CS 21 Decidability and Tractability **Winter 2018**

Solution Set 2

Posted: January 24

If you have not yet turned in the Problem Set, you should not consult these solutions.

1. We build a NPDA for L. There will be four states, labeled "match," "add-one", "add-two," and "accept" plus three separate state S, R and R'. State S is the start state and "accept" is the (single) accept state. There is a transition labeled $\epsilon, \epsilon \to \$$ from S to R (this marks the bottom of the stack with \$, as is customary). Then there is a transition $d, \epsilon \to d$ from R to R' for each $d \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ (we disallow leading zeros, so the first digit read must be a non-zero). There is then a self-loop on R' labeled with $d, \epsilon \to d$, for each $d \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and a transition labeled $\#, \epsilon \to \epsilon$ from R' to state "add-two". These transitions cause the digits to the left of the first # to be pushed onto the stack, and the machine ends in state R'.

   Now we have a transition from "add-two" to "match" labeled with $d + 2, d \to \epsilon$ for $d \in \{0, 1, 2, 3, 4, 5, 6, 7\}$. This handles the case in which there is no carry, so the least significant digit (which appears first in $N'(i+2)$) is 2 larger than the least significant digit of $N(i)$, and the remaining digits must simply match.

   We also have transitions from "add-two" to "add-one" labeled with $0, 8 \to \epsilon$ and $1, 9 \to \epsilon$. These handles the two cases in the least significant digit that generate carries.

   We have a transition from "add-one" to "add-one" labeled with $0, 9 \to \epsilon$. These will match 0's at the beginning of $N'(i+1)$ to 9's on the stack, for the zero or more 9's leading up to the least significant digit of $N(i)$. We also have a transition from "add-one" to "match" labeled with $d + 1, d \to \epsilon$ for $d \in \{0, 1, 2, 3, 4, 5, 6, 8\}$, which will then transition to "match" upon matching the first non-9 digit with that digit plus one.

   From this point on, we just need to match digits one by one. So, we have transitions from "match" to "match" labeled with $d, d \to \epsilon$ for $d \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Finally, we have transitions from "match" to "accept" labeled with $\epsilon, \$ \to \epsilon$, from "add-one" to "accept" labeled with $1, \$ \to \epsilon$.

2. This grammar generates the language L consisting of all strings with equal numbers of a's and b's.

   We first prove that every x generated by the grammar is in L, by induction on the length of the derivation. For the base case, the only string that can be derived with a derivation of length 1 is ϵ, which is in L. Now consider a derivation $S \to aSb \to^* ayb$. By induction y is in L, and thus ayb is in L. A similar argument applies to each of the other two cases (i.e., the one in which the first step in the derivation is $S \to bSa$ and the one in which the first step in the derivation is $\to SS$).

   We now prove that every $x \in L$ can be generated by the grammar. This is done by induction on the length of x. Let $x = st$, where s is the shortest prefix of x that is in L. Following the

hint, we show that s cannot begin and end with the same character. To see this, define $f(i)$ to be the number of a's minus the number of b's in the prefix of x of length i. We have $f(0) = 0$, and $f(|s|) = 0$ (since $s \in L$). If s began and ended with a, then we would have $f(1) = 1$ and $f(|s| - 1) = -1$, which implies that there is some $1 < j < |s| - 1$ for which $f(j) = 0$, contradicting our assumption that s is the *shortest* prefix of x that lies in L. Similarly, if s began and ended with b, then we would have $f(1) = -1$ and $f(|s| - 1) = 1$, which implies that there is some $1 < j < |s| - 1$ for which $f(j) = 0$, contradicting our assumption that s is the shortest prefix of x that lies in L. We can therefore give a derivation of x that is either: $S \to SS \to aSbS \to^* st$ or $S \to SS \to bSaS \to^* st$, where the first S is used to derive s (which is possible by induction), and the second S is used to derive t (which again is possible by induction on the length of the string).

3. (a) Consider the two languages:

$$\begin{aligned} A &= \{a^m b^m c^n : m, n \geq 0\} \\ B &= \{a^m b^n c^n : m, n \geq 0\} \end{aligned}$$

They are context-free languages because we can construct a context free grammar for A and B, respectively:

$$\begin{array}{lcl} S \to TC \\ T \to aTb|\epsilon \\ C \to CC|c|\epsilon \end{array} \qquad \begin{array}{lcl} S \to AT \\ T \to bTc|\epsilon \\ A \to AA|a|\epsilon \end{array} \tag{2.1}$$

Equivalently, one can also construct NPDAs that recognize each language.

