

Solutions for Problem Set 3
cs172

Allison Coates

2.2a,b, 2.7, 2.18 a,b

2.2a Use the languages $A = \{a^m b^n c^n \mid m, n \geq 0\}$ and $B = \{a^n b^n c^m \mid m, n \geq 0\}$ together with Example 2.21 to show that the class of context-free languages is not closed under intersection.

First, we need to show that A and B are context free languages. One simple solution is to construct grammars for A and B . Here is the grammar for A :

$$\begin{aligned} S &\rightarrow TV \\ T &\rightarrow Ta \mid \epsilon \\ V &\rightarrow bVc \mid \epsilon U \end{aligned}$$

Likewise, here is the grammar for B :

$$\begin{aligned} S &\rightarrow VT \\ T &\rightarrow cT \mid \epsilon \\ V &\rightarrow aVb \mid \epsilon \end{aligned}$$

Given that A and B are context free, we then prove that CFLs are not closed under intersection by intersecting them, and getting something not context free.

Example 2.20 shows that $C = \{a^n b^n c^n\}$ is not a CFL. But the intersection of A and B as defined above is C . Hence, CFLs are not closed under intersection.

2.2b Using this result and de Morgan's laws we show that CFLs are not closed under intersection.

We prove by contradiction that CFLs are not closed under complement. First, we assume that CFLs are closed under complement. CFLs are closed under union (i.e. if A and B are CFLs, then $A \cup B$ is a CFL). Assuming CFLs are closed under complement, then $\overline{A \cup B}$ is a CFL. But by de Morgan's laws, $\overline{A \cup B} = \overline{A} \cap \overline{B}$. Since these languages are closed under complement, then $\overline{A} \cap \overline{B}$ is a CFL. But CFLs are not closed under intersection, shown above; there exist languages A and B such that $\overline{A} \cap \overline{B}$ is not a CFL. Hence, contradiction. Hence, CFLs are not closed under complement.

2.7 Given an informal of a pushdown automaton that recognizes the language:

- a. The set of strings over the alphabet $\{a, b\}$ with twice as many a 's as b 's.

Build a PDA that contains two states in its finite control: a "0" state, which is an accept state, and another state, the "need 1 a" state. You can think of the "0" state as meaning that we need to see two a 's, or that we have seen 0 a 's. These two states keep track of whether we have seen an odd or even number of a s so far. Informally, we define the transitions rules as follows: we will need to push either an a or a b onto the stack. A b on the stack means we have seen one b , and need two a s; an a on the stack means we have already seen an a , and now we need one b . We will keep track of whether we have seen We can't simply solve this problem by saying "push all the a 's onto the stack" or "push all of the b 's onto the stack" because we may see all b 's followed by all a 's, and vice versa. so instead, we will need to push either a 's or b 's—depending on the incoming symbol and the machine's current state. We will never see any mixtures of a 's and b 's on the stack. at any given time, there will only be a 's or b 's. As they are popped off until there are no more letter on the stack, then the stack can change the symbol it holds, so that over time, it could contains either a 's or b 's.

More specifically, the rules for the PDA are:

Before reading the input, push the symbol \$ onto the stack. We will accept the input if after reading it, the stack contains only the \$ symbol and the finite control is in an accept state. Read the first letter.

If input contains a letter other than a/b , reject.

If it is an a , enter the “need 1 a ” state. Do not push this symbol onto the stack.

If the character is a b , enter a “0” state and push the b onto the stack.

Repeat for length of input: For all inputs after the first input, use the following rules:

If the symbol is an a , and we are in the “need 1 a ” state, check the stack. If the top of the stack contains a b , pop the b , and move the finite control to the “0” accept state. If we are in the “need 1 a ” state and the top of the stack does not contain the symbol b , push the a onto the stack and return to the 0 state.

If we are in the “0” state, update the finite control to enter the “need 1 a ” state, and do nothing to the stack.

If the next symbol is a b , and the stack contains an a , pop the a off the stack. Do not change the finite control. If there is no a on the stack, we are in the “need 1 a ” state,

- b. The complement of the language $\{a^n b^m | n \geq 0\}$.

As seen in the last homework, the complement of this language consists of three types of strings:

all strings containing the substring ba

the strings $a^n b^m, n < m$

the strings $a^n b^m n > m$.

We will describe a nondeterministic PDA. Assume the PDA will correctly guess which type of string we have; we need to show that we can write rules for a PDA to solve each of these problems. The PDA for the first subset is simple, since a finite automata can solve it. We create a machine whose finite control accepts if and only if the string contains ba . The PDA to recognize the languages in the second and third subset can be described as follows: If the input contains symbols other than a 's and b 's, or the string has a 's following the b 's, reject. Push the symbol \$ onto the stack. Then, begin reading the a s from the input. Push all a 's onto the stack. As soon as the PDA reads the first b in the input, pop an a off of the stack for every b that is read. If the stack is empty, but there are still more b 's, accept. If the stack contains a 's but the input is finished, accept. Reject if the input is finished and the stack's top symbol is \$.

