CS 181 PRACTICE FINAL A

You may state without proof any fact taught in lecture.

1 Construct one-tape, deterministic Turing machines for the tasks below. You must provide both a state-transition diagram and a detailed verbal description:

a. a Turing machine that decides the language $\{0^n 1 0^n : n \geqslant 0\}$;

b. a Turing machine that computes the function sort: $\{0,1\}^* \to \{0,1\}^*$, which rearranges the bits of the input string in sorted order. For example, sort$(1010) = 0011$.

2 Give an algorithm that takes as input a PDA P and a symbol γ of its stack alphabet, and decides whether P ever pops γ from the stack in any computation.

3 For a nonempty language L, the *longest common prefix* of L is denoted $\mathrm{lcp}(L)$ and defined as the longest string w such that all strings in L start with w. For example, $\mathrm{lcp}(\{0111, 0101\}) = 01$ and $\mathrm{lcp}(0^+ \cup 1^+) = \varepsilon$.

a. For every nonempty language L, prove that $\mathrm{lcp}(L)$ exists and is unique.

b. Give an algorithm that inputs a PDA P with $L(P) \neq \varnothing$ and computes $\mathrm{lcp}(L(P))$.

4 For a language L, a *motif* of L is any string that occurs as a substring in infinitely many strings of L. Prove that the following problems are decidable:

a. on input a DFA D and a string w, decide whether w is a motif of $L(D)$;

b. on input a DFA D, compute the number of motifs of $L(D)$ (an integer or "∞").

5 For each problem below, determine whether it is decidable and prove your answer:

- **a.** on input a Turing machine M, decide whether $L(M)$ is regular;
- **b.** on input a Turing machine M that recognizes a regular language, compute a regular expression for $L(M)$;
- **c.** on input a Turing machine M, decide whether M accepts every input in at most 2019 steps.

SOLUTIONS

CS 181 PRACTICE FINAL A

You may state without proof any fact taught in lecture.

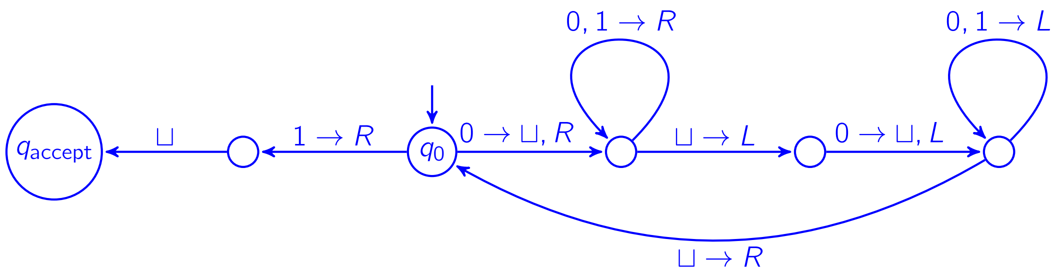
1 Construct one-tape, deterministic Turing machines for the tasks below. You must provide both a state-transition diagram and a detailed verbal description:

a. a Turing machine that decides the language $\{0^n10^n : n \geqslant 0\}$;

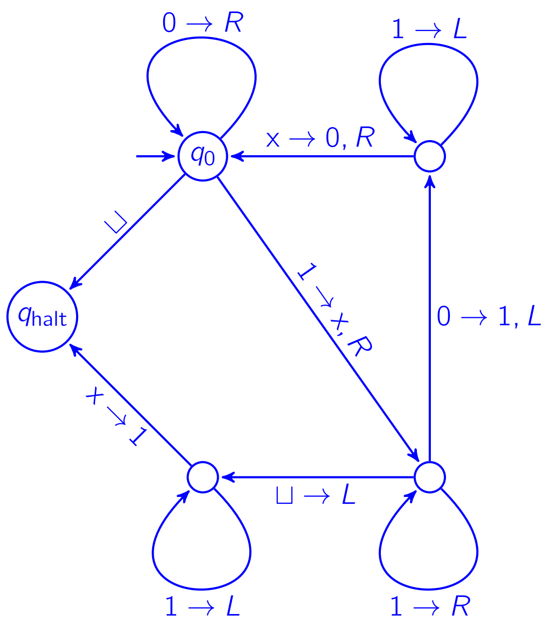
b. a Turing machine that computes the function $\mathrm{sort}\colon \{0,1\}^* \to \{0,1\}^*$, which rearranges the bits of the input string in sorted order. For example, $\mathrm{sort}(1010) = 0011$.

