Homework 4

COMS W3261, Summer A 2023

This homework is due **Tuesday, 6/20/2023, at 11:59pm EST**. Submit to GradeScope (course code: K3VK75). If you use late days, the absolute latest we can accept a submission is Friday at 11:59 PM EST.

Grading policy reminder: LaTeX is preferred, but neatly typed or handwritten solutions are acceptable.[1] Feel free to use the .tex file for the homework as a template to write up your answers, or use the template posted on the course website. Your TAs may dock points for indecipherable writing.

Proofs should be complete; that is, include enough information that a reader can clearly tell that the argument is rigorous.

If a question is ambiguous, please state your assumptions. This way, we can give you credit for correct work. (Even better, post on Ed so that we can resolve the ambiguity.)

LaTeX resources.

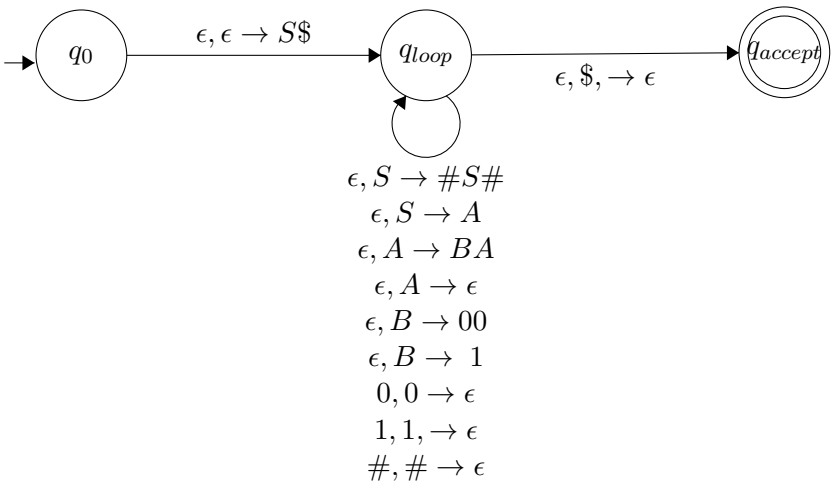
- Detexify is a nice tool that lets you draw a symbol and returns the LaTeX codes for similar symbols.
- The tool Table Generator makes building tables in LaTeX much easier.
- The tool Finite State Machine Designer may be useful for drawing automata. See also this example (PDF) (.tex) of how to make fancy edges (courtesy of Eumin Hong).
- The website mathcha.io allows you to draw diagrams and convert them to LaTeX code.
- To use the previous drawing tools (and for most drawing in LaTeX), you'll need to use the package Tikz (add the command "\usepackage{tikz}" to the preamble of your .tex file to import the package).
- This tutorial is a helpful guide to positioning figures.

[1] The website Overleaf (essentially Google Docs for LaTeX) may make compiling and organizing your .tex files easier. Here's a quick tutorial.

Problem 1 (12 points)

The following questions concern the pushdown automaton P pictured below. P's tape alphabet is $\Sigma = \{0, 1, \#\}$, and P's stack alphabet is $\Gamma = \{0, 1, \#, S, A, B, \$\}$.

In the state diagram below, we have employed the same shorthand as we used in class when converting CFGs to PDAs: if a transition indicates to push the string "abc", for instance, this is short for pushing c, then b, and then a, so that a is on the top of the stack.



1. (3 points.) Which of the following six strings are accepted by this PDA? ϵ, ##, #111#, 001100, #0100#, ###1001###.

   Every string except for #0100# is accepted. (See language description, below.)

2. (5 points.) P was constructed by starting with a context-free grammar, and following the process sketched in class to turn a CFG into an equivalent PDA. Construct a grammar G that derives the same language that P recognizes. (You may wish to reverse-engineer the PDA construction process. However, any equivalent grammar is OK).

   No explanation required for this question; however, if you're uncertain about part of the grammar please feel free to include an explanation to ensure partial credit.

   We can read the rules of our CFG off the central state, q_{loop}. We get the following grammar:

   $$S \to \#S\# \mid A$$
   $$A \to BA \mid \epsilon$$
   $$B \to 00 \mid 1$$

3. (4 points). What is the language recognized by P (and derived by your grammar)? If you like, you may state your answer using set notation and/or regular expressions. Justify your

answer by reference to P or by reference to your newly created grammar from the previous part.

P recognizes the language $\{\#^k(00 \cup 1)^*\#^k \mid k \geq 0\}$.

To see this, first observe that any derivation starts by invoking the rule $S \to \#S\#$ a total of k times for some $k \geq 0$, which generates two substrings of # characters on either side of the variable A.

