CSE 103 Computational Models
Fall 2021 Prof. Fremont HW 5

(3 questions, 160 points total)

1. (40 pts.) Designing a PDA

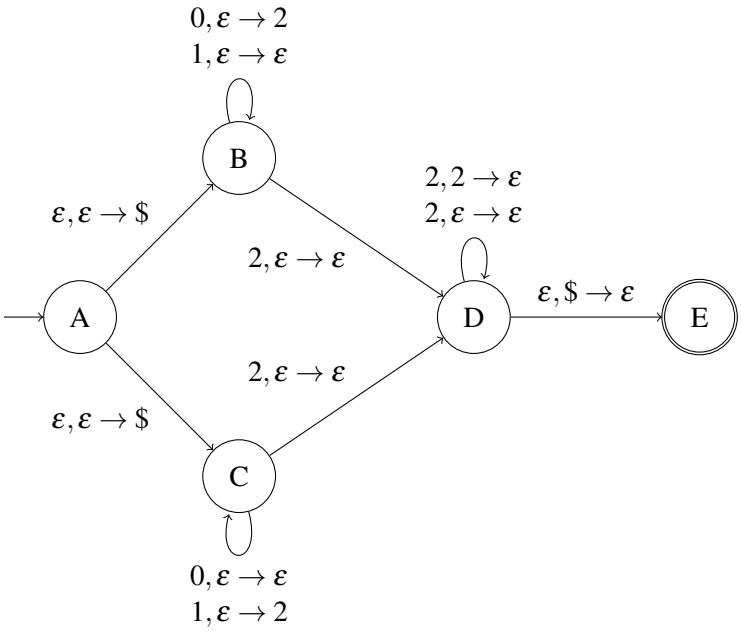
Given a binary string $x \in \{0,1\}^*$, let's write $Z(x)$ for the number of zeroes in x and $O(x)$ for the number of ones. Design a PDA with input alphabet $\Sigma = \{0,1,2\}$ recognizing the language

$$L = \{x2^k \mid x \in \{0,1\}^* \text{ and } k > \min(Z(x), O(x))\};$$

that is, strings consisting of a binary string followed by some number of 2s, such that 2 is not the least-common digit overall (for example, $2, 12, 00022, 01022 \in L$, whereas $012, 21 \notin L$).

(*Hint:* Remember that PDAs can be nondeterministic.)

Solution: If we only had to compare the number of 2s against the number of 0s, we could use the approach we've seen before: push the 0s onto the stack, and then pop one whenever we see a 2, making sure we have enough 2s to completely clear the stack (plus one extra 2, since we need to have strictly more 2s than 0s). However, this approach won't work to keep track of *both* the number of 0s and the number of 1s we've seen, since we only have one stack and must decide whether to pair 2s with 0s or with 1s. The key idea is that since PDAs can be nondeterministic, we can *guess* which number to pair against: since $k > \min(Z(x), O(x))$ if and only if either $k > Z(x)$ or $k > O(x)$, we can build separate PDAs for each of the latter conditions and then nondeterministically choose which one to use. Combining some common parts of the two PDAs, this gives us:



In state A, we can choose to go to either state B or state C, pushing a \$ on the stack so that we can detect an empty stack later. In state B, we push a 2 on the stack for every 0 we see, ignoring any 1s; state C does likewise with 0 and 1 reversed. When we see a 2, we move to state D, whereupon we can not have any more

0s or 1s, but we can read 2s, with the (nondeterministic) option to pop a 2 off the stack for each one. If we clear the stack, so that \$ is on top, we can transition to state E and accept if there are no more input symbols.

(*Note:* There are many slightly-different ways of setting up the PDA that work just as well: for example, we could have moved the $2, \varepsilon \to \varepsilon$ self-loop at state D to state E; we could also have pushed 0s and 1s onto the stack unchanged rather than converting them to 2s, as long as we had transitions allowing us to pop either of them when we read a 2.)

If $w \in L$, then it is of the form $x2^k$ with $x \in \{0,1\}^*$ and either $k > Z(x)$ or $k > O(x)$. In the first case, if we transition from A to B, after reading x the stack will have $Z(x)$ copies of 2 on it; we can then read in $1 + Z(x) \leq k$ copies of 2 from the input using the transition from B to D and the self-loop at D. After this, the stack will only have a \$ on it; if there are any further copies of 2 in x we can read them using the $2, \varepsilon \to \varepsilon$ self-loop before transitioning to E and accepting. If instead $k > O(x)$, we can transition from A to C and build an accepting path in an exactly analogous way to above, just exchanging 0s and 1s. So in either case our PDA will accept w.

Conversely, if our PDA accepts a word $w \in \Sigma$, then it must reach state D with an empty stack (other than the \$). If the accepting path went through state B, then upon entering state D for the first time, the stack will contain $Z(x)$ copies of 2, and since the only transition that pops a symbol in state D is $2, 2 \to \varepsilon$, the input must contain at least $Z(x) + 1$ copies of 2 (where the $+1$ comes from the transition from B to D). Likewise, if the accepting path went through state C, the input must contain at least $O(x) + 1$ copies of 2. Furthermore, any 2s in the input must come after the 0s and 1s, since only transitions leading to state D can read a 2. So w must be of the form $x2^k$ where $x \in \{0,1\}^*$ and $k > \min(Z(x), O(x))$.

Therefore, the language of our PDA is L, as desired.

2. (60 pts.) Closure properties of context-free languages

Recall that we proved that the regular languages are closed under various operations, including union, concatenation, Kleene star, intersection, and complement. Let's see what happens when these operations are applied to context-free languages.

(a) Suppose we have two CFGs G_1 and G_2, with start variables S_1 and S_2 respectively. What rule(s) should we add to get a grammar (say with start variable S) for $L(G_1) \cup L(G_2)$?

