CS 3100 – Models of Computation – Fall 2011 – Practice Midterm-2

- These questions will help toward MT2 (which will be shorter than this practice list).
- In all questions, @ or ε mean "epsilon," the empty string.
- NFA states beginning with `I`: initial states; beginning with `F`: accepting states; beginning with `IF`: both.
- Anything such as `0, 1, 2, #, a, b` will be considered to be terminals (members of Σ)
- You are responsible for reading through the notes, especially the notions of acceptance of various machine types, as covered in notes22.pdf
- While all regular languages are context-free, when we say "context-free" in a general setting, it always means "context-free and not regular." This is similar to calling 0 a complex number; makes no sense, even though $0 = 0.0 + i0.0$, a perfectly fine complex number.

1. Is this a purely left-linear or purely right-linear grammar?

```
S -> a T
T -> S b
S -> @
```

2. How about this one? What class (regular or context-free)

```
T -> c T u | @
```

3. Is this linear? What NFA does it denote?

```
T -> c T | d U | @
U -> a T
```

Answer: Yes, it denotes the NFA

```
IFT - c -> IFT
IFT - d -> U
U - a -> IFT
```

4. Convert this NFA to a left-linear grammar or a right-linear grammar:

```
I - 0 -> F1
F1 - 1 -> F1
F1 - 0 -> I
```

Answer: The CFG is

```
I -> 0 F1
F1 -> @
F1 -> 1 F1
F1 -> 0 I
```

5. Know how to reverse a CFG: Reverse

```
S -> a T
T -> S b
S -> @
S -> U | V
U -> S a T b U
V -> S
```

Answer: Reversed CFG is

```
S_r -> T_r a
T_r -> b S_r
S_r -> @
S_r -> U_r | V_r
U_r -> U_r b T_r a S_r
V_r -> S_r
```

6. Simplifying CFGs

```
S -> a T
T -> S b
S -> @
S -> U | V
U -> S a T b U
V -> S
```

The simplified CFG is:

```
S -> a T
T -> S b  -- 3: T is generating
S -> @    -- 1: generating
S -> U | V
U -> S a T b U -- 4: U is not generating
               -- U is not reachable
V -> S   -- 2: V is generating


S -> a T - keep
T -> S b - keep
S -> @   - keep
S -> V   - keep
U -> S a T b U - remove
V -> S - keep

Finally,
S -> a S b
S -> @
```

7. Get the Pumping Lemma facts straight

   Incorrect Proof:

   In the PL proof for $\{a^i b^j c^k \mid if\ even(i)\ then\ j = k\}$, given an m-state DFA, someone picks the string $a^3 b^m c^{m+1}$ and pumps. They pick $y =$ "a" and show that by pumping, we can make "a" part even, and then b and c don't match. What is wrong?

   Answer: Have to consider *all possible splits of the first m positions into x, y, z.* In our PL version (same as the one used in the JFLAP tutor), the xyz string begins at the beginning of the string—not in the "middle" like another PL version in my book allows.

   Corrected Proof: Even if you pick $a^{2m+1} b^m c^{m+1}$, you don't win, because you don't know where the pump lies within a and how long it is.

8. Easier Proof:

   Reverse the language to obtain

   $\{c^k b^j a^i \mid if\ even(i)\ then\ j = k\}$

   then pick $c^m b^m a^2$ which is in the language; then pump "c". No case analysis needed - only one case.. the string goes outside in one pump, regardless of what "y" (consisting only of "c"s) is.

   Let's say we pump up and obtain $c^n b^m a^2$ where $n > m$. Goes outside. So can't be regular.

9. Be able to design simple PDAs and CFGs

   Design a PDA for the language

   $$\{a^i b^j \mid i = j\ or\ i = 2j\}$$

   Push the a's. When the b's come, for each b pop 2 a's.

10. Write a CFG for $\{a^i b^j \mid i = j\ or\ i = 2j\}$

```
S -> M | D

M -> a M b | @

D -> a a D b | @
```

11. Converting CFG to PDA (not the other way) Standard conversion.

12. Consistency and completeness of CFGs with respect to a language

    given

    $$L = \{a^i b^{2i} \mid i \geq 0\},$$

    show that the grammar

```
S -> a S bb | @
```

is consistent and complete.

Answer: Consistency is easy (always double the number of b's). Completeness: If you assume, any such string of length n generated from this CFG, the next longer string has one more a to the left and two more b's to the right; so one more derivation steps gets it!

13. Ambiguity, inherent ambiguity.

    Is this grammar ambiguous?

    ```
    S -> if E then X else X | if E then X
    E -> q
    X -> p
    ```

    Is this grammar ambiguous?

    ```
    S ->  M | U
    M -> if E then M else M | O
    U ->  if E then STMT
        | if E then M else U
    O -> p
    E -> q
    ```

    What is the language of this grammar? Is the language of this grammar *inherently ambiguous*?

14. Show one use of this rule: `S -> if E then M else U`

    i.e. an actual if-then-else stmt. Draw a parse tree.

    Ans: `if q then if q then p else p else if q then p`

15. Be able to design simple Turing machines. Thoroughly study the `w#w` DTM and the `ww` NDTM. I'll ask variations in the exam. I may also ask you to step them for simple inputs.

16. Also read the basic sections on TMs – when they accept! What 'looping' means.

17. Closure results and how to use them correctly.

18. The *family* of regular languages is properly contained in the *family* of CFLs. This is a different idea from a *particular* CFL which always has a regular sub-language and a regular super-language.

19. Show that CFLs are closed under union, concatenation, and starring .

    Ans: Doing star is easy because we can use `U -> S U` @ — (this stars the language of S).

20. Be able to classify languages in the PL tutor as being regular or context-free or non context-free.

21. Know about the Chomsky Hierarchy (notes19).

22. Know where non-determinism makes a difference (for which machines).