Next, consider the rules applied to the variable A. To reach a terminal string, we must use the rule $A \to BA$ a total of j times for some $j \geq 0$, after which we eliminate the variable A by using the rule $A \to \epsilon$.

Our string now has the form $\#^k B^j \#^k$ for some $j, k \geq 0$. Each B variable is then replaced with 00 or 1, so the middle substring of 0's and 1's matches the regular expression $(00 \cup 1)^*$.

Rationale: The goal of this problem is to practice evaluating PDAs, determining which language they recognize, and relating between PDAs and CFGs.

References: Sipser p.112 for an introduction to PDAs, p.113 for the formal definition and pp.114-116 for examples. For converting CFGs to PDAs, see Lemma 2.21 on page 117 and the Proof Idea/Proof sections that follow. See also Lightning Review 6 on Pushdown Automata.

Problem 2 (8 points)

Use the context-free pumping lemma to prove that the following language is not context-free. [Hint: the CFPL has the form "In any context-free language, all sufficiently long strings can be divided into substrings in a way that satisfies three properties". To show that a language is not context-free, we need to contradict this statement; i.e., show that "In this particular language, there exists at least one long string such that *no* division of this string into substrings satisfies all three properties."]

1. (8 points).

$$L_1 = \{q\#r\#q^R \mid q, r \in \{0,1\}^*, \text{ and } |q| \geq |r|. \ \}.$$

That is, q and r are binary strings, but L_1 is over the alphabet $\{0, 1, \#\}$. Recall that q^R denotes the reverse of q: the third section of the string should be the same as the first section, but backwards.

Assume for contradiction that L_1 is context-free. Then L_1 satisfies the context-free pumping lemma: there exists a pumping length p such that for every string $s \in L_1$ with $|s| \geq p$, we can divide s into five pieces $s = uvxyz$ such that (1) $|vy| > 0$, (2) $|vxy| \leq p$, and (3) $uv^ixy^iz \in L_1$ for all $i \geq 0$.

Consider the string $s = 0^p1^p\#0^{2p}\#1^p0^p$. Note $s \in L_1$ and $|s| \geq p$. We proceed to show that every possible division of s into five substrings fails the conditions. We consider three cases:

(a) Either v or y contains a $\#$. In this case, the string $uvvxyyz$ contains at least three $\#$ characters and is not in the language, so we cannot satisfy (3).

(b) Either v or y contains a character that is part of the first 0^p1^p substring or the last 1^p0^p substring. Note that, if we are to satisfy (2), it is not possible that vxy contains characters from *both* of these two substrings, as there are $2p + 2$ characters in between. Thus $uvvxyyz$ increases the length of either q or q^R, which means they are no longer the reverse of each other, and we cannot satisfy (3).

(c) The remaining case is that v and y contain no characters from the first $0^p1^p\#$ substring or the last $\#1^p0^p$ substring. Thus vxy is a substring of the middle 0^{2p} substring. In this case, $uvvxyyz$ increases the length of the middle substring of zeroes, which implies $|r| > |q|$ and violates (3) if we satisfy (1) and $|vy| > 0$.

