CS 3100, Fall 2019,
Basic Battery of Problems for Midterm-2

1 Some Student FAQs

1. Is dealing with strings of the form $0^M 1^N$ for $M > N$ easier in a PDA compared to a language where the number of 0's are required to be greater than the number of 1's? Why?

2. What are final states? What are accept states? Are they the same? (Answer: yes; we tend to use "accept" states a bit more with TM, because that is a final state in which the TM halts, and we then know that the TM has accepted the input—i.e. not rejected the input.)

3. How do you include an "epsilon move" in a PDA? This is a move akin to an NFA's epsilon move.

4. How do you feed the shortest string in these languages to a TM:

   (a) $\{0^i 0^i \# 0^i \ : \ i \geq 0\}$
   (b) $\{0^i \# 0^i 0^i \ : \ i > 0\}$
   (c) $\{w \# ww \ : \ w \in \{0,1\}^*\}$

5. A CFL is a language for which there is a PDA that serves as an exact acceptor. That is, L is a CFL, if and only if there is a PDA P such that

   - $w \in L \Rightarrow\ P$ when run on w will end up in a final state **at which point the input is fully consumed.** *Note that unlike with a TM,* a PDA can continue on and transition to other states.

But since the input is fully consumed, those transitions do not read additional input.

- $w \notin L \Rightarrow$ M when run on w will **not** reach one of the final states **with the input fully consumed.**

6. Can a PDA "loop" (i.e., infinitely loop) like a TM? Yes it can: one can simply sit in a tight loop and push things on the stack. However, such excursions are useless. While this is somewhat difficult to prove in the PDA-land, it is much easier to prove in the CFG-land. Here is how: (1) convert the PDA to a CFG (this conversion is not that hard, and found in many books out there). (2) See if the CFG can parse w using one of the standard parsing methods. This construction shows that we don't need to be "stuck" in the PDA infinite loops before deciding whether w is accepted or not.

7. Answer these questions either thinking of a PDA or a CFG. Ideally, you must provide an answer through both means.
   (a) Is the union of two CFLs a CFL?
   (b) Is the concatenation of two CFLs a CFL?
   (c) Is the Kleene-star of a CFL a CFL?
   (d) Is the intersection of two CFLs a CFL? Hint: think of the intersection of
      i. $\{a^m b^n c^n \;:\; m, n \geq 0\}$
      ii. $\{a^n b^n c^m \;:\; m, n \geq 0\}$
   (e) Based on your answer to the above questions, can the complement of a CFL be a CFL?

8. A recursively enumerable language is one for which there is a TM that serves as an exact acceptor. That is, L is RE (recursively enumerable)[1] if and only if there is a TM M such that
   - $w \in L \Rightarrow$ M when run on w will halt in one of the accept states of M
   - $w \notin L \Rightarrow$ M when run on w will **not** halt in one of the accept states of M. (This may mean that M may halt in a non-accept state (this is compactly stated as "M rejects") or M may loop (we can't tell this; so long ans M does not accept, it is the other two possibilities, namely rejection or looping, that must be true)

9. Answer these questions either thinking of a PDA or a CFG. Ideally, you must provide an answer through both means.
   (a) Is the union of two RE languages RE?
   (b) Is the concatenation of two RE languages RE?
   (c) Is the Kleene-star of an RE language RE?
   (d) Is the complement of an RE language RE?
   (e) Is the intersection of two RE languages RE?

10. What are the two rules that help translate **any** CFG into a PDA?

11. This CFG arose naturally in one of my adventures:

[1]Not "L is a regular expression"

```
S -> ( W S | ''
W -> ( W W | )
```

- What familiar language is this CFG describing?
- How do you show it is consistent? (Exam questions won't be this hard, but this is given for your own curiosity.)
- How do you show it is complete? (Exam questions won't be this hard, but this is given for your own curiosity.)

12. What are two classes of strings in the **complement** of the language ww (set of strings of the form w followed by w; examples being 01001 01001)? How do you write a CFG for the complement of the ww language?

13. Suppose there is a CFG production for nonterminal A. How do you write a CFG production for the language A^*?

14. Suppose there is a CFG production for nonterminals A and B. How do you write a CFG production for the language $(A+B)$?

