CPSC 351 — Tutorial Exercise #10

More about Turing Machines

This exercise is intended to help you to understand multi-tape Turing machines. It also includes a second "Turing machine design exercise" — intended to show show using multi-tape Turing machines (and the use of solutions for problems that have been solved already) can make this easier.

Getting Started

It is unlikely that this problem — which students should be able to complete — will be discussed in the tutorial for this exercise.

1. Let $\Sigma = \{0, 1\}$. Consider a 3-tape Turing machine

   $$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

   such that $\Gamma = \{0, 1, \$, \sqcup\}$, that is as shown in Figure 1 on page 2.

   As usual, to keep the picture simple, the accept state is shown in the picture as "q_A" instead of "q_{accept}". Transitions given short names in the picture, and these names correspond to the following.

   A_0: $0, \sqcup, \sqcup/0, \mathtt{S}, \$, \mathtt{R}, \$, \mathtt{R}$
   A_1: $1, \sqcup, \sqcup/1, \mathtt{S}, \$, \mathtt{R}, \$, \mathtt{R}$
   $A_\sqcup$: $\sqcup, \sqcup, \sqcup/\sqcup, \mathtt{S}, \sqcup, \mathtt{S}, \sqcup, \mathtt{S}$
   B_0: $0, \sqcup, \sqcup/0, \mathtt{R}, 0, \mathtt{R}, \sqcup, \mathtt{S}$
   B_1: $1, \sqcup, \sqcup/1, \mathtt{R}, \sqcup, \mathtt{S}, 1, \mathtt{R}$
   C: $\sqcup, \sqcup, \sqcup/\sqcup, \mathtt{S}, \sqcup, \mathtt{L}, \sqcup, \mathtt{L}$
   D: $\sqcup, 0, 1/\sqcup, \mathtt{S}, \sqcup, \mathtt{L}, \sqcup, \mathtt{L}$
   E: $\sqcup, \$, \$/\sqcup, \mathtt{S}, \sqcup, \mathtt{S}, \sqcup, \mathtt{S}$

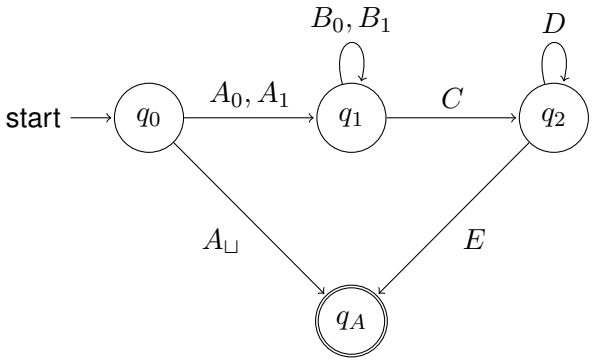

Figure 1: A Multi-Tape Turing Machine

Transitions that are not shown go to the ***reject*** state (which is also not shown). In particular, if $q \in \{q_0, q_1, q_2)$ and a transitions for and input $(q, \sigma_1, \sigma_2, \sigma_3)$ is not shown, where $\sigma_1, \sigma_2, \sigma_3 \in \Gamma$, then the transition is as follows:

$$\delta(q, \sigma_1, \sigma_2, \sigma_3) = (q_{\text{reject}}, \sigma_1, \text{S}, \sigma_2, \text{S}, \sigma_3, \text{S}).$$

Show the sequence of configurations that are reached when M is executed on each of the following input strings. Then say whether each of these strings is in the language of this Turing machine.

(a) λ
(b) `0`
(c) `1`
(d) `10`
(e) `010`

Finally, describe the language of this machine and explain what it is using its tapes to do, in order to decide whether an input string should be accepted.

Problem To Be Discussed in the Tutorial

Once again, let $\Sigma_b = \{0, 1\}$, let $\Sigma_p = \{0, 1, \#\} = \Sigma_b \cup \{\#\}$, and let $L_{\text{pair}} \subseteq \Sigma_p^\star$ be the set of strings in $\Sigma_p^\star$ with the form

$$\mu\#\nu \tag{1}$$

where $\mu, \nu \in L_{\text{bin}}$, for the language $L_{\text{bin}} \subseteq \{0, 1\}^\star$ introduced in the lecture presentation for Lecture #11 —so that μ and ν are both ***unpadded binary representations*** of non-negative integers. Let $f_{\text{mult}} : \Sigma_p^\star \to \Sigma_b^\star$ be defined as follows:

- If $\omega = \mu\#\nu \in L_{\text{pair}}$ so that, in particular, μ and ν are unpadded binary representations of non-negative integers n and m, then $f_{\text{mult}}(\omega)$ is the unpadded binary representation of their product, $n \times m$.
- On the other hand, if $\omega \in \Sigma_p^\star$ and $\omega \notin L_{\text{pair}}$, then $f_{\text{mult}}(\omega) = \lambda$.

The goal of the following questions is to design a multi-tape Turing machine, with ***seven tapes***, that computes the function f_{mult}. When working on these questions it might be useful to consider the (standard) Turing machines that compute the functions $f_{+1}, f_{-1} : \Sigma_b^\star \to \Sigma_b^\star$ that were described in the lecture presentation for Lecture #11 and the solutions for Tutorial Exercise #9, respectively, as well as the 3-tape Turing machine, computing the function $f_{\text{add}} : \Sigma_p^\star \to \Sigma_b^\star$ that was described in the lecture presentation for Lecture #11.

