vxy sits in the string $s = uvxyz$. We have to consider various possibilities.

We know that $s = uvxyz = 0^p1^p0^p1^p$. Condition (3) says that the length of vxy is at most p. This means that vxy is a substring of one of the following: the first 0^p1^p segment; or 1^p0^p; or the last 0^p1^p segment. We take these three possibilities one by one, and in each case show that $uv^2xy^2z \notin A$:

1. The first case is that vxy is a substring of the first 0^p1^p segment of $s = uvxyz = 0^p1^p0^p1^p$. Condition (2) says that vy has non-zero length. This means that uv^2xy^2z has the form $a0^p1^p$ where a either has more than p 0s or more than p 1s. In either case $a \ne 0^p1^p$ and hence $uv^2xy^2z \notin A$.

2. The second case is that vxy is a substring of 1^p0^p. Condition (2) says that vy has non-zero length. This means that uv^2xy^2z has the form 0^pa1^p where a either has more than p 0s or more than p 1s. In either case, $a \ne 1^p0^p$ so $uv^2xy^2z \notin A$.

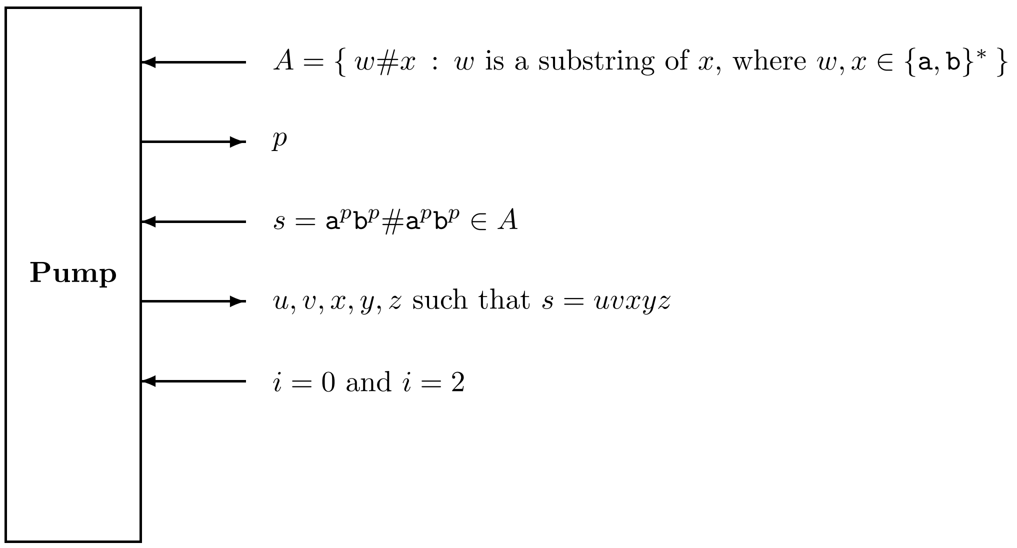
3. The last case is that vxy is a substring of the second 0^p1^p segment of $s = uvxyz = 0^p1^p0^p1^p$. Condition (2) says that vy has non-zero length. This means that uv^2xy^2z has the form 0^p1^pa where a either has more than p 0s or more than p 1s. In either case $a \ne 0^p1^p$ and hence $uv^2xy^2z \notin A$.

Problem 6 Problem 2.18 part (c), page 121 of the text.

We use the same template as above. The proof is by contradiction.

Assume: $A = \{\, w\#x \;:\; w \text{ is a substring of } x, \text{ where } w, x \in \{\mathtt{a}, \mathtt{b}\}^* \,\}$ is a CFL.

The assumption means that the pumping lemma (Theorem 2.19, page 115 of the text) applies to A. We imagine ourselves "interacting" with the lemma as follows:


Pump
$A = \{\ w\#x\ :\ w$ is a substring of x, where $w, x \in \{\mathtt{a}, \mathtt{b}\}^*\ \}$
p
$s = \mathtt{a}^p\mathtt{b}^p\#\mathtt{a}^p\mathtt{b}^p \in A$
u, v, x, y, z such that $s = uvxyz$
$i = 0$ and $i = 2$

We give it A, and it returns a pumping length p. Now, we choose a string $s \in A$ of length greater than p, and return it to the lemma. The choice of string is important for the rest of the argument, and we set it to $s = \mathtt{a}^p\mathtt{b}^p\#\mathtt{a}^p\mathtt{b}^p$. Because s is in A and has a length greater than p, the pumping lemma says that s can be split into $uvxyz$ which obey the three conditions of the pumping lemma. The lemma returns u, v, x, y, z to us. We are now allowed to choose i. In the past we have selected a single value of i. But notice that the lemma says the condition involving i is true for all values of i. This time, we will need to use this, and we select two different values of i, namely $i = 0$ and $i = 2$, and return them to the lemma. At this point, the lemma guarantees that

(1) $uxz \in A$ (because $uv^ixy^iz = uv^0xy^0z = uxz$) and also $uv^2xy^2z \in A$

(2) $|vy| > 0$

(3) $|vxy| \leq p$

The interaction emphasizes what we can choose and what we cannot choose. We are allowed to choose s and i, but, having chosen s, we have no control over u, v, x, y, z. But we know they satisfy the three conditions.

Now our goal is to show that conditions (2) and (3) contradict condition (1). This means our assumption that A was a CFL is false.