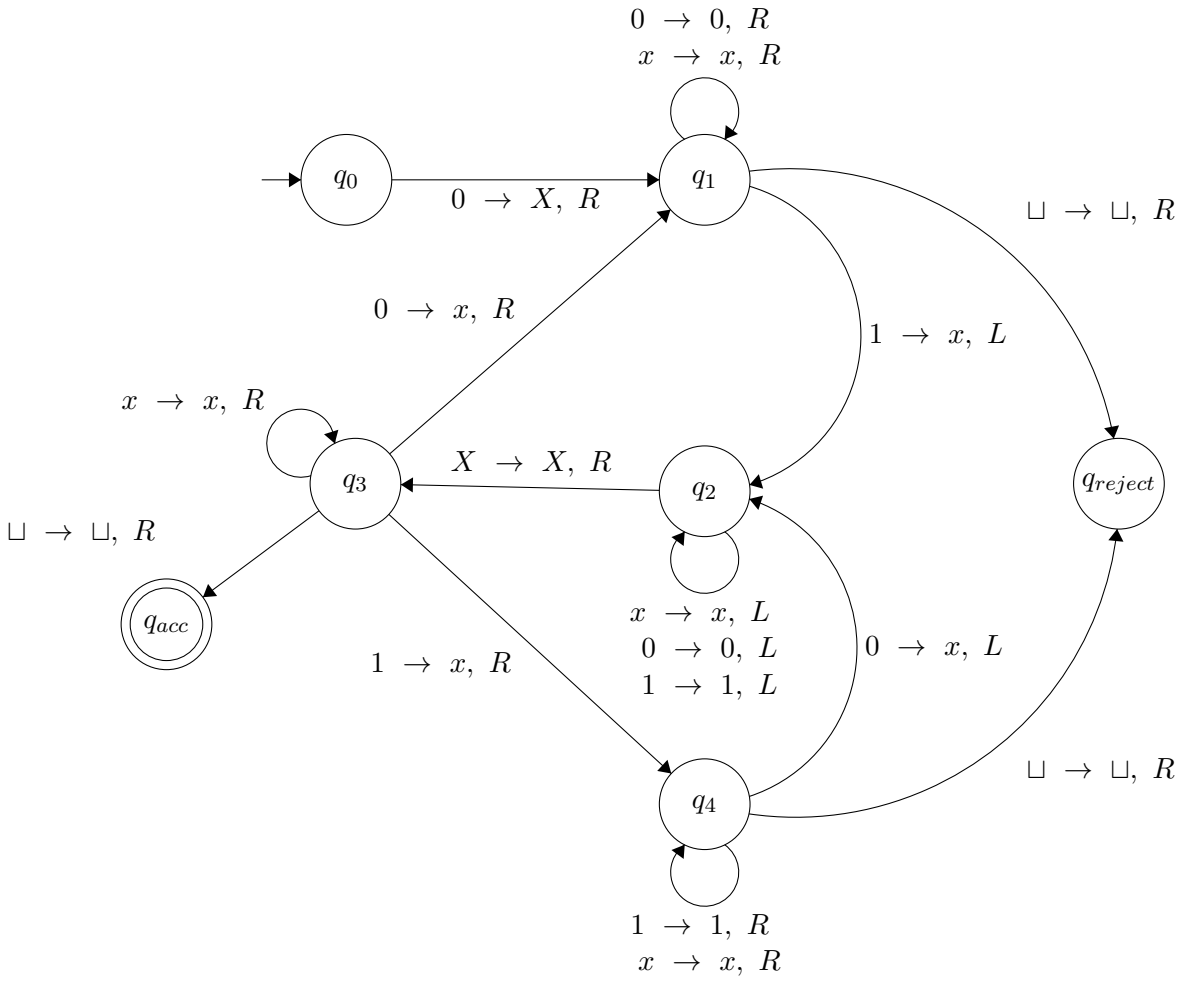
Thus there is no way to divide s into 5 substrings in a way that meets the conditions of the pumping lemma. L_1 fails the CFPL and is thus non-context-free.

Rationale: The goal of this question is to practice using the context-free pumping lemma.

References: The CFPL is stated in Sipser p.125, with examples on pp. 128-129. Also see the video Example 5: Using the Context-Free Pumping Lemma on the resources section of the course webpage.

Problem 3 (8 points)

For the following question, refer to the Turing Machine state diagram T drawn below. The input alphabet of T is $\Sigma = \{0, 1\}$, and the tape alphabet of T is $\Gamma = \{\sqcup, 0, 1, x, X\}$ (here $\sqcup$ indicates a blank tape square). Certain transitions have been omitted for tidiness: if a state does not have a transition on a certain symbol, you may assume that the missing transition goes to q_{reject}.



1. For each of the following strings, list the sequence of states that occur when T runs on this string, and the contents of the tape when computation ends at q_{accept} or q_{reject}. (For example, on the string 00, the TM passes through the sequence $q_0, q_1, q_1, q_{reject}$, and leaves the string $X0$ on the tape.) Hint: a careful way to keep track of TM computation is to keep track of the sequence of tape strings, annotated with the current state and the position of the tape head.

   (a) (1 point). The string 000.
   We pass through the states $q_0, q_1, q_1, q_1, q_{reject}$ and leave $X00$ on the tape.

   (b) (1 point). The string ϵ.
   We pass through the states q_0, q_{reject} and leave an empty tape.

(c) (1 point). The string 0111.

We pass through the states $q_0, q_1, q_2, q_3, q_3, q_4, q_4, q_{reject}$, and leave $Xxx1$ on the tape.

(d) (1 point). The string 01.

We pass through the states $q_0, q_1, q_2, q_3, q_3, q_{acc}$, and leave Xx on the tape.

(e) (1 point). The string 0110.

We pass through the states $q_0, q_1, q_2, q_3, q_3, q_4, q_2, q_2, q_2, q_3, q_3, q_3, q_3, q_{accept}$, and leave $Xxxx$ on the tape.

(f) (1 bonus point). The string 00111100.