(b) It is easy to see that the intersection of A and B is:

$$C = A \cap B = \{a^n b^n c^n : n \geq 0\}$$

This is not a CFL, as proven in class using the Pumping Lemma.

4. (a) There are two cases to consider here. First, consider the case when $i = 0$. In this case, j, k, and l can take on any values. So chose any $uvxyz$ where v and/or y contain only one type of symbol, and pumping will maintain order creating strings that are still in the language. Therefore, the pumping property is satisfied. The other case is when $i \neq 0$ and $j = k = l$. In this case, chose v or y to contain only a's. Then pumping only increases the number of a's, not affecting the equality of j, k, and l creating strings still in the language, also satisfying the pumping property. Therefore, since the pumping property can be satisfied in all cases, the pumping lemma cannot be used to prove that L is not a CFL.

(b) Suppose L is a CFL that is represented in CNF. By the definition of Chomsky Normal Form, each nonterminal has either two nonterminals or a single terminal on the right hand side. Let N be the number of nonterminals.

Take a string w in L of length at least $p = 2^{N+1}$. We know that w has a parse tree of height at least $N + 2$ since each nonterminal node of the tree can have exactly two nonterminal children. Apply any marking of p or more symbols.

Now let's follow a path from the root of the parse tree to a marked descendant. Start at the root and always travel to the child with the greater number of marked descendants. A nonterminal whose two children both lead to marked descendants is called a branch node. Each time a branch node is passed, the number of marked symbols that are accessible by the path is reduced by at most one half (since we chose the child with *more* marked descendants). Since there are at least 2^{N+1} marked symbols, and each branch node reduces the reachable number by at most half, there are at least $N+1$ such branch nodes. As there are only N nonterminals, there must be a repeated nonterminal branch node on this path. If there are multiple nonterminals that are repeated, choose the instance that occurs latest in the path, and call this repeated nonterminal R. For convenience, call the first instance of the repeated nonterminal R_1 and the repeated instance R_2. (Remember they are still the *same* nonterminal, just labelled differently for convenience.)

String w can be divided into $uvxyz$ and pumped with respect to R just as was done in the original proof in class. The difference from the other proof is that we now must show that vy has at least one marked symbol, and that vxy has at most p marked symbols.

First, vy contains at least one marked symbol. Note how vxy is divided. x contains the symbols generated by R_2. v contains the symbols generated by the left-hand side of R_1 that are not in x, and y contains the symbols generated by the right-hand side of R_1 that are not in x. Remember that R_1 (and R_2) is a branch node, which means both of its children have marked descendants. Since R_2 cannot be both a right-hand side and left-hand side descendant of R_1 at the same time, v or y must have marked descendants by the definition of the branch node. So vy has at least one marked descendant.

Now we must show that vxy contains at most p marked positions of w. Note that R_1 exactly generates vxy. We have chosen R to be the lowest instance of repeating nonterminals, so it has no other repeating set of nonterminals below it. Therefore, since there are N nonterminals, and one is repeated exactly once, there are at most $N+1$ branch nodes from R_1 to the leaf. Since each branch node has at most twice as many marked descendants as the branch nodes below it, there are at most $2^{N+1} = p$ marked descendants in the subtree rooted at R_1 and hence in vxy.

So by this construction, the requirements of Ogden's Lemma are satisfied for a CFL.

(c) The language in part (a) can be proven to not be a CFL by applying Ogden's Lemma. One way to do this is to mark all of the b's in a valid string where $i \neq 0$. Let $w = ab^pc^pd^p$ (clearly w is in the language). We need to consider all partitions of w into $uvxyz$ where vy contains at least one marked symbol. If v or y contained two different symbols, then pumping would destroy the ordering of the string, and it would not be a valid string after pumping. So each of v and y may only contain a single type of symbol. Moreover, since the b's are the only marked characters, and vy must contain at least one marked symbol, at least one of v and y is contained in the b's. However, if the number of b'are increased, then the number of c's and d's must be increased at the same rate. But only one of them can be increased along with the b's since v and y can only contain one type of symbol. So the string cannot be partitioned into $uvxyz$ where uv^ixy^iz is in L. Therefore, L is not a CFL.

5. Let A and B be two CFL. Suppose CFLs are closed under complementation, then $\overline{\overline{A} \cup \overline{B}}$

must also be CFL, because CFLs are closed under union. However that expression is also equal to $A \cap B = \overline{\overline{A} \cup \overline{B}}$, but by Problem 3 CFLs are not closed under intersection.