15. How do you go about designing a CFG for the set of strings where the #1's exceeds the #0's (Assignment 4 question)?

    Hint: don't try to do it in one rule! It won't work. Try to conquer the "equality" and "ascending" parts separately. Then stitch together using the tricks mentioned in the earlier questions. You'll end up with many CFG rules (4 or 5 in my estimate).

16. Design a PDA for the set of strings with twice as many b's as a's.

Hint: You have to maintain a discipline wrt the things on the stack. I'll also show you how to interactively arrive at the PDA design, executing piece by piece.

2 Battery of Questions

1. (Simple CFG patterns) Write a CFG for 0^*
2. (Simple CFG patterns) Write a CFG for 10^*1
3. (Simple CFG patterns) Write a CFG for the set of odd-length palindromes over $\{0, 1\}$
4. (Simple CFG patterns) Write a CFG for the set of odd-length strings over $\{0, 1\}$
5. (Simple CFG patterns) Write a CFG for the language $0(00)^*$
6. (Simple CFG patterns) How many productions will there be in a CFG for the language over $\{a, b\}$ with two a's for every three b's?
7. (Designing/understanding CFG) Given this CFG with `S` as the start symbol,

```
S -> XSX | R
R -> aTb | bTa
T -> XTX | X | ''
X -> a|b
```

answer the following questions:

   (a) What are the terminals and nonterminals?

(b) How many productions are there (the | is an abbreviation; de-abbreviate those and count)
(c) Draw a parse tree for `abaa`
(d) Draw a parse tree for `abaaa`
(e) Write a regular expression for the language of this grammar (in general, this is impossible, but here you can). **CAN YOU??** I don't have a proof yet for this!! You are invited to try!!

8. There was another "trickster" that I ran into for which I think I have a solution—will try later. Here is that one, which is **Sipser's Problem 2.25**. Suppose we are given a CFG G

```
S -> a S b | b Y | Y a
Y -> b Y | a Y | ''
```

   (a) If $L(G)$ is regular, simplify it down to a regular expression.
   (b) If $L(G)$ is not regular, obtain a Pumping Lemma proof for that fact.

   CAN YOU?? I think I got a simplification for it. You are invited to try!!

9. (Designing/understanding CFG) Design a CFG for the set of strings w that contain at least three 1's ($\Sigma = \{0, 1\}$)

10. (Designing/understanding CFG) Design a CFG for the set of strings w that are odd in length and the middle symbol is a 0 ($\Sigma = \{0, 1\}$). Draw a parse tree for 01011 using this grammar.

11. (Designing/understanding CFG) Design a CFG for the set of balanced parentheses, square brackets, and braces. For example, here are the legal strings in this language:

    (a) `()[]{}`
    (b) `{{[()()][{}]}}`
    (c) `[ {{[()()][{}]}} ]`
    (d) `[ {{[()()][{}]}} {[()()][{}]} ] ()[]{}`

12. Argue the consistency and completeness of the grammar you arrive at in Question 11. To simplify the case analysis during completeness, ignore `{` and `}` (i.e. assume the strings only consist of `(`, `)`, `[`, or `]` – of course, well-parenthesized).

13. (Designing/understanding CFG) Write a CFG for the set of strings over $\{a, b, c\}$ where the number of as equals the number of bs plus the number of cs.

14. (Designing/understanding CFG) Design a CFG for the language of strings $\{a^i b^j c^k \ : \ i = j \text{ or } j = k\}$

15. (Designing/understanding CFG) Design a CFG for the language of strings $\{a^i b^j c^k \ : \ i = j \text{ or } i = k\}$

16. (Designing/understanding CFG) Design a CFG for the language of strings of the form $(waw^R)(wbw^R)$ where $w \in \{0, 1\}^*$ and $a, b \in \{0, 1\}$, with $a \neq b$.

17. (Designing/understanding CFG) Consider the expression grammar

```
E -> E + E | E * E | E ** E | 2 | (E)
```

Here, `**` stands for exponentiation.

(a) Define grammar ambiguity.
(b) Is this grammar ambiguous? Why? (Present through a clear example.)
(c) Disambiguate this grammar. While disambiguating, assign exponentiation the highest precedence, followed by `*` (multiplication), and finally addition (`+`).
(d) Provide an informal proof that the grammar has been disambiguated.

