CSE 103 Computational Models

Fall 2021 Prof. Fremont

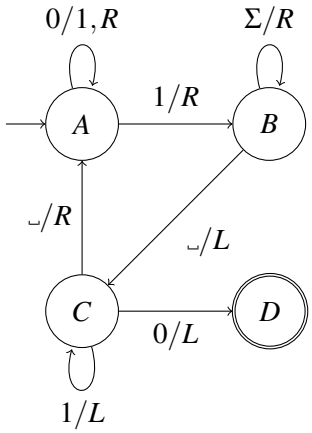
Practice 2

Final Practice Problems

This set of problems tries to cover most of the material we've seen in the second half of the class, so it is large. Feel free to jump around and find (sub)problems which you think will be most helpful. Each subproblem is a reasonable question to have on the final.

1. Working with a TM

Consider the following Turing machine M with $\Sigma = \{0,1\}$ (where we do not draw the reject state):



(a) List the computation history of M when run on input 0101.

Solution:

$A0101$

$1A101$

$11B01$

$110B1$

$1101B\sqcup$

$110C1$

$11C01$

$1D101$

(b) For each of the following input strings, state whether M accepts, rejects, or does not halt.

- ε
- 1
- 10

Solution:

- ε: This is rejected: in the start configuration, the symbol under the head is $\sqcup$, for which the graph above does not show a transition, so we move to the reject state.
- 1: M does not halt on this input: from the start state we read 1, move right, and go to B; then we read a blank, move left, and go to C; then read 1, move left, and stay in C; then read a blank, move right, and go to A. Now we are in the same state we started in, with the same tape contents, so the sequence above will repeat forever.
- 10: This is accepted: M reads the 1, goes to B, moves past the end of the input then left to the last symbol and state C; then it reads the 0 and accepts.

(c) What is the language of M?

Solution: From the examples above (and confirming with the graph), we can see that state A causes us to move past all 0s at the start of the input, converting them to 1s. When we see the first 1, we go to state B, which moves past the rest of the input, leaving it unchanged. Then we enter C on the last input symbol, and move left past all 1s; if we find a 0, we accept, whereas if we get back to the start of the input (and so read a blank) we return to state A. So if the input string has any 0 after the first 1, we will accept; otherwise, after leaving state A the tape will contain only 1s and M will run forever. So $L(M) = 0^*1\Sigma^*0\Sigma^*$.

2. Designing TMs

Design TMs deciding each of the following languages (not worrying about runtime). Give high-level verbal descriptions, like the solution to Problem 3 on HW 6.

(a) Binary strings with twice as many 0s as 1s.

Solution: We'll make successive passes through the input, crossing off two 0s every time we see a 1. Then if we eventually have no symbols left, there are exactly twice as many 0s as 1s.

(1) Scan the entire input to find a 1. If one is found, cross it off and continue to step (2). Otherwise, scan the input to find a 0: if there is one, reject, and otherwise accept.

(2) Scan the entire input to find a 0, then cross it off and find another 0; cross it off as well. If either 0 cannot be found, reject. Otherwise return to step (1).

(b) Binary strings of the form $x\overline{x}$, where $\overline{x}$ is x with all its bits flipped (e.g. $\overline{001} = 110$).

Solution: First, we'll find the middle of the string, where the transition from x to $\overline{x}$ should take place. Then we can use the usual back-and-forth traversal to compare the first and second halves of the string, except that we match 0s in the first half to 1s in the second half and vice versa.

(1) Mark the first symbol, then scan to the end of the input and mark the last symbol. Return to the first unmarked symbol and repeat, so that we mark two symbols on each pass. If we find there are no unmarked symbols left, then use a different mark to indicate the last symbol marked as the beginning of the second half. If there is only one unmarked symbol left, then the input has odd length, so we reject.

(2) Go through the symbols of the first half, crossing off each one in turn so that we don't process it again. For each symbol x, use states to remember whether it is a 0 or a 1, then proceed to the second half of the string and find the corresponding symbol y (i.e. the leftmost one which is not yet crossed off). If $x = y$, reject; otherwise cross off y and continue the iteration. Once every symbol of the first half is crossed off, accept.

(c) Strings over $\Sigma = \{0, 1, \$\}$ of the form $x\$y$ where $x, y \in \{0,1\}^*$ and x is a substring of y.

Solution: Our TM will have two stages: first, we'll check that the input has the form $x\$y$; second, we'll compare x against y starting from the first symbol of y, then the second symbol, etc., looking for a match at all possible positions.

