CS 21 Decidability and Tractability **Winter 2025**

Midterm Solutions

Out: February 12

If you have not yet turned in the Midterm you should not consult these solutions.

1. (a) The language L_1 is context free but not regular. To see it is context free, consider the NPDA that first pushes a \$ marker onto the bottom of the stack. Then it does the following: as it reads a's it pushes a's onto the stack until it sees the first b, then pops a's from the stack as it reads b's. If the \$ marker is encountered, it begins pushing b's for each b read. When the first c is encountered, we know that either (1) the stack has a b at the top, and the number of b's on the stack equals the excess of b's over a's in the string read so far, or (2) the stack has an a at the top, and the number of a's on the stack equals the excess of a's over b's in the string read so far or (3) the stack has a \$ at the top, and the number of a's equals the number of b's in the string read so far.

   In the first case, as we read c's, we pop b's, until we get to \$, at which point we know $i + k \geq j$ and we accept (see below). Otherwise, if we reach the end of the string (or see another character besides a c) we reject.

   In the second case we know that $i > j$ and we accept (see below).

   In the third case, we know that $i = j$ and we accept (see below).

   In all cases, when we say "we accept" it means we enter a dedicated portion of the machine that reads to the end of the input and accepts *as long as the characters are in order* – b's before c's. Otherwise we reject. And when we say "we reject" it just means we enter a distinguished reject state and stay there.

   To see that L_1 is not regular, we use the pumping lemma: let $w = a^p b^{2p} c^p$, and consider the ways w can be written as $w = xyz$. If y straddles the boundary between characters, then pumping results in an out-of-order string (not in the language). And, if y is within any single type of character, then pumping on it results in a string not in the language: if y is within a's or c's then pumping down (i.e., replacing y with y^0) gives this string; if y is within b's, then pumping up (i.e., replacing y with y^2) gives the desired string.

   (b) Language L_2 is regular. It is the union of the languages $a^{1001}a^*b^*c^*$ and the *finite* language $\{a^n b^n c^n : n < 1000\}$ (which is regular because it is finite).

   (c) Language L_3 is not context free. Let p be the pumping lemma, and define $w = a^q b^q c^q$ for $q \geq \max\{p, 1001\}$. Consider the ways w can be written as $uvxyz$. If v or y straddle the boundary between characters, then pumping results in an out-of-order string, which is not in the language. Otherwise, pumping on v and y results in a string $a^r b^s c^t$ with r, s, t not all equal *and with* $r > 1000$, which is not in the language.

2. (a) Language L is co-RE. To see this we will consider the language

$$\text{co} - L = \{\langle M \rangle : \text{Turing Machine } M \text{ has no unreachable states}, \}$$

   and show that this language is RE. Our recognizer for $\text{co} - L$ operates as follows. Given M, it "simulates M in parallel" on all strings in Σ^*, accepting after it has observed every state being visited. More precisely, if $w_1, w_2, w_3, \ldots$ is an enumeration of Σ^* in lexicographic order, then our recognizer does this: for $j = 1, 2, \ldots$, we simulate M for j steps on strings $w_1, w_2, \ldots, w_j$. If in some round j we see that every state of M has been visited, we halt and accept. Clearly, if every state of M is reachable, then there is a finite set of strings that together reach every state (pick one for each state, for example). Thus there is some round j in which we will simulate M on all of these strings, for enough steps for M to reach each state, and we will accept. Otherwise, if some state is *unreachable*, then this simulation will go on forever, and never accept.

