CS 21 Decidability and Tractability **Winter 2014**

Midterm Solutions

Out: February 12

1. (a) The language L_1 is not context-free.

   To see it is not context-free, we use the CFL pumping lemma: let $w = a^p b^p c^p d^p$, and consider the ways w can be written as $w = uvxyz$. If u or v straddles the boundary between characters, then pumping results in an out-of-order string (not in the language). Otherwise, we use the fact that $|vxy| \leq p$ to argue that v and y can only be within a single block or two adjacent blocks of characters. In all such cases, pumping on v and y results in a string not in the language. We conclude that L_1 is not context free.

   (b) The language L_2 is context-free but not regular. To see it is context free, consider the NPDA that first pushes a's onto the stack until it sees the first b, then pushes b's onto the stack until it sees the first c, then pops b's from the stack as it reads c's, and finally pops a's from the stack as it reads d's. To see that it is not regular, we use the pumping lemma. Let $w = a^p b^p c^p d^p$ and consider the ways w can be written as xyz. If string y straddles the boundary between characters, then pumping on y results in an out-of-order string (not in the language). Otherwise pumping on y increases the number of only a single character type, again resulting in a string not in the language. We conclude that L_2 is not regular.

   (c) The language L_3 is regular: it is the union of the regular languages $a^{1000}a * b * c*$ and the finite language $\{a^n b^n c^n : n < 1000\}$.

2. (a) Decidable. We will reduce this problem to E_{CFG} (emptiness of context-free-grammars), which we saw in lecture was decidable. Given E we first build the DFA that recognizes language A, the complement of $L(E)$. This is possible because regular languages are closed under complement. We also know how to construct a NPDA that recognizes the language $B = L(G) \cap A$ (from the hint: Sipser problem 2.18). We now check if B is empty. From lecture we know that emptiness of CFGs is decidable. Moreover the complementation step and the intersection step are all computable transformations. Finally, note that B is empty iff $L(G) \subseteq L(E)$, so the language CFG-IN-REG is decidable.

   (b) Undecidable. We reduce ALL_{CFG} to REG-IN-CFG. Set $E = \Sigma^*$. Given an instance G of ALL_{CFG}, we produce the pair (E, G). If $G = \Sigma^*$ then clearly $L(E) \subseteq L(G)$; if $G \neq \Sigma^*$ then $L(E) \not\subseteq L(G)$. Therefore we have reduced ALL_{CFG} to REG-IN-CFG, and we know from lecture that ALL_{CFG} is undecidable.

3. Suppose there exists a decidable language D such that $L_1 \cap D = \emptyset$ and $L_2 \subseteq D$, with a corresponding TM M_D. Then considering $M_D(\langle M_D \rangle)$ we come to a contradiction as follows.

   Suppose $M_D(\langle M_D \rangle)$ accepts; i.e. $\langle M_D \rangle$ is in the language D. Then by the definition of L_1, $\langle M_D \rangle$ is in the language L_1, which contradicts the fact that $L_1 \cap D = \emptyset$.

Suppose $M_D(\langle M_D \rangle)$ rejects; i.e. $\langle M_D \rangle$ is not in the language D. Then by the definition of L_2, $\langle M_D \rangle$ is in the language L_2, which contradicts the fact that $L_2 \subseteq D$.

4. (a) Let G be a right-linear CFG. We will construct a NFA M recognizing $L(G)$. Our machine M will have a single state for each non-terminal in the grammar, a distinguished "accept" state, and other states. The start state of M is the state corresponding to the start symbol in the grammar. For each transition of the form:

$$A \to x_1 x_2 \dots x_n B$$

we add $n-1$ states $s_1, s_2, \dots, s_{n-1}$ "linking" A to B, with a transition from A to s_1 labelled x_1, a transition from s_1 to s_2 labelled x_2, etc..., and a transition from s_{n-1} to B labelled x_n.

For each transition of the form:

$$A \to x_1 x_2 \dots x_n$$

we add $n-1$ states $s_1, s_2, \dots, s_{n-1}$ "linking" A to the accept state, with a transition from A to s_1 labelled x_1, a transition from s_1 to s_2 labelled x_2, etc..., and a transition from s_{n-1} to the accept state labelled x_n.