Solution.

a. The TM shuttles back and forth on the input, alternately erasing a 0 from the left end and a 0 from the right end. At all times, the blank cells serve to delimit the remaining part of the string. When the string shrinks to "1," the TM accepts it. The state-transition diagram is as follows.



b. The TM starts by scanning the input string looking for a 1. If the string contains only 0s, then it is already sorted, and the TM halts. Otherwise, the TM overwrites the first 1 with an x to mark it as a candidate for a swap, and scans from then on for a 0. If it finds no 0s, then the string is sorted, so the TM returns to the x, changes it back to a 1, and halts. If the TM does find a 0, then it changes it to a 1, returns to the x, changes it to a 0, and recursively sorts the remainder of the string from that point on. The state-transition diagram is below.



2 Give an algorithm that takes as input a PDA P and a symbol γ of its stack alphabet, and decides whether P ever pops γ from the stack in any computation.

Solution. Let $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be given. Construct a PDA P' that operates just like P but accepts if and only if γ has been popped from the stack at some point. This can be achieved by "remembering" in state whether γ has been popped. Formally, $P' = (\{\text{yes, no}\}\times Q, \Sigma, \Gamma, \Delta, (\text{no}, q_0), \{\text{yes}\}\times Q)$, where

$$\Delta((b, q), \sigma, \tau) = \begin{cases} \{\text{yes}\} \times \delta(q, \sigma, \tau) & \text{if } \tau = \gamma, \\ \{b\} \times \delta(q, \sigma, \tau) & \text{otherwise,} \end{cases}$$

for all $b \in \{\text{yes}, \text{no}\}$, $\sigma \in \Sigma \cup \{\varepsilon\}$, and $\tau \in \Gamma \cup \{\varepsilon\}$. Now that P' has been constructed, convert it to a context-free grammar and check whether its language is nonempty.

3 For a nonempty language L, the *longest common prefix* of L is denoted $\mathrm{lcp}(L)$ and defined as the longest string w such that all strings in L start with w. For example, $\mathrm{lcp}(\{0111, 0101\}) = 01$ and $\mathrm{lcp}(0^+ \cup 1^+) = \varepsilon$.

a. For every nonempty language L, prove that $\mathrm{lcp}(L)$ exists and is unique.

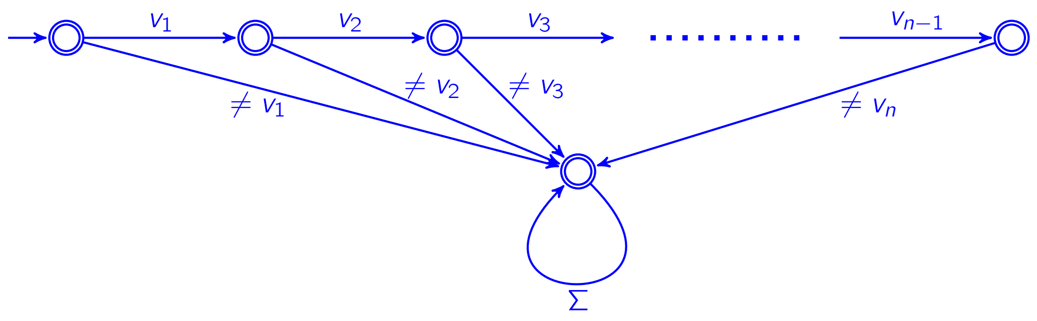
b. Give an algorithm that inputs a PDA P with $L(P) \neq \varnothing$ and computes $\mathrm{lcp}(L(P))$.

Solution.

a. If u and v are common prefixes of L, then one of them is a prefix of the other. Therefore, the only way they can both be longest prefixes of L is if $u = v$. This proves the uniqueness of $\mathrm{lcp}(L)$. To prove existence, consider the set A of common prefixes of L. Then A is nonempty (because $\varepsilon \in A$) and finite (because the strings in A are bounded in length by the shortest string in L). Thus, A has a longest string.

b. We compute $\mathrm{lcp}(L(P))$ iteratively, building it up one character at a time starting with the empty prefix. Formally, initialize $p \leftarrow \varepsilon$. In each iteration, we look for $\sigma \in \Sigma$ such that $L(P) \subseteq p\sigma\Sigma^*$. If found, we set $p \leftarrow p\sigma$ and start the next iteration; if not, we know that p is maximal and hence $\mathrm{lcp}(L(P)) = p$.