(1) Scan from left to right looking for the \$ and then continuing on to the end of the input: if we get through the input without finding a \$, or find multiple \$s, reject. If x is empty (i.e. the \$ is the first symbol), we accept immediately since it is trivially a substring of y.

(2) Compare the symbols of x with those of y, marking the symbols we've already seen (e.g. replacing 0 with $\hat{0}$) and using states to remember the symbol of x we're currently comparing. If all symbols of x have counterparts in y, we accept. If we run off the end of y, then a match is not possible and we reject. Otherwise, x is not a prefix of y, but could appear as a substring later on in y.

(3) Cross off the first symbol of y (say using an X symbol) and unmark all other symbols. Repeat step (2), except skipping over all crossed-off symbols. This will compare x against y starting from the second position. Once that comparison is complete, we cross off the next symbol of y, and continue until we either accept or reject in step (2) (which must happen eventually, since once enough of y has been crossed off the remaining part will be shorter than x and we will reject).

3. Properties of reductions and decidable and RE languages

For each of the following statements, state whether it is true or false and give a 1-3 sentence justification.

(a) If $A \leq_m B$, then $\overline{A} \leq_m \overline{B}$.

Solution: This is true: if f is the mapping reduction from A to B, then $x \in A \iff f(x) \in B$, so $x \in \overline{A} \iff f(x) \in \overline{B}$. So f is also a mapping reduction from $\overline{A}$ to $\overline{B}$.

(b) Every language is either RE or co-RE.

Solution: This is false: there are languages which are neither RE nor co-RE, like EQ_{TM} (see Problem 5 below).

(c) A language L is decidable if and only if $L \leq_m \{010\}$.

Solution: This is true. If L is decidable, then the function

$$f(x) = \begin{cases} 010 & x \in L \\ 1 & \text{otherwise} \end{cases}$$

is computable, and gives a mapping reduction from L to $\{010\}$. Conversely, given such a mapping reduction f, we can decide whether $x \in L$ by computing $f(x)$ and checking whether the result is the string 010.

(d) If $L_0, L_1, L_2, \ldots$ is a sequence of RE languages, then $L = \cup_{i \geq 0} L_i$ is RE.

Solution: This is false: enumerate all the elements of $\overline{HALT}$ as $s_0, s_1, s_2, \ldots$. Then each language $L_i = \{s_i\}$ is RE (in fact decidable), since it contains only a single string. But $L = \cup_{i \geq 0} L_i = \overline{HALT}$ is not RE.

(e) If $A \leq_m B$, then $A \leq_m C$ for all $C \supseteq B$.

Solution: This is false: take any languages A and B over Σ where $A \leq_m B$ and there is some string $x \notin A$. Then $\Sigma^* \supseteq B$, but $A \not\leq_m \Sigma^*$, since any mapping reduction f would have to satisfy $f(x) \notin \Sigma^*$, which is impossible.

4. A closer look at the emptiness problem

Recall that $E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$.

(a) Prove that E_{TM} is co-RE.

Solution: We need to show that the complement,

$$\overline{E_{TM}} = \{x \mid x \text{ does not encode a TM, or it encodes a TM } M \text{ with } L(M) \neq \emptyset\},$$

is RE. We can build a recognizer R for $\overline{E_{TM}}$ as follows: given an input x, first check if it encodes a TM; if not, accept. Otherwise, x encodes some TM M with alphabet Σ, and we need to check whether its language is nonempty. Run M "in parallel" on all words in Σ^*, i.e. use the parallel simulation idea from class where we run simulations on N inputs for up to N steps, iteratively increasing N. If any of these simulations accepts, accept; otherwise continue searching.

If $x \in \overline{E_{TM}}$, then either x doesn't encode a TM, in which case we accept immediately, or x encodes a TM M whose language is nonempty, in which case we will eventually simulate M to completion on a word in $L(M)$ and accept. If instead $x \notin \overline{E_{TM}}$, then x encodes a TM M with $L(M) = \emptyset$, so none of our simulations will ever accept and so R will run forever. Therefore R accepts x if and only if $x \in \overline{E_{TM}}$, so R is a recognizer for $\overline{E_{TM}}$.

(b) Is E_{TM} Turing-recognizable? Why or why not?

Solution: It is not, since if it were recognizable it would be RE, and since we showed above that it is co-RE, it would then be decidable. But we proved in class that E_{TM} is undecidable.