Now, if M accepts a string w, then the sequence of "non-terminal" states it traverses to reach the accept state dictates a derivation of w in the grammar. In the other direction, if w has a derivation in the grammar, then it must arise from applying a sequence of rules of the first type, followed by a single application of a rule of the second type. This derivation dictates a path from the start state of M to the accept state, and thus M accepts w.

(b) Given a FA M, we construct a right-linear CFG G as follows. The non-terminals of G are exactly the states of M. The start symbol of G is the start state of M. For each transition in M from state A to state B, labelled with the symbol x, we add the following rule: $A \to xB$. For each transition from state A to an accept state B, labelled with the symbol x, add the following rule: $A \to x$.

If M accepts a string w, then the sequence of states traversed from the start state to an accept state dictates a derivation of w in the grammar. In the other direction, if w has a derivation in the grammar, then this derivation dictates a path from the start state of M to an accept state (since it must end with a rule of the second type).

(c) Consider the following linear CFG G:

$$\begin{aligned} S &\to 0A|1B|\epsilon \\ A &\to S0 \\ B &\to S1 \end{aligned}$$

We prove two claims to establish that this indeed generates exactly the palindrome language L. First, we claim that $L \subseteq L(G)$. Let w be a palindrome, so $w = xx^R$, where x^R denotes the revers of string x. We prove the claim by induction on $|x|$. If $|x| = 0$ then $x = w = \epsilon$ and indeed S derives ϵ. Now assume that for all x' with length $< n$, the string $w' = x'(x')^R$ is in $L(G)$. Then we can derive S as follows: if the first character of x is

0, then we use $S \Rightarrow 0A \Rightarrow 0S0 \Rightarrow^* 0x'(x')^R0 = xx^R$ where the $\Rightarrow^*$ follows by induction; similarly, if the first character of x is 1, then we use $S \Rightarrow 1B \Rightarrow 1S1 \Rightarrow^* 1x'(x')^R1 = xx^R$ where the $\Rightarrow^*$ follows by induction.

Second, we claim that $L(G) \subseteq L$. Consider a derivation of some string w. The proof is by induction on the length of the derivation. If the length is 1, then the only string w could be is ϵ, and $\epsilon \in L$. Otherwise, assume all derivations of length $< n$ result in strings in L and consider the first step in a length n derivation. If it is $S \to 0A$, then the only rule that can be applied next is $A \to S0$, so we deduce that the derivation has the form $S \Rightarrow 0A \Rightarrow 0S0 \Rightarrow^* 0w'0 = w$. Now since $S \Rightarrow^* w'$ in fewer than n steps, we know that w' is palindrome, and thus $w = 0w'0$ is as well. Otherwise, the first step in the derivation is $S \to 1B$, then the only rule that can be applied next is $B \to S1$, so we deduce that the derivation has the form $S \Rightarrow 1B \Rightarrow 1S1 \Rightarrow^* 1w'1 = w$. Now since $S \Rightarrow^* w'$ in fewer than n steps, we know that w' is palindrome, and thus $w = 1w'1$ is as well. We conclude that $w \in L$ and then that $L(G) \subseteq L$.

5. We use the idea used in the proof showing that a language is R.E. if some enumerator enumerates it. Specifically, let M_A be the machine recognizing language A and let M_B be the machine recognizing language B. We fix some enumeration $s_1, s_2, \ldots$ of Σ^* in lexicographic order. Our machine M for A/B does the following for $i = 1, 2, 3, \ldots$: in parallel, simulate M_B on the inputs $s_1, s_2, \ldots, s_i$ and machine M_A on the inputs $xs_1, xs_2, \ldots, xs_i$ for i steps. If during any of the simulations, M_B accepts some s_j and M_A accepts the corresponding xs_j, then M halts and accepts.

   Suppose x is in A/B. Then there is some y which is accepted by M_B after n_B of steps, for which xy is accepted by M_A after n_A steps. If y appears in the lexicographic order at position m (i.e. $s_m = y$), then our machine M will accept when $i = \max(m, n_a, n_b)$. Thus every string $x \in A/B$ is accepted by M.

   Conversely, if M accepts some string x, then it can only have done so because for some value i, there was some string s_j with $j \leq i$ for which M_B accepted s_j and M_A accepted xs_j. Therefore there is a string $y = s_j$ in B for which xy is in A, which implies $x \in A/B$ as required.