CSE396 Problem Set 6 Answer Key Spring 2017

Assignment 6, due **in hardcopy** and **in class Thu. 4/6***
(**except R4 may submit Fri. 4/7 before 2:30pm to the Davis 300+ TA space since I am away)
Please write your name, Student ID#, **and recitation attended** atop your HW.

(1) Let $G = (\{ I \}, \{ s, d \}, \mathcal{P}, I)$ be the context-free grammar with rules $I \longrightarrow sI \mid sIdI \mid \epsilon$. Let T be the language of strings x over $\{ s, d \}$ such that for every prefix y of x, $\#s(y) \geq \#d(y)$. For a fact, $L(G) = T$—you are not asked to prove this. For further interpretation, note that if you interpret s = "spear" and d = "dragon," then T specifies the strings in which you "survive if you can hold arbitrarily many spears." More prosaically, if you interpret s as a left-paren and d as a right-paren, then T becomes the language of strings that might not yet be balanced, but can be closed out to be balanced by appending some number of ')' parens.

(a) Give both a parse tree and a leftmost derivation for each of the following strings in T ($3 \times 6 = 18$ pts.):

(i) $x_1 = sdssd$

(ii) $x_2 = ssddsdd$

(iii) $x_3 = ssddssdssdd$.

(b) Show that G is ambiguous, by finding an ambiguous string in T and giving two distinct derivation trees *or* two distinct leftmost derivations—your choice. (There are even shorter strings than the above. 9 pts., for 27 on the problem.)

Answer:

```
(b) Parses:  (i) x = sdssd.  Since we get a 0 diff with i = 2, we break
x =: s.d.ssd and begin I ==> sIdI ==> sdI.  Now recursing with x' = ssd,
we are in the "no internal diff = 0" case, so we do I ==> sI to begin it.
This leaves another "sd", so in all our parsing algorithm gives the
leftmost derivation I ==> sIdI ==> sdI ==> sdsI ==> sdssIdI ==> sdssdI ==> x.
(This is not the only leftmost derivation---there is also I ==> sIdI ==> sdI
==> sdsIdI ==> sdssIdI ==>^2 x, but it is the one given by our algorithm.)

(ii) x = sssddsdd.  Since the *final* diff is 0, we must begin with the
I ==> sIdI production (thus "internal" includes "n" here).  So we parse
x = s.u'.d.v with u' = ssddsd and v' = e.  With u' we have both an internal
and a final count of 0, but by the "least i" rule, we parse according to
the former: ssddsd = s.u".d.v" with u" = sd and v" = sd.  These pieces we
know how to get, so the whole leftmost derivation is:

I ==> sIdI ==> s sIdI I ==> s s sIdI dI I ==> sssdIdIdII ==> sssddIdII
==> sssdd sIdI IdII ==> sssddsdIIdII ==>^3 sssddsdd = x.
```

(iii) x = ssddssdssdd. Break first as ssdd.ssdssdd = s.sd.d.ssdssdd.
So begin I ==> sIdI ==> s sIdI dI ==> ssdIdI ==> ssddI. Now we need to
get ssdssdd from the final I. This is a no-internal-0 case, so we do
ssddI ==> ssdd sI, and then we need I ==>* sdssdd. This breaks with i = 2,
leaving pieces u' = "sd" and v = "ssdd" for recursive calls we've already
seen, so we finish the leftmost derivation by

ssddsI ==> ssddssIdI ==> ssddssdI ==> ssddssdsIdI ==> ssddssdssIdIdI

==>^2 ssddssdssdd. The parse trees are uniquely defined by these derivations.