(c) Is there a mapping reduction from A_{TM} to E_{TM}? If so, describe one; if not, argue why one cannot exist.

Solution: There is no such mapping reduction, even though there is a *Turing reduction* ($A_{TM} \leq_T E_{TM}$), as we showed in class (see Example 5.27 in the textbook for a discussion). If $A_{TM} \leq_m E_{TM}$, then by the argument in Question 3 (a) we would have $\overline{A_{TM}} \leq_m \overline{E_{TM}}$. Then since we showed above that E_{TM} is co-RE, $\overline{E_{TM}}$ is RE and applying the mapping reduction we would get that $\overline{A_{TM}}$ is also RE. This means that A_{TM} would be co-RE. However, we know that A_{TM} is RE, since it is the language of the universal TM, so then A_{TM} would be both RE and co-RE and therefore decidable. This is a contradiction, since we know A_{TM} is undecidable, so there cannot be a mapping reduction from A_{TM} to E_{TM}.

5. The equivalence problem for Turing machines

The equivalence problem for TMs is $EQ_{TM} = \{\langle M,N\rangle \mid M \text{ and } N \text{ are TMs and } L(M) = L(N)\}$.

(a) Prove that EQ_{TM} is not RE.

Solution: We'll prove that $\overline{HALT} \leq_m EQ_{TM}$: since $\overline{HALT}$ is not RE, this will show that neither is EQ_{TM}. If $\langle M,x\rangle \in \overline{HALT}$, then M runs forever on input x. We can design a TM N that accepts nothing in this case, otherwise accepting everything. Specifically, N does the following on input y:

(1) Run M on input x.

(2) Accept.

If $\langle M,x\rangle \in \overline{HALT}$, then N gets stuck on step (1) and so accepts no strings. If instead $\langle M,x\rangle \notin \overline{HALT}$, then step (1) finishes and N accepts all strings. Let $f(\langle M,x\rangle) = \langle N,A\rangle$ where A is a trivial TM rejecting all inputs. This function is a mapping reduction from $\overline{HALT}$ to EQ_{TM}, since $L(N) = L(A)$ if and only if $\langle M,x\rangle \in \overline{HALT}$.

(b) Prove that EQ_{TM} is not co-RE.

Solution: We'll follow the same approach as above to build a mapping reduction from $\overline{HALT}$ to $\overline{EQ_{TM}}$, showing that $\overline{EQ_{TM}}$ is not RE and so that EQ_{TM} is not co-RE. Given an instance $\langle M,x\rangle$ of $\overline{HALT}$, build a TM N in exactly the same way as above; as we showed there, N accepts all strings if and only if $\langle M,x\rangle \notin \overline{HALT}$. Let $f(\langle M,x\rangle) = \langle N,B\rangle$ where B is a trivial TM accepting all inputs. This function is a mapping reduction from $\overline{HALT}$ to $\overline{EQ_{TM}}$, since $L(N) \neq L(B)$ if and only if $\langle M,x\rangle \in \overline{HALT}$.

6. Decidable problems

Argue why each of the following problems is decidable.

(a) $COUNT_{DFA} = \{\langle D, n\rangle \mid D \text{ is a DFA and } |L(D)| = n\}$

Solution: To count the number of words in $L(D)$, we'll start by doing a graph traversal of D to find all the reachable states, deleting any which are not reachable. We can similarly prune away all states from which an accepting state is not reachable. If the start state is pruned away, then $L(D) = \emptyset$, so we'll accept only if $n = 0$. If the part of D remaining after pruning contains a cycle, then $L(D)$ is infinite (like in the pumping lemma: we can repeat the cycle as many times as desired to produce more strings in the language), so we'll reject. Otherwise, the graph is acyclic, so there are finitely-many paths from the start state to an accepting state. We can just enumerate all such paths, count them, and accept if there are exactly n.

(*Note:* In fact, you can even implement this algorithm in polynomial time, by using dynamic programming to count the paths without explicitly enumerating them. This is basically the same as the classical algorithm for counting paths in a DAG, if you've seen that before.)

(b) $L = \{\langle R\rangle \mid R \text{ is a regular expression matching only words of even length}\}$

Solution: We can decide L by first converting R to an equivalent DFA D (the conversion procedure we described in class can clearly be implemented by an algorithm), intersecting it with a DFA O that accepts all words of *odd* length, and applying the algorithm for E_{DFA}: if the intersection is empty, then R matches no words of odd length, so we accept; otherwise, we reject.

