

Theory of Computation
CS 378
Fall 2022

Problem Set 3

Due **AT THE BEGINNING OF CLASS** Thursday, November 3

- Standard file naming and HW submission conventions apply: Please submit your type-written answers in a PDF file named

CS378_PS3_<userid>.pdf

where <userid> is replaced by your Colby userid. Please use your full userid, including your class year. Place the properly named file in your **Private** directory in your CS378 filespace on **filer**.

- Please finish reading Sipser Ch. 2.1 and 2.2.
- **IMPORTANT:** As an exercise for yourself, please do Exercise 2.6(a) on page 155 of Sipser, the solution for which is given in the textbook. Make sure you understand the solution, and (as always) please feel free to ask your Prof. any questions you may have about it!
- As always, answers to all exercises should be accompanied by explanations; brief explanations sometimes suffice, but all answers should be explained. Answers without explanations may not be awarded full credit.
- *A general note:* When writing up your homework, please explain your arguments clearly and write neatly. Graders may not award credit to incomplete or illegible solutions. Clear communication *is* the point, on every assignment.
- *A general note:* Neatly written (or typeset) solutions, with enough blank space on the page to allow graders to write comments before returning the papers, are greatly appreciated!

1. Exercise 2.6(d) on page 155 of Sipser. Note that the alphabet is $\{a, b, \#\}$.

As part of this exercise (and for every CFG designed for CS378!), be sure to explain briefly what each variable in the grammar represents. (For example, “The variable S generates all strings in the language in the problem. The variable A generates all strings with an equal number of 0’s and 1’s.”) This should greatly simplify your explanation of how your answer is a correct grammar for the given language!

Although it is certainly possible to over-explain CFG design, it may be even more possible to under-explain it—to assume a reader will understand points of design that aren’t obvious to them. Please be thoughtful and thorough in explaining what each variable means; giving example strings generated by each variable can be a helpful part of that.

A Small Hint: We’ve gone over a CFG that generates palindromes already in a CS378 class meeting. What is the role of palindromes in this exercise? How might knowing how to generate a palindrome—or a palindrome with some modifications, perhaps—be useful here?

2. Context-Free Grammars and PDA's!

- (a) Give a context-free grammar for the language

$$L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and either } k = i \text{ or } k = i + j\}.$$

Explain briefly what each variable in your grammar represents. (For example, “the variable B in my grammar represents the situation that I have seen an equal number of 0’s and 1’s and now I expect to see only 1’s.”)

- (b) Draw the state diagram for a PDA for the above language.

Note: You may not use the mechanical construction from Theorem 2.20 and Lemma 2.21 for this exercise! Instead, consider Figures 2.15 and 2.17 in your Sipser textbook as examples. Please design the PDA and, as with designs of DFAs (indeed, as with any other machine models in CS378), be sure to explain the state set, transition function, etc.; answers to this exercise will be evaluated as all machine designs are in CS378, so please make sure the design isn’t extremely inelegant, please include a correctness argument (it can be brief if the design is explained well!), and follow other course guidelines for designs of machine models.

As always, please feel free to talk with your Prof. if there are any questions about this!

3. **A Funky Language!** Sipser, Exercise 2.22 (page 156). Note that $\#$ is one of the symbols in the alphabet here. In general, to show that the language is context-free you may either give a context-free grammar or a PDA. *In this case, describe in English how a PDA would work for this language.* (You may also include PDA state diagrams or grammars in your answer, if they improve the clarity of your English description, but the exercise here is to write a clear English description.) Your description might begin something like this: “First I would push blah, blah, blah onto the stack. Then, for each symbol of type blah that I see, I push blah onto the stack. Then, I blah, blah, blah.” Avoid using the term “blah” in your own description if at all possible!

Note: There may be **lots** of non-determinism in a PDA for this language! It may therefore be difficult to give a fully formal description of the PDA—that’s why this exercise asks for an English description. Formal notation may be part of an answer for this, and a description of every state and δ transition in the PDA should be somewhere in a fully complete answer, but a completely formal description (of the sort given for simple DFA’s, for instance) is not required. Please be sure to give a sufficiently detailed and clear English description, however, for full credit on this exercise!

Hint: In general, one way to show that two strings $x = x_0x_1 \dots x_n$ and $y = y_0y_1 \dots y_m$ are unequal (all $x_i, y_i \in \Sigma$) when $m = n$ is to show that there’s some position j for which $x_j \neq y_j$. (Do you see why?) Recall that there is no length restriction on x , y —they could be arbitrarily long!—so there’s no restriction on what position j might be that would be useful in showing $x \neq y$.

Recall that states of a DFA are essentially the memory of that machine. Similarly, states of a PDA can also be used to store things in (finite) memory, and the stack also serves as a kind of memory. What could the stack and the states of a PDA be used for to store information needed to determine if x and y are unequal?

4. **A Slightly Funkier Language!** Sipser, Exercise 2.23 (page 157). As with exercise 3 above, please give an English description of how your PDA works rather than just a grammar or a state diagram. Be sure to explain why your PDA accepts exactly this language.

Note: This exercise is an interesting companion to exercise 3—it’s similar in many ways, but with some interesting differences!

Optional: And Now For Something Truly Funky!

This exercise is not to be turned in! It’s an exercise that your Prof. thought might be of interest, and you are welcome to work on it and talk with your Prof. about it, but it will not be graded. (The result of this exercise, however, may well come up again later in the course!)

And Now For Something Truly Funky! Consider the language

$$(a + b)^* - \{(a^i b^i)^i \mid i \geq 1\}.$$

This is the complement of the language $\{(a^i b^i)^i \mid i \geq 1\}$. Show that this language is context-free by carefully describing in English how a PDA that accepts this language would work. Be sure to explain why your PDA accepts exactly this language.

Problem Set 4 Lookahead: The Shuffle!

Note: the exercise below is due for problem set 4, not problem set 3! It is here only to allow you to look ahead, if you’d like.

The Shuffle! Two words w_1 and w_2 can be “shuffled” into a single new word by taking zero or more consecutive letters from the beginning of w_1 followed by zero or more letters from the beginning of w_2 followed by the next 0 or more letters from w_1 , followed by the next 0 or more letters from w_2 , etc. until all characters are expended. For example if w_1 is the word “funny” and w_2 is the word “joke” then one possible shuffle of w_1 and w_2 is the word “fujnnokye”. The first two letters, “fu”, come from the beginning of w_1 . The next letter, “j” comes from the beginning of w_2 . The next two letters, “nn”, come from the next two letters of w_1 . Then comes “ok” from w_2 , “y” from w_1 , and finally “e” from w_2 . Another possible shuffle of these two words is “funnjokye”.

Given two languages, L_1 and L_2 , $\text{shuffle}(L_1, L_2)$ is a new language formed by shuffling each word in L_1 with each word in L_2 in all possible ways.

1. Show that the regular languages are closed under shuffle. That is, show that if L_1 and L_2 are regular, then $\text{shuffle}(L_1, L_2)$ must also be regular.
2. Show that the context free languages are **not** closed under shuffle. Again, prove this carefully. (*Note:* To do this you need only demonstrate a single counterexample. If you can find two languages L_1 and L_2 which are CFL’s but whose shuffle is not a CFL then you are done. If that doesn’t work, you might consider shuffling two CFL’s and then applying a known closure property of CFL’s.)
3. Show that the shuffle of a CFL and regular language is always a CFL. Explain this carefully.