We pass through the states $q_0, q_1, q_1, q_2, q_2, q_3, q_1, q_1, q_2, q_2, q_2, q_3, q_3, q_3, q_3$; then $q_4, q_4, q_2(\times 6), q_3(\times 5)$; then $q_4, q_4, q_2(\times 7), q_3(\times 8), q_{acc}$, and leave $Xxxxxxxx$ on the tape.

2. (3 points). What language does T decide? No justification necessary.

T decides the language of strings that have an equal number of 0's and 1's and start with an 0.

Rationale: The goal of this question is to practice reading TM state diagrams and tracking TM computation.

References: Sipser pp. 167-169 (formal definition of a Turing Machine) and pp. 171-173 (Turing Machine state diagrams). See also Lightning Review 7: Turing Machines in the resource section of the course webpage.

Problem 4 (6 points)

Provide *high-level descriptions* of Turing Machines that *decide* the following languages. A high-level description is an algorithm for a Turing Machine described in prose, ignoring implementation details such as the way information is encoded or where the head needs to move. However, your TM's behavior should still be *completely specified*: it should be clear what the Turing Machine does in every case.

For example: we can build a Turing Machine M that decides the language

$$\{\langle G\rangle \mid \langle G\rangle \text{ encodes a } \textit{connected graph.}\}$$

. Our Turing Machine will implement a breadth-first search on the encoded input as follows:

1. First, check to see if the input encodes a graph and reject if not.
2. Mark the first vertex v in the graph.
3. Review the encoded graph and mark every vertex adjacent to a marked vertex. Repeat this step until no new nodes are marked.
4. Accept if all vertices are marked and reject otherwise.

(Recall that a decider must halt and accept strings in the language, and must halt and reject on strings not in the language.)

1. (2 points.) A_1 = the language containing every finite set of integers S such that for every triple a, b, c drawn from S without replacement, $a + b \neq c$.

   Our TM to recognize A_1 will operate as follows:

   (a) First, scan the input and ensure that the input encodes a set of integers; reject if not. (This step is implicit; including it in a high-level description is optional.)

   (b) Loop through all triples (a, b, c) and reject if $a + b = c$. We can do this by the equivalent of a triply nested for loop; e.g., 'for all a, check that for all b, for all c it holds that $a + b \neq c$...'.

   (c) Accept if $a + b \neq c$ for all triples.

2. (4 points.) A_2 = the language containing every encoded DFA $\langle D\rangle$ such that D accepts at least one string s. (Here, our Turing Machine receives the description of a DFA, encoded in some alphabet and written on its tape. The TM has access to any information that would normally be part of the DFA 5-tuple or state diagram.)

   Our TM to recognize A_2 will operate as follows:

   (a) First, we reject any input that does not encode a DFA. (This step is implicit; including it in our high-level description is optional.)

   (b) Perform a breadth-first search from the start state of the DFA to enumerate all reachable states, as in our previous TM that decided whether a graph is connected.

(c) Accept if the set of reachable states contains an accept state, and reject if not.

Another way to do this would be to simulate the DFA and check every string of length at most $|Q| - 1$, where Q is the state set of D: since every accepting string longer than this contains a loop, if D accepts at least one string there must be an accepting string of length at most $|Q| - 1$.

Note that trying every string, perhaps in order from short to long, won't work: if our DFA rejects every string, then our TM will never halt.

Rationale: The goal of this question is to practice thinking about Turing Machines as general-purpose computers that can execute algorithms for solving complex decision problems.

References: Sipser pp. 186-187 contains the high-level description given as an example, as well as a breakdown from the high level to the implementation level. TM descriptions later in the book (for instance, those on pp. 195-196) are also given at the high-level.

Problem 5 (1 bonus point): Origins of the Turing Machine

Alan Turing first described the Turing Machine in his 1936 paper "On Computable Numbers, with Applications to the Entscheidungsproblem". The TM wasn't even the main focus of the paper: instead, it was just a tool used to formalize what computational processes can accomplish, used to show that a particular sort of mathematical statement *can't* be proved by a machine.

Among other things, Turing was a decent writer (at least, compared to some mathematicians). Read Sections 1 and 2 of 'On Computable Numbers' and consider Turing's original definition of the TM. You can find the paper here: https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf. (If you'd like a lighter Turing read, you might want to read 'Computing Machinery and Intelligence', the paper that introduced the Turing Test: https://redirect.cs.umbc.edu/courses/471/papers/turing.pdf.)

Turing's original machine has an additional function that makes it more similar to one of the three 'Variant TMs' (multitape TMs, NTMs, and enumerators) listed in Sipser Section 3.2 than to our definition of a TM. Which one of these three is the most similar?

The enumerator - like Turing's *a*-machine, it can print to the tape.