7. Undecidable problems

Prove that each of the following problems is undecidable.

(a) $L = \{\langle M \rangle \mid M \text{ is a TM that on any input either runs forever or halts in polynomial time}\}$

Solution: We will reduce $HALT$ to L. Given an instance $\langle M, x \rangle$ of $HALT$, let the TM N do the following on input y:

(1) Run M on input x.

(2) Run for $2^{|y|}$ steps.

If M halts on x, then N halts on every input y, but its runtime is at least $2^{|y|}$ and not polynomial, so $\langle N \rangle \notin L$. If instead M does not halt on x, then N runs forever on all inputs, so $\langle N \rangle \in L$. So we can decide whether $\langle M, x \rangle \in HALT$ by constructing N, asking the oracle whether $\langle N \rangle \in L$, and accepting if not. This shows that $HALT \leq_T L$, and since $HALT$ is undecidable, so is L.

(b) $L = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = SAT\}$

Solution: We will reduce $HALT$ to L. Given an instance $\langle M, x \rangle$ of $HALT$, let the TM N do the following on input y:

(1) Run M on input x.

(2) If $y \in SAT$, accept; otherwise reject.

If M halts on x, then N accepts y iff $y \in SAT$, so $L(N) = SAT$. If instead M does not halt on x, then N runs forever on every input, so $L(N) = \emptyset \neq SAT$. So the function $f(\langle M, x \rangle) = \langle N \rangle$ gives a mapping reduction from $HALT$ to L. Since $HALT$ is undecidable, so is L.

8. Polynomial-time decidable problems

For each of the following problems, argue why it is in P.

(a) $LONG_{DFA} = \{\langle D, 1^n\rangle \mid D \text{ is a DFA which only accepts words of length} \geq n\}$

Solution: If Σ is the alphabet of D, we can construct a DFA S such that $L(S) = \Sigma^{<n}$: we use a chain of n states to count how many symbols have been seen so far, and once we reach n we go into a rejecting sink state. Then to check if D accepts any words of length $< n$, we can use the product construction to intersect D and S and then apply the emptiness algorithm for DFAs. If $L(D) \cap L(S) = L(D) \cap \Sigma^{<n}$ is empty, then $\langle D, 1^n\rangle \in LONG_{DFA}$ and we accept; otherwise we reject. This procedure takes polynomial time, since S has size $O(n)$ and the product and emptiness algorithms take polynomial time. (In fact, this shows that $LONG_{DFA} \leq_P E_{DFA}$.)

(b) $FUNNY\text{-}CLIQUE = \{\langle G, k, 1^n\rangle \mid G = (V, E) \text{ is an undirected graph with a } k\text{-clique and } n = |V|^k\}$

(Notice the difference from *CLIQUE*, which we've proved is *NP*-complete.)

Solution: A k-clique is a set of k vertices. Since G has $|V|$ vertices, there are $\binom{|V|}{k} \leq |V|^k = n$ such sets; for each one, testing whether it is a k-clique can be done in polynomial time (we just look to see if G contains all the required edges). So checking all possible k-cliques can be done in time polynomial in $|G|$ and n, and thus polynomial in $|\langle G, k, 1^n\rangle|$.

(The point here is that the longer your input, the longer your algorithm can run while still being considered polynomial-time; here, the extra padding at the end of the input is exponentially long, allowing us to solve the inner instance of *CLIQUE* using an algorithm that takes time exponential in k while still being polynomial in the length of the entire input.)

9. NP problems

Prove that the following problems are in NP.

(a) $COMPOSITE = \{\langle n \rangle \mid n \text{ is a composite (non-prime) integer}\}$

Solution: If n is composite, then $n = ab$ for some integers $a, b < n$, and since we can do multiplication in polynomial time, we'll use the pair $\langle a, b \rangle$ as our certificate. Given the input $\langle n, y \rangle$, our verifier will check that y encodes a pair (a, b) of integers, that $a, b < n$, and that $ab = n$, accepting if so. Then if n is composite, there is a value of the certificate causing the verifier to accept; otherwise, $n \neq ab$ for any $a, b < n$ and so the verifier will always reject. Since the verifier runs in polynomial time, this shows that *COMPOSITE* is in NP.

(*Note:* This problem is actually also in P, but that was only proved in 2002!)

(b) $MAX\text{-}CUT = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a cut of size} \geq k\}$ (where a *cut* is a partition of the vertices of G into two groups; its *size* is the number of edges between the groups.)