18. (Designing/understanding CFG) Consider the regular expression grammar for regular expressions over $\Sigma = \{a\}$, with `@` used to represent Epsilon (for convenience).

```
E -> E E | E* | E+E | @ | a
```

(a) Define grammar ambiguity.
(b) Is this grammar ambiguous? Why? (Present through a clear example.)
(c) Disambiguate this grammar. While disambiguating, we want regular expression concatenation to have higher precedence than regular expression union (`+`), and Kleene star (`E*`) to have higher precedence than concatenation.
(d) Provide an informal proof that the grammar has been disambiguated.

19. Argue that the syntax of regular expressions is context-sensitive while the syntax of CFG productions is regular.

20. (Conversion of CFG to PDA) **Convert each and every CFG you designed earlier into a PDA.** Argue why this PDA works.

21. (Identifying languages to be CFL or not; PL proof if not) Is the language 0^n1^n for $n \geq 0$ a CFL? If so, present a CFG and a PDA for it. If not, provide a PL proof for it. (**HINT:** See if you can design a PDA for the language; if it appears impossible, it is likely a CFL.)

22. (Identifying languages to be CFL or not; PL proof if not) Is the language $0^n1^n0^m1^m$ for $m, n \geq 0$ a CFL? If so, present a CFG and a PDA for it. If not, provide a PL proof for it.

23. (Identifying languages to be CFL or not; PL proof if not) Is the language $0^n1^n0^m1^m$ for $m, n \geq 0$ and $m = n$ a CFL? If so, present a CFG and a PDA for it. If not, provide a PL proof for it.

24. (Identifying languages to be CFL or not; PL proof if not) Is the language $0^n1^n0^m1^m$ for $m, n \geq 0$ and $m \geq n$ a CFL? If so, present a CFG and a PDA for it. If not, provide a PL proof for it.

25. (Identifying languages to be CFL or not; PL proof if not) Is the language $0^n1^n0^m1^m$ for $m, n \geq 0$ and $m > n$ a CFL? If so, present a CFG and a PDA for it. If not, provide a PL proof for it.

26. (From Sipser)(Identifying languages to be CFL or not; PL proof if not) This is the language: $\{a^ib^jc^k \,:\, i \leq j \leq k,\ i, j, k \geq 0\}$

27. (From Sipser)(Identifying languages to be CFL or not; PL proof if not) This is the

language, where # is a separator: $\{0^n\#0^{2n}\#0^{3n} : n \geq 0\}$

28. (From Sipser)(Identifying languages to be CFL or not; PL proof if not) This is the language, where # is a separator: $\{w\#x : w \text{ is a subtring of x}\}$ where $w, x \in \{a, b\}^*$. w is a substring of x if w is a contiguous sequence of characters present in x.

29. (Designing/understanding CFG) What is the language generated by this CFG? Design a PDA for it both using direct design (without converting a CFG to a PDA) and through conversion. Here, ! is simply a separator.

```
T -> 0T | T0 | !
```

30. (Designing/understanding CFG) Consider this CFG once again:

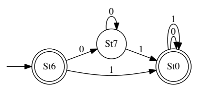
```
T -> 0T | 1T | !
```

    (a) Is this CFG purely left-linear?
    (b) Is it purely right-linear?
    (c) Is its language regular?
    (d) What is the theorem pertaining to mixed linear CFG productions and whether the language is regular or not? State it clearly as an implication or a bi-implication. Does this theorem help you judge whether this language is regular or not?

31. (Designing/understanding CFG) Reverse this CFG.

```
T -> 0T | 1T | !
```

Call the new nonterminal `Tr` for "T reversed." Then answer Question 30.

32. (Linearity; DFA to CFG conversion) Convert this DFA to a CFG by the direct approach of converting transitions to CFG 

productions.

33. (Linearity; DFA to CFG conversion) Consider this CFG once again:

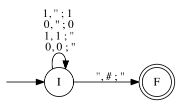
```
T -> 0T | T0 | !
```

    (a) Is this CFG purely left-linear?
    (b) Is it purely right-linear?
    (c) Is its language regular?
    (d) What is the theorem pertaining to mixed linear CFG productions and whether the language is regular or not? State it clearly as an implication or a bi-implication.