- c. $\{w\#x|w^R \text{ is a substring of } x \text{ for } w, x \in \{0, 1\}^*\}$.

This machine relies upon nondeterminism. Let M be a nondeterministic PDA. The rules for M are: Push the symbol \$ onto the stack. Push all symbols onto the stack until a # is read. From that point forward, use nondeterminism to correctly guess where the substring w^R begins inside the string x . Pop off matching symbols between w and x ; when the stack is empty, continue reading the rest of the input. If the stack contains only the \$ symbol when the input is finished, accept. If the stack contains other symbols, reject.

- d. $\{x_1\#x_2\#\dots\#x_k | k \geq 1, \text{ each } x_i \in \{a, b\}^* \text{ and for some } i \text{ and } j, x_i = x_j^R\}$. This is similar to the previous problem. Assume M is a nondeterministic automaton. M can correctly guess the proper point at which to match some x_i to some x_j^R . Assume M properly guesses which x_i and x_j are palindromic. Then M can push x_i onto the stack until the input is a #. Then, it can ignore symbols until it reaches the correct x_j . Then, it can pop symbols off the stack if they match x_j . If the stack is empty at the end of the input, accept, else reject.

2.18a Use the pumping lemma to show that the following language is not context free:

$$L = \{0^n 1^n 0^n 1^n | n \geq 0\}$$

The pumping lemma for CFLs is:

Lemma 1 *If A is a context free language, then there is a number p such that if s is any string in A of length at least p then s may be divided into five pieces $s = uvxyz$ satisfying the conditions:*

1. For each $i \geq 0$, $uv^i xy^i z \in A$,
2. $|vy| > 0$,
3. $|vxy| \leq p$.

Assume L as defined above is a context free language, with a corresponding pumping length p . Pick a string s , where $|s| = 0^p 1^p 0^p 1^p$. Hence, $s \in L$.

We now break s into pieces u, v, x, y, z .

Since $|vxy| \leq p$, then vxy is one of the following types of strings: a string of all 0s; a string of all 1s; a string of 0s followed by 1s, or a string of 1s followed by 0s. That is, this string cannot be of the form $000\dots 111\dots 0\dots 1\dots$

Assume vxy is a string of all 0s. Since $|vy| > 0$, then either v or y must contain at least one 0. Pumping v and y means that $s = uv^i xy^i z$ must contain more 0s than 1s. But this string is not in the language L . Likewise, assume that vxy is a string of all 1s. Then either v or y must contain at least one 1, and pumping that string will now contain more 1s than 0s. This string is not in the language L .

Now assume that vxy is a string of 0s followed by 1s. Either one of v and y is a string of the form $0\dots 1\dots$, or v and y are substrings where v is a string of 0s and y is a string of 1s. If the former is true, then pumping v and y will produce strings of the wrong form, and hence not in L . If the latter is true, but v and y are of differing lengths, then pumping them will produce a different number of 1s and 0s. Hence such strings are not in L . It is possible to produce string such that $v^i xy^i$ maintains the $0^n 1^n$ for the first half of string s , but there is no way to pump the rest of the string. Such pumped strings are of the form $0^r 1^r 0^n 1^n$, $r \neq n$. These strings are not in L .

Likewise, the same argument holds when vxy is a string of 1's followed by 0's. We then can, in the best case, produce string of the form $0^n 1^r 0^r 1^n$, $r \neq n$ which are not in L . Hence, there is no way to break up s so that s can be pumped and remain in L . Hence, L is not a CFL.

2.18b Prove using the Pumping Lemma that the following language is not a CFL:

$$L = \{0^n \# 0^{2n} \# 0^{3n} \mid n \geq 0\}$$

Proof by contradiction. Assume L is a CFL. Pick a string $s = 0^p \# 0^{2p} 0^{3p}$. $s \in L$. Hence, s can be pumped.

We can break s into the pieces $uvxyz$ according to the conditions of the Pumping Lemma. Since $|vxy| \leq p$, then vxy is one of the following types of strings: a string of all 0s, or a string containing exactly 1 $\#$ with 0s on either side. This string cannot be of the form $000\dots \# \dots 0\dots \# \dots$

Assume vxy is a string of all 0s. Pumping v and y will produce a string that contains too many 0s in one of the three subsets; the counts will no longer match the form $n, 2n, 3n$. Hence, these strings are not in L . Assume that vxy contains a $\#$. For now, assume vxy straddles the length n portion and the length $2n$ portion. If either v or y contains the $\#$, then pumping that string will produce a string with too many $\#$ s, which is not in L . If x contains the $\#$ symbol, then at best, we can choose v and y such that $|y| = 2|v|$. Then, pumping v and y can produce strings such that the first substring is correct relative to the second substring, but there is no way to pump the third substring. We now have strings of the form $0^{n+i} \# 0^{2(n+i)} \# 0^{3n}$, which are not in L . Likewise, we could have had vxy straddle the second $\#$, but we have no way to straddle both. Hence, s cannot be pumped and remain in L . so L is not context free.