This argument is harder than the ones we have seen for regular languages because in this case we are not given information on the length of u. So we do not know where vxy sits in the string $s = uvxyz$. We have to consider various possibilities. For each possibility, we will show that *either* $uxz \notin A$ *or* $uv^2xy^2z \notin A$, which contradicts (1), since the latter says that both are in A.

We know that $s = uvxyz = \mathtt{a}^p\mathtt{b}^p\#\mathtt{a}^p\mathtt{b}^p$. Condition (3) says that the length of vxy is at most p. This means that one of the following is true: either vxy is a substring of the first $\mathtt{a}^p\mathtt{b}^p$ of s; or vxy contains $\#$; or vxy is a substring of the second $\mathtt{a}^p\mathtt{b}^p$ of s. We take these three possibilities one by one, and in each case show that either $uxz \notin A$ or $uv^2xy^2z \notin A$:

1. The first case is that vxy is a substring of the first $\mathtt{a}^p\mathtt{b}^p$ segment of $s = uvxyz = \mathtt{a}^p\mathtt{b}^p\#\mathtt{a}^p\mathtt{b}^p$.

Condition (2) says that vy has non-zero length. This means that uv^2xy^2z has the form $w\#\mathtt{a}^p\mathtt{b}^p$ where w either has more than p a's or more than p b's. In either case w is not a substring of $\mathtt{a}^p\mathtt{b}^p$ and hence $uv^2xy^2z \notin A$.

2. The second case is that vxy contains #. If v or y contain # then uv^2xy^2z has more than one #, meaning $uv^2xy^2z \notin A$. So, it must be the case that x contains #. Condition (2) implies that either v or y has non-zero length. If $v \neq \varepsilon$, then by Condition (3), v consists of one or more b's and y consists of zero or more a's. Therefore, uv^2xy^2z has the form $w\#\mathtt{a}^{p+j}\mathtt{b}^p$ where $j \geq 0$ and w has more than p b's. Thus w is not a substring of $\mathtt{a}^{p+j}\mathtt{b}^p$ and hence $uv^2xy^2z \notin A$. If $v = \varepsilon$, then by Conditions (2) and (3), y consists of one or more a's. Therefore, uxz has the form $\mathtt{a}^p\mathtt{b}^p\#w$ where w has less than p a's. Thus $\mathtt{a}^p\mathtt{b}^p$ is not a substring of w and hence $uxz \notin A$.

3. The last case is that vxy is a substring of the second $\mathtt{a}^p\mathtt{b}^p$ segment of $s = uvxyz = \mathtt{a}^p\mathtt{b}^p\#\mathtt{a}^p\mathtt{b}^p$. Condition (2) says that vy has non-zero length. This means that uxz has the form $\mathtt{a}^p\mathtt{b}^p\#w$ where w either has less than p a's or less than p b's. In either case $\mathtt{a}^p\mathtt{b}^p$ is not a substring of w and hence $uxz \notin A$.

Problem 7 Exercise 2.17 part (a), page 121 of the text.

The way to approach this question is to begin with the template we have used many times before of writing down what we are given and have to show, and then providing a construction.

Given: Language C is context-free, meaning there is a PDA $M_C = (Q_C, \Sigma, \Gamma, \delta_C, q_C, F_C)$ that recognizes C. Also language R is regular, meaning there is a DFA $M_R = (Q_R, \Sigma, \delta_R, q_R, F_R)$ that recognizes R.

Want: To construct a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ that recognizes $C \cap R$, showing that $C \cap R$ is a context-free language.

Construction: The idea of the construction is that M can run M_C and M_R "in parallel", using its stack to simulate the stack of M_C. Parallelism is implemented by making a state of M a pair consisting of a state of M_C and a state of M_R. Here now is the description of the PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$. As the notation indicates, the stack alphabet of M is the same as the stack alphabet of M_C, and the input alphabets of M, M_C, M_R are all the same. In addition:

- $Q = Q_C \times Q_R$ (that is, a state of M is a pair (c, r) where $c \in Q_C$ is state of M_C and $r \in M_R$ is a state of M_R)
- $q_0 = (q_C, q_R)$ (that is, the start state of M is the pair consisting of the start state of M_C and the start state of M_R)
- $F = F_C \times F_R$ (that is, a final state of M is a pair (c, r) where $c \in F_C$ is a final state of M_C and $r \in F_R$ is a final state of M_R).
- The transition function δ takes input a state $(c, r) \in Q$, an input symbol $\sigma \in \Sigma_\varepsilon$ and a

top-of-stack symbol $\gamma \in \Gamma_\varepsilon$, and returns the following:

$$\delta((c,r),\sigma,\gamma) = \begin{cases} \{\, ((c',\delta_R(r,\sigma)),\gamma') \;:\; (c',\gamma') \in \delta_C(c,\sigma,\gamma) \,\} & \text{if } \sigma \neq \varepsilon \\ \{\, ((c',r),\gamma') \;:\; (c',\gamma') \in \delta_C(c,\sigma,\gamma) \,\} & \text{if } \sigma = \varepsilon \end{cases}$$

What's going on here? If PDA M_C was in state c, scanning input symbol σ, and γ was on top of its stack, then its transition function δ_C tells us that it could go to state c' and push γ' on top of the stack for any (c',γ') in the set $\delta_C(c,\sigma,\gamma)$. Our machine M starts from a state of the form (c,r) where r is state of M_R. It wants to update c and its stack according to δ_C, while simultaneously updated r as per the transition function of M_R, meaning to the value $\delta_R(r,\sigma)$. That's what the above does, except that it is careful to make a special case when M_C has an ε-transition. In that case, c and the stack should be updated, but r stays the same.