34. (Designing/understanding CFG) What is the language generated by this CFG? Design a PDA for it both using direct design (without converting a CFG to a PDA) and through conversion. Here, ! is simply a separator.

```
S -> 0S00 | !
```

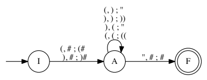
35. (PDA basics) In a few sentences describe the language of the PDA presented below (Jove commands to generate the PDA are also shown, for added information).



Why is this PDA's language not that of the set of palindromes over $\{0, 1\}$?

```
pda1 = md2mc('''PDA
I : 0,''; 0 | 1,''; 1 | 0,0; '' | 1,1; '' -> I
I : '', #; '' -> F
''')
DO_pda1 = dotObj_pda(pda1, FuseEdges="True")
DO_pda1.render()
DO_pda1
```

36. (PDA basics) In a few sentences describe the language of the PDA presented below.



Why is this PDA's language not that of the Dyck language (which is the set of perfectly nested parentheses, such as `()()` or `(())` or `((())())`. What changes will you make to this PDA's design to convert it into a PDA for the Dyck language?

```
pda2 = md2mc('''PDA
I : (,#; (# | ),#; )# -> A
A : (,(; (( | ),); )) | (,); '' | ),(; '' -> A
A : '',#; # -> F
''')
DO_pda2 = dotObj_pda(pda2, FuseEdges="True")
DO_pda2.render('DO_pda2')
DO_pda2
```

37. (Designing/understanding simple PDA) Directly design a PDA (without converting from a CFG) for the language of Question 11.

38. (Designing/understanding simple PDA) Suppose $x, w \in \{a, b\}^*$ and x is a substring of w^R. An exmaple would be $x = ab$ and $w = bbab$ whereupon $w^R = babb$ and you see that x is a substring of w^R. In any case, x is a substring of w-reversed. Directly design a PDA for the language
$\{xw^R\}$ under these conditions, where `!` is a separator.

39. (Designing/understanding simple PDA) Directly design a PDA for the language of Question 14.

40. (Designing/understanding simple PDA) Directly design a PDA for the language of Question 15.

41. (Designing/understanding simple PDA) Directly design a PDA for the language of strings over $\{0, 1\}$ where the number of 1's is strictly more than the number of 0's.

42. (Designing/understanding simple PDA) Directly design a PDA for the language of strings w over $\{a, b\}$ such that w has twice as many b's as there are a's.

43. Design a CFG and PDA for these languages. If they are not CFL, then write a PL proof:
    (a) $L_1 = \{a^i b^j c^k d^l \;\; : \;$ if $i = 2$ then $j = k$else $k = l$
    (b) $reverse(L_1)$

44. (From Chapter 11) Argue that this language is a CFL by building a CFG for it. Answer for both cases of ‘OP’ listed below:

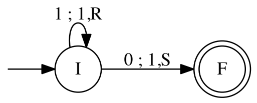
$$L_{abcd} = \{a^i b^j c^k d^l \ : \ i, j, k, l \geq 0 \text{ and } ((i = j) \text{ OP } (k = l))\}$$

(a) (Case 1) OP is AND
(b) (Case 2) OP is OR

45. (From Chapter 11) Show that L_{acbd} is not context-free if OP is AND but is context-free if OP is OR:

$$L_{acbd} = \{a^i c^k b^j d^l \ : \ i, j, k, l \geq 0 \text{ and } ((i = j) \ OP \ (k = l))\}$$

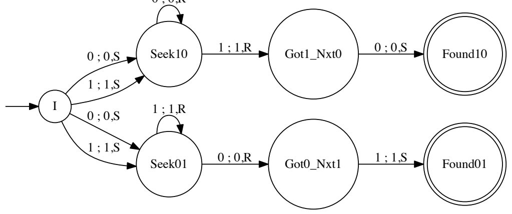
46. (Simple TM question) Read the description of this simple TM



Now answer these questions:

(a) What does this TM do upon input `101`?
(b) What does this TM do upon input `111`?
(c) What is the language of this TM?

47. (Simple TM question) Read the description of this simple TM



Now answer these questions:

(a) What does this TM do upon input `1111`?
(b) What does this TM do upon input `0010`?
(c) What is the language of this TM?