(c) An ambiguous string (it happens to be the shortes one) is "ssd".
The intuitive reason for the ambiguity is, which sword killed the dragon?
Two different LM derivations:

I ==> sI ==> ssIdI ==> ssdI ==> ssd (first sword ignored)
I ==> sIdI ==> ssIdI ==> ssdI ==> ssd (first sword used)

(2) Let $G = (\{ S, A, B, C, D \}, \Sigma, \mathcal{R}, S)$ be the context-free grammar with $\Sigma = \{ a, b \}$ and rules $\mathcal{R} =$

$$\begin{aligned} S &\longrightarrow AD \mid bbC \mid SaBS \\ A &\longrightarrow BAB \mid \epsilon \\ B &\longrightarrow SB \mid b \\ C &\longrightarrow ACD \mid BA \mid DAS \\ D &\longrightarrow BaaC \mid \epsilon. \end{aligned}$$

(Since the period is not in Σ, it is just punctuation.)

Find a grammar G' without ϵ-rules such that $L(G') = L(G) \setminus \{ \epsilon \}$. Show clearly which variables in G are nullable. Is $\epsilon \in L(G)$? It is OK if your final G' looks "gross" so long as you show the steps of the algorithm clearly. (18 pts.)

Answer:

(1) S --> AD | bbC | SaBS
A --> BAB | \epsilon
B --> SB | b
C --> ACD | BA | DAS
D --> BaaC | \epsilon.

A and D are immediately nullable, then so is S by the rule S --> AD.
That in turn makes C nullable, by the rule C --> DAS. However, B remains
non-nullable, since (by SI) every string it derives has at least one 'b'.
Modified grammar, in BNF form---every variable other than B is optional:

```
    S --> [A][D] | bb[C] | [S]aB[S]
    A --> B[A]B | \epsilon
    B --> [S]B | b
    C --> [A][C][D] | B[A] | [D][A][S]
    D --> Baa[C] | \epsilon.

Conversion back to CFG, w/o writing new epsilon-rules and deleting old ones:

    S --> AD | A | D | bbC | bb | SaBS | aBS | SaB | aB
    A --> BAB | BB
    B --> SB | B | b
    C --> ACD | AC | AD | CD | A | C | D | BA | B |
          DAS | DA | DS | AS | D | A | S
    D --> BaaC | Baa.

Since S is nullable, yes \epsilon is in L(G), but this new grammar G'
gives L(G') = L(G) \ {\epsilon}.  Note that G' can be cleaned up by
deleting redundant "unit rules" including "B --> B".
```

(3) Let $G = (\{ S, A \}, \Sigma, \mathcal{R}, S)$ be the context-free grammar with $\Sigma = \{ a, b \}$ and rules $\mathcal{R} =$

$$\begin{aligned} S &\longrightarrow SS \mid ASa \mid aA, \\ A &\longrightarrow bA \mid SAa \mid a. \end{aligned}$$

And let T be the language of strings x such that $\#a(x)$ is even.

(a) Show that this grammar is *unsound* for T—that is, find a string in $L(G) \setminus T$. (6 pts.)

(b) Change one of the rules to make a grammar G' that is sound for T. Make it a "single-edit" change: that is, inserting, deleting, or changing just one character (terminal or variable symbol). Then explain why your new G' is sound. (12 pts. total)

(c) Is the original grammar comprehensive for T? OK, it's not: $\epsilon \in T \setminus L(G)$. So let's change the target language to be $T' = T \setminus \{\epsilon\}$ and revise the question: Is the original grammar comprehensive for T'? If you say yes, argue why as best you can; if you say no, give a *nonempty* string with an even number of a's that G does not generate, and explain why not. (12 pts., for 30 on this problem and 75 on the set)

```
(a) S ==> aA ==> aSAa ==> aaAAa ==>^2 aaaaa, but aaaaa is not in T.
(Many students will give 7 a's).

(b) Neatest fix, IMHO: change SAa to SAb.  Or just to SA.  Or...

(c) G is not comprehensive.  It does not derive "aab", and indeed derives only
strings that end in 'a'.  To see this "at-a-glance", augment the properties:
```

```
    P_S  +=  "...and every string I derive ends in an 'a'.
    P_A  +=  "...and every string I derive ends in an 'a'.
```

Two of the six rules supply the ending 'a' immediately, while the other four end in S or A, and the augmented P_S/P_A on RHS immediately satisfies the augmented P_S/P_A on LHS.
(Simpler still---because I changed the original version of the problem---it cannot derive "b".)