Solution: We should add the rule $S \to S_1 \mid S_2$. Then we can derive any $w \in L(G_1)$ by first applying the rule $S \to S_1$ and then proceeding according to the derivation of w in G_1, and similarly for any $w \in L(G_2)$. Conversely, any derivation in G must start with either the rule $S \to S_1$ or $S \to S_2$ (since these are the only rules with S on the left-hand side), so any string in $L(G)$ is a member of either $L(G_1)$ or $L(G_2)$ (or both).

(*Note:* This argument assumes that G_1 and G_2 have no variables in common: otherwise, you might be able to derive something from S_1 using some of the rules from G_2, which would then not necessarily be in either $L(G_1)$ or $L(G_2)$. We can always make sure that the variables of G_1 and G_2 are disjoint by renaming variables as needed.)

(b) If instead we want a grammar for $L(G_1)L(G_2)$ (the concatenation of any word in $L(G_1)$ with any word in $L(G_2)$), what rule(s) should we add?

Solution: We add the rule $S \to S_1 S_2$. An argument along the lines of that in part (a) then shows that $L(G)$ consists exactly of those strings which are concatenations of a word in $L(G_1)$ and a word in $L(G_2)$.

(c) How about the Kleene star $L(G_1)^*$?

Solution: We add the rule $S \to SS_1 \mid \varepsilon$. Then we can apply the rule $S \to SS_1$ repeatedly $n \geq 0$ times to derive the string SS_1^n, then apply $S \to \varepsilon$ to get S_1^n, and finally expand each copy of S_1 into a word of $L(G_1)$. So $L(G_1)^n \subseteq L(G)$ for every $n \geq 0$, and so $L(G)^* = \cup_{n \geq 0} L(G_1)^n \subseteq L(G)$. Conversely, any $w \in L(G)$ is either ε or a concatenation of a finite number of words from $L(G_1)$, so $L(G) \subseteq L(G_1)^*$.

(d) The previous three parts show that the context-free languages are closed under the regular operations (union, concatenation, and Kleene star). Prove that the context-free languages are *not* closed under intersection, i.e., there are context-free languages L_1 and L_2 such that $L_1 \cap L_2$ is not context-free.

(*Hint:* Start by arguing that the languages $\{0^n 1^n 2^* \mid n \geq 0\}$ and $\{0^* 1^n 2^n \mid n \geq 0\}$ *are* context-free.)

Solution: We gave a grammar for $0^n 1^n$ in class, and 2^* is context-free since it is regular, so by closure under concatenation (from part (b)) so is L_1. Alternatively, we can write down a grammar for it directly:

$$\begin{aligned} S &\to AB \\ A &\to 0A1 \mid \varepsilon \\ B &\to 2B \mid \varepsilon \end{aligned}$$

Similar arguments show that L_2 is also context-free.

Now if the context-free languages were closed under intersection, the language $L_1 \cap L_2 = \{0^n 1^n 2^n \mid n \geq 0\}$ would be context-free, but we proved in class that it is not. So the context-free languages are not closed under intersection.

(e) Prove that the context-free languages are not closed under complement.

(*Hint:* Assume that CFLs *are* closed under complement, and get a contradiction with part (d).)

Solution: For any context-free languages L_1 and L_2 we have $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ (in fact this is true for any two languages). If the context-free languages were closed under complement, then $\overline{L_1}$ and $\overline{L_2}$ would also be context-free, and since CFLs are closed under union as we saw in part (a), $\overline{L_1} \cup \overline{L_2}$ would

be context-free. Applying closure under complement again, $\overline{\overline{L_1} \cup \overline{L_2}}$ would be context-free, but this is the same as $L_1 \cap L_2$, and therefore the intersection of any two CFLs would be context-free. This contradicts part (d).

3. (60 pts.) The pumping lemma for context-free languages

Let's prove that the language $L = \{0^i 1^j 2^k \mid i, j, k \geq 0 \text{ and } j < i, k\}$ is not context-free.

(a) Given a pumping length $p \geq 1$, choose an appropriate word $w \in L$ with $|w| \geq p$.

Solution: One possible word is $w = 0^{p+1}1^p 2^{p+1}$. As required, we have $w \in L$ and $|w| = 3p+2 \geq p$.

(b) Given a decomposition $w = uvxyz$ with $|vy| > 0$ and $|vxy| \leq p$, we need to pick a $k \geq 0$ such that $uv^k xy^k z \notin L$. Show how to do this in the case when vy contains a 1.

(*Hint:* Can vy also contain both a 0 and a 2?)

Solution: Since $|vxy| \leq p$, it is not possible for vxy to contain all three symbols 0, 1, and 2, since the 0s and 2s in w are separated by p copies of 1. So vy either contains no 0s or no 2s (or both): let us call (one of) the missing symbols m. If we put $k = 2$, then uv^2xy^2z will contain strictly more 1s than $uvxyz = w$, since vy contains a 1 by assumption, but it will contain the same number of ms. So uv^2xy^2z will have at least $p+1$ 1s but exactly $p+1$ ms, and therefore will not be in L.

(c) Show how to pick $k \geq 0$ so that $uv^k xy^k z \notin L$ in the other case, namely when vy contains no 1s.

Solution: If vy contains no 1s, then since $|vxy| \leq p$, either vy consists only of 0s or only of 2s: we can't have both, since the 0s and 2s are separated by p copies of 1. Say that vy is made up of ms. Then picking $k = 0$, the string $uv^k xy^k z = uxz$ contains strictly fewer ms than w, since $|vy| > 0$, and the same number of the other two symbols. But then uxz has at most p copies of m, while it has exactly p copies of 1, and therefore will not be in L.