To begin, consider the algorithm shown in Figure 2 on page 4 — which will be used as a "high-level" description of the Turing machine to be implemented.

2. One can see, by inspection of the code, that this algorithm returns $f_{mult}(\omega)$ when executed on an input string $\omega \in \Sigma_p^\star$ such that $\omega \notin L_{\text{pair}}$ — because the test at line 1 fails, so that the step at line 8 is reached and executed, and since $f_{\text{mult}}(\omega) = \lambda$ in this case. It remains only to consider the case that $\omega \in L_{\text{pair}}$.

   Consider an execution of this algorithm on an input string $\omega = \mu\#\nu$, where μ and ν are unpadded binary representations of non-negative integers n and m, respectively. In this case the test at line 1 is passed, so that the steps at lines 2 and 3 are reached and executed. At this point $\widehat{\nu}$ and φ are, respectively, the unpadded binary representations of the non-negative integers m and 0.

   (a) It can be shown, by induction on i, that the test at line 4 is reached and executed at least i times, for every integer i such that $1 \leq i \leq m + 1$. The variables $\widehat{\nu}$ and φ are the unpadded binary representations immediately before the test at line 4 is executed for the i^{th} time. Say what these integers (which, somehow, depend on n, m and i) are at this point.

On input $\omega \in \Sigma_p^\star$:

1. `if` ($\omega \in L_{\text{pair}}$) `{`
   // Let $\mu, \nu \in \{0,1\}^\star$ such that $\omega = \mu\#\nu$
2. $\widehat{\nu} := \nu$
3. $\varphi := 0$
4. `while` ($\widehat{\nu} \neq 0$) `{`
5. $\varphi := f_{\text{add}}(\varphi\#\mu)$
6. $\widehat{\nu} := f_{-1}(\widehat{\nu})$
   `}`
7. `return` φ

   `} else {`
8. `return` λ

   `}`

Figure 2: Algorithm to Compute the Function f_{mult}

(b) Use this to sketch a (very brief) proof that this algorithm corrects the function f_{mult}, as desired.

As noted above we are trying to design a multi-tape Turing machine with ***seven*** tapes. These tapes will be used as follows.

- Tape #1 will store the input string, ω.

If $\omega \notin L_{\text{pair}}$ then the other tapes will never be used at all. Otherwise, $\omega = \mu\#\nu$ for a pair of strings $\mu, \nu \in \{0,1\}^\star$ — and the remaining tapes will be used as follows.

- Tape #2 will store the string $\widehat{\nu}$, which is initialized when the step at line 2 is reached and executed.
- Tape #3 will store the string φ, which is initialized when the step at line 3 is reached and executed.
- Tapes #4, #5, #6 will be used when the step at line 5 is carried out.
- Tape #7 will (eventually) store the output string, $f_{\text{mult}}(\omega)$.

The tape alphabet,

$$\Gamma = \{0, 1, \#, \mathtt{X}, \sqcup\},$$

used for an multi-tape Turing machine to compute the function f_{add} (in the lecture presentation for Lecture #11) can be used as the tape alphabet for this Turing machine, as well.

The algorithm to compute f_{add} began by checking whether its input string belongs to L_{pair} — just as the above algorithm does. The same implementation of this step (which only examines or updates cells, or move its tape head, on the first tape) can be used for this step in both of these algorithms. You may assume that the contents of all tapes and locations of all tape heads are the same after this test as they were before it.

3. Consider the continuation of the execution on a input string $\omega \in \Sigma_p^\star$ such that $\omega \in L_{\text{pair}}$, so that $\omega = \mu\#\nu$, for the unpadded binary representations, μ and ν, of a pair of non-negative integers n and m.

   (a) Consider the step at line 2. What should M's tapes look like, before and after this step is executed? Give an "implementation-level" description of this step — including some detail about how states and transitions out of them can be used to carry this out, assuming that the Turing machine is in a state "q_{valid}" when this step begins.

   (b) Consider the step at line 3 in the same way. You should find that this is considerably simpler than it was for the step at line 2.

   (c) Describe how the test at line 4 can be implemented. This should also be easy to do.

   (d) Recall that, in the lecture presentation for Lecture #11, a 3-tape Turing machine, "M_{add}", was described that computes the function f_{add}. Explain how this could be used (with moves before and after) in an implementation of the step at line 5.

   (e) Recall as well that a (standard) Turing machine computing the function f_{-1} was described in the solutions for Tutorial Exercise #9.[1] Explain how this could be used in an implementation of the step at line 6.

   (f) Briefly describe what else must be done to complete an implementation-level description of a 7-tape Turing machine that computes the function f_{mult}.

4. A ***formal description*** of a Turing machine, corresponding to the implementation-level description that has now been outlined, would be *very* large. Explain how could the process for Turing-design, now being described and used, be helpful when one tries to present and explain a formal description of this Turing machine.

5. Explain how the use of a process like this could also be helpful when you try to present a proof that your Turing machine is correct — that is, that it really *does* compute the function f_{mult}.

[1] You may assume that this Turing machine exists, even if you have not seen the solutions for that tutorial exercise yet.