Homework 6: Turing Machine Variants

Theory of Computing (CSE 30151), Spring 2024

Due: **Thursday** 2023-03-28 5pm

Instructions

- Create a PDF file (or files) containing your solutions. You can write your solutions by hand, but please scan them into a PDF.
- Please name your PDF file(s) as follows to ensure that the graders give you credit for all of your work:
  - If you're making a complete submission, name it `netid-hw6.pdf`, where `netid` is replaced with your NetID.
  - If you're submitting some problems now and want to submit other problems later, name it `netid-hw6-part123.pdf`, where `123` is replaced with the problem number(s) you are submitting at this time.
- Submit your PDF file(s) in Canvas.

Problems (10 points each)

1. **Doubly infinite tapes** [Problem 3.11]. A Turing machine with a doubly infinite tape is like a TM as defined in the book, but with a tape that extends infinitely in both directions (not just to the right). Initially, the head is at the first symbol of the input string, as usual, but there are infinitely many blanks to the left. Show how, given a TM with doubly infinite tape, to construct an equivalent standard TM. An **implementation description** in the style of Proof 3.13 is fine, and it's also fine to use any results proved in the book or in class.

2. **Two-stack PDAs.** A *two-stack pushdown automaton (2PDA)* is a pushdown automaton with two stacks. It has a start state and zero or more accept states like a standard PDA, and its transitions look like this:

$$q \xrightarrow{a, x_1, x_2 \to y_1, y_2} r$$

This means, if the machine is in state q, the next input symbol is a, the top of the first stack is x_1, and the top of the second stack is x_2, then consume a, pop x_1 from the first stack, pop x_2 from the second stack, push y_1 onto the first stack, push y_2 onto the second stack, and go to state r.

Show that any Turing machine M can be converted into an equivalent 2PDA P. Use formal descriptions of both M and P. Be sure to include in your construction the following:

- For each state q of M, you should create a state q in P.
- If s is the start state of M, what should you do?
- If q_{accept} is the accept state of M, what should you do?
- If q_{reject} is the reject state of M, what should you do?
- For each transition of M that looks like this, what should you do?

$$q \xrightarrow{a \to b, \mathrm{R}} r$$

- For each transition of M that looks like this, what should you do?

$$q \xrightarrow{a \to b, \mathrm{L}} r$$

3. **Brain fun.** This problem is about a programming language known as $\mathcal{P}''$ in polite company.[1] It was invented in 1964, in one of the foundational papers about structured programming, to show that we don't need `goto`.

   Let $\Gamma = \{a_0, \ldots, a_{n-1}\}$ and $\Sigma \subseteq \Gamma \setminus \{a_0\}$. A $\mathcal{P}''$ program works on a singly-infinite tape like a Turing machine. Each cell contains a symbol from Γ. The tape is initialized to an input string over Σ, followed by infinitely many a_0's. The head starts at the leftmost cell. Then a sequence of commands is executed sequentially. The possible commands are as follows:

`<`	Move the head to the left if possible; do nothing otherwise.
`>`	Move the head to the right.
`+`	Increment the symbol under the head: a_0 becomes a_1, a_1 becomes a_2, and so on; a_{n-1} becomes a_0.
`-`	Decrement the symbol under the head: a_{n-1} becomes a_{n-2}, a_{n-2} becomes a_{n-3}, and so on; a_0 becomes a_{n-1}.
`[` *cmds* `]`	Like a `while` loop: `while` the symbol under the head is not a_0 `do` *cmds*. These loops can be nested.

[1] `https://bit.ly/pprimeprime`

When the program finishes, if the symbol under the head is not a_0, the program accepts the input string; otherwise it rejects.

For example, the following program (with $\Sigma = \{a_1, \ldots, a_{n-1}\}$) recognizes the language $\{a_i a_j w \mid i + j \neq n, w \in \Sigma^*\}$:

```
[->+<]>
```

That's equivalent to the following pseudocode:

while $\mathit{tape}[\mathit{head}] \neq 0$ **do**
  $\mathit{tape}[\mathit{head}] \mathrel{-}= 1 \pmod{n}$
  $\mathit{head} \mathrel{+}= 1$
  $\mathit{tape}[\mathit{head}] \mathrel{+}= 1 \pmod{n}$
  $\mathit{head} \mathrel{-}= 1$
$\mathit{head} \mathrel{+}= 1$
return $\mathit{tape}[\mathit{head}] \neq 0$

Choose **one** of the following problems. If you do more than one, you'll get credit for the best one.

(a) Describe how to compile any $\mathcal{P}''$ program P into the **formal description** of a Turing machine M_P equivalent to P. The input to M_P would be a string $w \in \Sigma^*$, and it should accept iff P accepts w. It should be a standard single-tape TM, but you can use S ("stay") actions.

(b) Give an **implementation description** of a multitape Turing machine that can interpret any $\mathcal{P}''$ program. The input would be a string $P\texttt{\#}w$ where P is a $\mathcal{P}''$ program and w is an input string, and it should accept iff P accepts w.

(c) Much harder: Describe how to translate any Turing machine M into a $\mathcal{P}''$ program P_M equivalent to M.