(b) We show that L is undecidable by reduction from E_{TM}. Given an instance $\langle M \rangle$, we will produce a machine M'. This machine will have the property that $L(M') = L(M)$, *and* before M' accepts, it visits all of its states. To do this, start out by making M' identical to M. Let $q_0, q_1, q_2, \ldots, q_m$ be an enumeration of all of the states of M excluding q_{accept}. Replace any transition to q_{accept} with an identical transition to a new state q_{tour1}. State q_{tour1} has transitions $a \to \%, R$ to new state q_{tour2} for each $a \in \Gamma \cup \{\_\}$. State q_{tour2} has transitions $a \to \%, L$ to new state q_{tour3} for each $a \in \Gamma \cup \{\_\}$. Finally there is a transition $\% \to \%, R$ from state q_{tour3} to state q_0, and a transition $\% \to \%, L$ from state q_0 to state q_1, and a transition $\% \to \%, R$ from state q_1 to state q_2, and a transition $\% \to \%, L$ from state q_2 to state q_3, and so on, until there is finally a transition $\% \to \%, L$ to state q_{accepts}.

The effect of this modification is as follows. If w is not accepted by M, then q_{accept} is never reached, no % symbol is ever written on the tape, and thus M' also does not accept w. If w *is* accepted by M, then M' also accepts w, but it first writes two consective % symbols on the tape, leaving the head positioned over the first one. It then moves left and right repeatedly over these two symbols while visiting every state before finally entering q_{accept} and halting. The effect is that every state is visited when accepting w. We see that $L(M) = L(M')$. And, if $L(M') = \emptyset$, q_{accept} is an unreachable state of M'. Conversely, if $L(M') \neq \emptyset$, then some string is accepted by M' and every state of M' is reachable. Thus $\langle M' \rangle$ is in L iff $L(M) = \emptyset$, which completes the reduction from E_{TM} to L, showing that L is indeed undecidable. We also should note that constructing M' from M is an easy modification entailing adding some states and transitions to M (and thus is computable).

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:

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

   we add $n - 1$ states $s_1, s_2, \ldots, s_{n-1}$ "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:

$$A \to x_1 x_2 \ldots x_n$$

   we add $n - 1$ states $s_1, s_2, \ldots, s_{n-1}$ "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:

$$\begin{aligned} S &\to aT|\epsilon \\ T &\to Sb \end{aligned}$$

We claim that $L(G) = \{a^n b^n : n \geq 0\}$ (which is not regular as seen in class, so the "linear" constraint on CFGs is not sufficient to force the language to be regular). We first show that all strings of this form are generated by G. We prove this by induction on n: assume all strings in L of length $< n$ are derivable; the base case with $n = 0$ is trivially true, and then to derive string $a^n b^n$, we use the derivation $S \Rightarrow 0T \Rightarrow 0S1 \Rightarrow^* aa^{n-1}b^{n-1}b$, where the last step is possible by induction.

In the other direction, we prove by induction on the length of the derivation strings derivable from S are in L. Our induction hypothesis is the stronger statement: all strings derivable in $< n$ steps from S are in L, and all strings derivable in $< n$ steps from T are of the form $a^{m-1}b^m$. Clearly the base case (deriviation of length 1, which can only derive ϵ) is true. Now consider a derivation of length n. If the first step is $S \to aT$, then we know by induction that in the remainder of the derivation $aT \Rightarrow^* aa^{m-1}b^m$ for some m, so S derives a string in the language as required. If the first step is $T \to Sb$, then we know by induction that in the remainder of the derivation $Sb \Rightarrow^* a^m b^m b$ for some m, so T derives a string of the required form.

5. Let M be a recognizer for L. We are given an input $\#x_1\#x_2\#\cdots\#x_k\#$ for some $k \geq 0$. We simulate M on each x_i *in parallel*, and accept as soon as at least 50 of these simulations accept. Specifically, we do the following for $j = 1, 2, 3, \ldots$: simulate M on each x_i for j steps. If in some round j we observe that at least 50 of the simulations halt and accept, then we halt and accept.

   Now, it is clear that if at least 50 of the x_i are in L, then for some j (corresponding to the maximum number of steps for M to accept any one of these x_i) the new machine will accept. Otherwise, we will never experience accepts for at least 50 of the x_i and the machine will not accept.