

Midterm Solutions

Out: February 13

1. (a) The language L_1 is not context-free. We use the pumping lemma. Let p be the pumping length, and consider the string $w = a^{2^\ell}$, for $2^\ell > p$. When written as $w = uvxyz$, with $|vy| = k$, and $0 < k \leq p$, we see that $w' = uv^2xy^2z$ has length $2^\ell + k$. But since $k \leq p < 2^\ell$, we know that $2^\ell < 2^\ell + k < 2^{\ell+1}$ and thus w' is not in the language.
- (b) $L = \{a^i b^j c^k d^\ell : i + j + k + \ell = 7x + 11y \text{ for integers } x, y \geq 0\}$ is regular. We know that regular languages are closed under intersection (they are closed under union, and they are closed under complement by swapping accept and non-accept states). One particularly nice way to see that L is regular is to observe that it is the intersection of the following two regular languages ($\Sigma = \{a, b, c, d\}$):

$$L_1 = (\underbrace{\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma}_{7 \text{ times}} \cup \underbrace{\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma}_{11 \text{ times}})^*$$

$$L_2 = a^* b^* c^* d^*$$

One could also build a FA recognizing this language.

- (c) Language L_3 is not context-free. We use the pumping lemma. Let p be the pumping length, and consider the string $w = a^p b^p c^p d^p$. The pumping lemma says we can write w as $uvxyz$ with $|vxy| \leq p$ and $|vy| \geq 1$. If v or y straddles the boundary between two different symbols, then it is clear that pumping produces a string not in the language. So, v and y must each be wholly within a block of similar characters, and the fact that $|vxy| \leq p$ ensures that v and y are either in the same block or adjacent blocks. Thus, if we pump, we either change the number of a single type of character, which produces a string not in the language, or we change the number of characters in adjacent blocks, which also produces a string not in the language. In all cases, we produce a string not in the language.
2. (a) Language L_1 is undecidable. The proof is by reduction from the language ALL_{CFG} which we saw in class was undecidable. Given an instance $\langle G \rangle$ of ALL_{CFG} , our reduction produces the instance $\langle G_1, G_2 \rangle$ where G_1 is just grammar G and G_2 is the grammar Σ^* . To be precise, G_2 is the grammar with rules $S \rightarrow aS$ for every $a \in \Sigma$ and $S \rightarrow \epsilon$. Clearly, if $L(G) = \Sigma^*$, then $L(G_1) = L(G_2)$, so “yes maps to yes.” And, if $L(G) \neq \Sigma^*$, then clearly $L(G_1) \neq L(G_2)$, so “no maps to no.”
 - (b) Language L_2 is decidable. By the Pumping Lemma for CFGs we know that if $L(G)$ contains strings of length p or greater, then it contains infinitely many strings (by pumping). Thus, to decide whether $L(G)$ is finite, we only need to check if it contains a string of length p or greater. The language L_p of all strings up to length $p - 1$ is regular, since it is described by the regular expression $(\Sigma \cup \epsilon)^{p-1}$. Regular languages are closed under

complement, so $\overline{L_p}$, the language of all strings of length p or greater, is also regular. The intersection of $L(G)$ and $\overline{L_p}$ is context free, by 2.18(a) in Sipser. Note that it is empty iff $L(G)$ contains *no* string of length p or greater. And, as noted above, $L(G)$ contains *no* string of length p or greater iff it is finite.

We need to verify that the transformations above are all computable. First, we need to actually compute the pumping length p , but from the proof of the Pumping Lemma, we know that $p = b^{m+2}$, where b is the maximum number of nonterminals on the right-hand-side of a production in G , and m is the number of nonterminals in G . Thus we can easily compute p from looking at the given grammar G .

Then we need to construct a NPDA from G , and a DFA for $\overline{L_p}$, both of which are straightforward to compute. Finally, we see from the proof of 2.18(a) in Sipser that producing the NPDA for the intersection of $L(G)$ and $\overline{L_p}$ is also straightforward to compute. We convert this NPDA into a grammar as done in class (also easy to compute), and then check emptiness of this grammar using the algorithm from class for the language E_{CFG} .

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:

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

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:

$$A \rightarrow x_1 x_2 \dots x_n$$

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: $A \rightarrow xB$. For each transition from state A to an accept state B , labelled with the symbol x , add the following rule: $A \rightarrow x$.

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 :

$$\begin{aligned} S &\rightarrow aT \\ T &\rightarrow Sb \\ T &\rightarrow b \end{aligned}$$

We claim that $L(G) = \{a^n b^n : n \geq 1\}$, which is not regular as seen in class. We first show that all strings of this form are generated by G . We prove this by induction on n ; the induction hypothesis states that all sentential forms $a^n T b^{n-1}$ for $n \geq 1$ are derivable from S . The base case of $n = 1$ is trivial by the derivation $S \Rightarrow aT$. Then, since $S \Rightarrow^* a^n T b^{n-1}$ by induction, we obtain the derivation $S \Rightarrow^* a^n T b^{n-1} \Rightarrow a^n S b^n \Rightarrow a^{n+1} T b^n$ as required. An application of the production $T \rightarrow b$ to all of these sentential forms yields the strings $a^n b^n$ for all $n \geq 1$.

In the other direction, we prove by induction on the length of the derivation that all sentential forms derivable from S are of the form $a^n T b^{n-1}$ for some $n \geq 1$ or $a^n S b^n$ for some $n \geq 1$ or $a^n b^n$ for some $n \geq 1$. The base case follows from considering all derivations from S up to length 2: $S \Rightarrow aT$, $S \Rightarrow aT \Rightarrow ab$, and $S \Rightarrow aT \Rightarrow aSb$. Applying one additional production to $a^n T b^{n-1}$ yields $a^n S b^n$ or $a^n b^n$, and applying one additional production to $a^n S b^n$ yields $a^{n+1} T b^n$, which completes the induction step. Thus the only strings of terminals derivable from S are strings of the form $a^n b^n$ for $n \geq 1$.

5. Let M be a recognizer for L . We are given an input $\#x_1\#x_2\#\dots\#x_k\#$ for some $k \geq 0$. We simulate M on each x_i in parallel, and accept as soon as more than $k/2$ of these simulations accept. Specifically, we do the following for $j = 1, 2, 3, \dots$: simulate M on each x_i for j steps, and if more than $k/2$ of the simulations halt and accept, then we halt and accept.

Now, it is clear that if a majority of the x_i are in L , then we will accept by the time j is the maximum of the number of steps required for M to accept the x_i that are in L (or before). Otherwise, we will never experience more than $k/2$ acceptances among the simulations, for any j , so we will never accept.