Solution: The certificate is the cut. Given the input $\langle G, k, y \rangle$, with $G = (V, E)$ an undirected graph, our verifier will first check that y encodes a cut, i.e. a pair (A, B) where $A, B \subseteq V$ and $A \cap B = \emptyset$. If so, and the number of edges $(u, v) \in E$ with $u \in A$ and $v \in B$ is at least k (which we can check in polynomial time by iterating over all the edges of G), the verifier accepts; otherwise it rejects. If G has a cut (A, B) of size $\geq k$, then the verifier will accept with certificate $y = \langle A, B \rangle$; otherwise, there is no cut of size $\geq k$ and so the verifier will reject for any value of y. Since the verifier runs in polynomial time, this shows that *MAX-CUT* is in NP.

(*Note:* This problem is actually also NP-complete, although the reduction to prove NP-hardness is a bit challenging. See Problem 7.27 in the textbook if you'd like to try it.)

10. NP-complete (or not) problems

For each of the following problems, state whether it is NP-complete or not under the assumption that $\mathsf{P} \neq \mathsf{NP}$ (since otherwise all nontrivial NP problems are NP-complete), and give a 1-3 sentence justification.

(a) $ALL_{DFA} = \{\langle D \rangle \mid D \text{ is a DFA with alphabet } \Sigma \text{ and } L(D) = \Sigma^*\}$

Solution: This is not NP-complete (if $\mathsf{P} \neq \mathsf{NP}$), since it is in P: to check whether a DFA accepts all strings we can complement it and apply the polynomial-time algorithm for E_{DFA}.

(b) *AT-MOST-3SAT* $= \{\langle \varphi \rangle \mid \varphi \text{ is a Boolean formula in CNF with } \leq 3 \text{ literals in each clause}\}$

Solution: This is NP-complete: we can reduce *3SAT* to it trivially, since every instance of *3SAT* is also an instance of *AT-MOST-3SAT*, so it is NP-hard. It is also in NP, being a special case of SAT.

(c) *5-CLIQUE* $= \{\langle G \rangle \mid G \text{ is an undirected graph with a 5-clique}\}$

Solution: This is not NP-complete (if $\mathsf{P} \neq \mathsf{NP}$), since it is in P: we can go through all sets of 5 vertices and check whether they form a clique. If G has v vertices, there are $\binom{v}{5} = O(v^5)$ such sets, so this procedure will take polynomial-time (noting that v is at most linear in the size of the input).

(d) $A_{TM} = \{\langle M, x \rangle \mid M \text{ is a TM that accepts the string } x\}$

Solution: This is not NP-complete, since it is undecidable and so not in NP.

(e) *BOUNDED-*$A_{TM} = \{\langle M, x, 1^n \rangle \mid M \text{ is a TM that accepts the string } x \text{ in at most } n \text{ steps}\}$

Solution: This is not NP-complete (if $\mathsf{P} \neq \mathsf{NP}$), since it is in P: we can simulate M on input x for n steps and see if it accepts within that time. The simulation takes time polynomial in n and the sizes of M and x, which is polynomial in the length of the input $\langle M, x, 1^n \rangle$.

(f) *BOUNDED-*$A_{NTM} = \{\langle M, x, 1^n \rangle \mid M \text{ is an NTM that accepts } x \text{ in at most } n \text{ steps along some branch}\}$ (where an NTM is a *nondeterministic* Turing machine)

(*Hint:* Use the fact that every NP problem has an NTM deciding it in polynomial time.)

Solution: This is NP-complete. It is in NP, since there is a polynomial-time verifier: the certificate is the sequence of $\leq n$ nondeterministic choices specifying the branch of M to simulate (alternatively, we can give an NTM deciding this language in polynomial time: it is a nondeterministic version of the universal TM which simulates M for n steps). It is also NP-hard: for any $A \in \mathsf{NP}$, there is an NTM M deciding A in time at most $p(n)$ for some polynomial p, and the function $f(x) = \langle M, x, 1^{p(|x|)} \rangle$ gives a polynomial-time reduction from A to *BOUNDED-*A_{NTM}.

11. NP-hardness proofs

Prove each of the following problems is NP-hard by reducing a problem we already know is NP-hard to it.