The basic operation in this algorithm is testing whether $L(G) \subseteq v\Sigma^*$, for a nonempty string $v = v_1 v_2 \ldots v_n$. To implement this test, start by constructing an NFA for $\overline{v\Sigma^*}$:



Then apply the Cartesian product construction to P and N to obtain a PDA P' for $L(P) \cap \overline{v\Sigma^*}$. Finally, convert P' to a context-free grammar and check whether its language is empty.

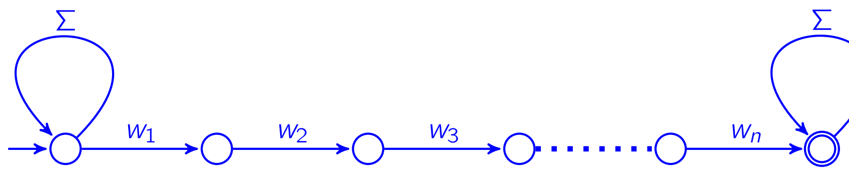
4 For a language L, a *motif* of L is any string that occurs as a substring in infinitely many strings of L. Prove that the following problems are decidable:

a. on input a DFA D and a string w, decide whether w is a motif of $L(D)$;

b. on input a DFA D, compute the number of motifs of $L(D)$ (an integer or "∞").

Solution.

a. Let $w = w_1 w_2 \ldots w_n$ be given. By definition, w is a motif iff $L(D) \cap (\Sigma^* w \Sigma^*)$ is infinite. With this in mind, construct an NFA N for $\Sigma^* w \Sigma^*$:



Next, convert this NFA to a DFA and combine the result with D via the Cartesian product construction to produce a DFA for $L(D) \cap (\Sigma^* w \Sigma^*)$. Finally, use the algorithm from class to check whether this new DFA accepts infinitely many strings.

b. If $L(D)$ is finite, then by definition L has no motifs. If $L(D)$ is infinite, then the pumping lemma ensures that $xy^*z \subseteq L$ for some nonempty string y, which means that L has infinitely many motifs $(y, y^2, y^3, \ldots)$. This motivates the following solution: use the technique from class to check whether $L(D)$ is infinite, and output "∞" in that case and 0 otherwise.

5 For each problem below, determine whether it is decidable and prove your answer:

a. on input a Turing machine M, decide whether $L(M)$ is regular;

b. on input a Turing machine M that recognizes a regular language, compute a regular expression for $L(M)$;

c. on input a Turing machine M, decide whether M accepts every input in at most 2019 steps.

Solution.

a. Undecidable. Let $\mathscr{C}$ be the set of regular languages. Then $\mathscr{C}$ is a nonempty proper subset of Turing-recognizable languages because, for example, $\mathscr{C}$ contains 0^* and does not contain $\{0^n1^n : n \geqslant 0\}$. Rice's theorem now implies that it is undecidable, on input a Turing machine M, whether $L(M) \in \mathscr{C}$.

b. Undecidable. For the sake of contradiction, suppose that this problem can be solved by some Turing machine T. Then the following algorithm is a decider for the halting problem.

```
Require: Turing machine $M$, string $w$
1: $newTM$ ← ┌─────────────────┐
             │ Input: $x$        │
             │ Run $M$ on $w$      │
             │ Accept $x$        │
             └─────────────────┘
2: if $T(newTM) = \Sigma^*$ then
3:     return "$M$ halts on $w$"
4: else
5:     return "$M$ does not halt on $w$"
6: end if
```

On input M, w, the above algorithm constructs a Turing machine called $newTM$ such that $L(newTM) = \Sigma^*$ if M halts on w, and $L(newTM) = \varnothing$ otherwise. Thus, the language of $newTM$ is regular and reveals whether M halts on w. Next, the algorithm runs T on $newTM$ and checks whether the resulting regular expression is equivalent to Σ^*, using the equivalence test for regular expressions from class. Altogether this gives a decider for the halting problem. Since the halting problem is undecidable, we conclude that T cannot exist in the first place.

c. Decidable. A Turing machine that runs for at most 2019 steps cannot go beyond the first 2019 symbols of the input. This motivates the following algorithm. Run M on every string of length at most 2019, cutting off each such execution at 2019 steps. If all these executions result in acceptance, output "yes." Otherwise, output "no."