CSE396 **Problem Set 5 Answer Key** **Spring 2017**

(1) Define L to be the language of strings over $\Sigma = \{a, b\}$ that do not begin with aa and do not end in bb. (Note this is different from the language on Prelim 1 by having **and** in place of **xor**.) Find a PD set S of size 6 for L.

Answer: Among the first few strings in order $\epsilon, a, b, aa, ab, \ldots$, ϵ, a, b, ab, ba all belong to L, whereas aa and bb do not. Moreover, aa is "dead"—no string beginning with aa can belong to L—while bb can be made "live" if an a comes next since $bba \in L$. This already tells us that $z = a$ distinguishes aa from bb, so let's keep both of them for our set S.

We need 4 more strings. What if we try those first four, ϵ, a, b, ab, to make $S_0 = \{\epsilon, a, b, ab, aa, bb\}$? First thing to note is that each of the first four is distinguished from both of the last two automatically (by $z = \epsilon$) because the first four strings belong to L and the others do not. This already gives us $4 \times 2 = 8$ distinguished pairs, plus $aa \not\sim_L bb$ gives us 9 out of the $\binom{6}{2} = 15$ pairs we need to consider. Thus we are just left with the $\binom{4}{2} = 6$ pairs among $\{\epsilon, a, b, ab\}$. A little "hacking" distinguishes 5 of those pairs:

	ϵ	a	b	ab
ϵ		a	b	b
a			a	a
b				??
ab				

But how to separate b from ab? How many strings z do we need to consider? Well, you'll never find one: a little reflection shows that b and ab leave you in exactly the same "menatl state" regarding the language L, so in fact $b \sim_L ab$. So let'smove on and try ba in place of ab, i.e. $S = \{\epsilon, a, b, ba, aa, bb\}$. Since $ba \in L$, the distinctions from aa, bb are preserved, so we need only focus in on the 4×4 section of the 6×6 grid that was suggested:

	ϵ	a	b	ba
ϵ		a	b	aa
a			a	a
b				b
ba				

We distinguished b from ba by $z = b$. The only other revision that was tricky was that to separate ϵ from ba you can't use $z = a$ or $z = b$ anymore, but $z = aa$ works. Thus S is a PD set of size 6, so any DFA M such that $L(M) = L$ needs at least 6 states. And you can build an M with 6 states: Branch off a and b apart from the start state s and each other. Make aa go to "dead" and ab go to the same place as b, which is the middle state of the DFA called M_1 on the Prelim I answer key. Then you just need the other two states of M_1 to make 6.

(2) For the following languages L_1, L_2 over $\{0, 1\}$, design context-free grammars G_1, G_2 such that $L(G_1) = L_1$ and $L(G_2) = L_2$. You need not prove your grammars correct, but as usual you should include a few comments explaining how and why the grammars work correctly. ($2 \times 12 = 24$ pts., for 42 total on the set)

1. $L_1 = \{0^m 1^n 0^n 1^m : m \geq 1, n \geq 0\}$,
2. $L_2 = \{x0y : \#0(x) = \#1(y)\}$.

Answer: G_1 has rules $S \rightarrow 0S1 \mid 0T1$, $T \rightarrow 1T0 \mid \epsilon$. The design pattern is "nesting"—S handles the outer 0^m and 1^m layers, and the fact that the dropdown to T comes with $0T1$ ensures $m \geq 1$. The inner T handles $1^n 0^n$ with $T \rightarrow \epsilon$ allowing $n = 0$.

For G_2 the key idea is that the displayed 0 between x and y—which is a movable pain-in-the-neck when you are trying to parse strings—is derived when the grammar halts a recursion on S_2. So the idea of "on the left of S_2" rigorously becomes "on the left of *that* 0." It follows that we can add a single 1 on the left of S_2 without changing the balance, and likewise add a single 0 on the right of S_2. But whenever we add a 0 on the left of S_2, we have to balance it immediately with a 1 on its right. This shows thatthe following grammar is *sound*:

$$S_2 \rightarrow 0 \mid 1S_2 \mid S_2 0 \mid 0S_2 1$$

Is it comprehensive? Let any $w \in L$ be given, and write $w = x0y$ such that $\#0(x) = \#1(y) = k$, say. Mark the k-many 0s in x and the k-many 1's in y. Pair them up in a nested fashion. In-between the 0s in x you have a "filling" of however-many 1s, and in-between the 1s in y you have fillings of 0s. Working from the outside-in, use the $S_2 \rightarrow 1S_2$ and $S_2 \rightarrow S_2 0$ rules to handle the "fillings" until you reach a marked pair on both the left and right. Then you use $S_2 \rightarrow 0S_2 1$ to generate that pair. Rinse, repeat, enjoy, and finally to $S_2 \rightarrow 0$ to finish w. So $L(G_2) = L_2$.

There was no requirement to use only one variable. The explanation above is arguably clearer if we apply it to this grammar instead:

$$S_2 \rightarrow 0 \mid TS_2U \mid 0S_2 1, \quad T \rightarrow 0T \mid \epsilon, \quad U \rightarrow U1 \mid \epsilon.$$

Then using the notation in lecture for the language of each variable, we have $L_T = 0^*$ and $L_U = 1^*$; these are the classic ways of making lists (and for those in CSE305, of simulating the BNF "star" operator).