(a) $L = \{\langle G, H\rangle \mid G$ and H are undirected graphs, and some subgraph of G is isomorphic to $H\}$
(i.e. G contains a copy of H, up to relabeling vertices; this is called the *subgraph isomorphism problem*)

Solution: We'll reduce *CLIQUE* to L. Given an instance $\langle G, k\rangle$ of *CLIQUE*, let H be the complete graph with k vertices (i.e. the graph with k vertices all connected to each other). A k-clique in G is isomorphic to H, so $\langle G, k\rangle \in$ *CLIQUE* $\iff \langle G, H\rangle \in L$. We can build H in polynomial time, since k is at most the number of vertices of G, so the function $f(\langle G, k\rangle) = \langle G, H\rangle$ is a polynomial-time reduction from *CLIQUE* to L.

(b) *COUNT-SAT* $= \{\langle \varphi, k\rangle \mid \varphi$ is a Boolean formula with at most k satisfying assignments$\}$
(Side note: this problem is thought to *not* be in NP (or co-NP), but in a higher complexity class. It's usually viewed as a counting problem called *#SAT* rather than a decision problem.)

Solution: We'll reduce *SAT* to *COUNT-SAT*. Given a *SAT* instance $\langle \varphi \rangle$, consider the negated formula $\overline{\varphi}$ (that is, the formula ϕ with an extra NOT operator applied). Notice that a satisfying assignment of φ does *not* satisfy $\overline{\varphi}$ and vice versa. So if φ has a satisfying assignment, then $\overline{\varphi}$ has at least one non-satisfying assignment, and so at most $2^n - 1$ satisfying assignments if φ has n variables (since there are 2^n assignments total). On the other hand, if φ is unsatisfiable, then every assignment satisfies $\overline{\varphi}$, so $\overline{\varphi}$ has 2^n satisfying assignments. Therefore φ is satisfiable if and only if $\langle \overline{\varphi}, 2^n - 1\rangle \in$ *COUNT-SAT*. Since we can compute $\overline{\varphi}$ and $2^n - 1$ in polynomial time, the function $f(\langle \varphi \rangle) = \langle \overline{\varphi}, 2^n - 1\rangle$ gives a polynomial-time reduction from *SAT* to *COUNT-SAT*.

12. Pumping lemma for context-free languages

Prove that the following languages are not context-free:

(a) $L = \{a^i b^j a^i b^j \mid i, j \geq 1\}$

Solution: Given $p \geq 1$, we'll pick the word $w = a^p b^p a^p b^p$, which satisfies $w \in L$ and $|w| = 4p \geq p$. Now for any decomposition $w = uvxyz$ with $|vy| > 0$ and $|vxy| \leq p$, since the blocks of each letter have length p, the string vxy can only overlap at most two consecutive blocks. In particular, it is not possible for vxy to overlap both blocks of as or both blocks of bs. Furthermore, since $|vy| > 0$, the string vy contains either an a or a b (or both). So setting $k = 2$, the string $uv^k xy^k z$ has more as or bs than w. If the new symbols cause the string to no longer have the required 4-block form, then it is not in L. Otherwise, the length of at least one of the blocks has increased, but the length of the corresponding block with the same letter has not changed (since vxy can't overlap both blocks), so again the string is not in L. Therefore in all cases $uv^2xy^2z \notin L$, and so by the contrapositive of the pumping lemma L is not context-free.

(b) $L = \{0^i 1^i 2^j \mid i, j \geq 0 \text{ and } i \leq j\}$

Solution: Given $p \geq 1$, we'll pick the word $w = 0^p 1^p 2^p$, which satisfies $w \in L$ and $|w| = 3p \geq p$. Now for any decomposition $w = uvxyz$ with $|vy| > 0$ and $|vxy| \leq p$, since the blocks of each digit have length p, the string vxy cannot contain all three digits. So if vy contains a 0, then it cannot contain a 2, and setting $k = 2$ yields a string $uv^k xy^k z$ with strictly more 0s but the same number of 2s as w. Therefore $uv^k xy^k z$ has strictly more 0s than 2s (since there were equal numbers of these in w) and is not in L. If instead vy does *not* contain a 0, then it must contain either a 1 or a 2 (or both) since $|vy| > 0$. Then setting $k = 0$, the string $uv^k xy^k z = uxz$ has strictly fewer 1s or 2s than w, but the same number of 0s: so as above it has strictly fewer 1s or 2s than 0s, and again is not in L. Therefore in all cases there is a $k \geq 0$ such that $uv^k xy^k z \notin L$, and so by the contrapositive of the pumping lemma L is not context-free.