Lecture #7: Simulations and Turing Machine Variants

What To Do Before the Lecture

1. Watch the videos for Lecture #7 —noting that they will probably be understandable if you play them at double speed. If you do not have time for this then look at the "Key Concepts" document that is found, immediately after the videos for this lecture, on the course web site, instead.
2. ***Print*** and read through the rest of this document and — if you have time — try to solve the problems! These should help you to check that you understand how Turing machines process strings, and that you understand how to prove things about them.
3. The supplemental material includes the completion of a proof of equivalence of two Turing machine variants that was started in the lecture videos. It also includes additional details about multi-tape Turing machines that might be helpful as you work with these. Finally, it includes additional information about nondeterministic Turing machines and the Church-Turing thesis that will not be needed for this course but that might be of interest.

Problems To Be Solved

Turing Machines with Two-Way Infinite Tapes

A New Turing Machine Variant

A ***Turing machine with a two-infinite tape*** is similar to an ordinary Turing machine, but its tape is infinite to the left as well as to the right. The tape is initially filled with blanks except for the portion that contains the input — and the tape head is initially pointing to the leftmost symbol in the input if the input is nonempty. Computation is as usual, except that the tape head never encounters an end of the tape as it moves left.

This kind of Turing machine can also be modelled as a 7-tuple

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

where Q, Σ, Γ, δ, q_0, q_{accept} and q_{reject} are exactly as described as for a standard Turing machine. The most significant difference is that, since there is no "leftmost cell" of the tape, an attempt to move left, past the initial position of the tape head, will be successful — with a copy of "$\sqcup$" being visible, when this attempt (initially) succeeds.

A Reduction in One Direction

Claim. *Let $L \subseteq \Sigma^\star$ for an alphabet Σ. If L is Turing-recognizable — so that there is a standard (one-tape) Turing machine M with input alphabet Σ and language L. Then there exists a Turing machine with a two-way infinite tape, $\widehat{M}$ with alphabet Σ such that $L(\widehat{M}) = L$ as well. Furthermore, if M decides L then $\widehat{M}$ decides L, as well.*

Using $\widehat{M}$ to Represent a Configuration of M

Initialization

Simulating a Move of M

Completing the Proof

Reduction in Another Direction

Claim. *Let $L \subseteq \Sigma^\star$ for an alphabet Σ. If there exists a Turing machine with a two-way infinite tape, M, with input alphabet Σ and language L, then L is Turing-recognizable — so that there exists a (standard) Turing machine $\widehat{M}$ with alphabet Σ such that $L(\widehat{M}) = L$ as well. Furthermore, if M decides L then $\widehat{M}$ decides L, as well.*

Using M to Represent a Configuration of M

Initialization

Simulating a Move of M

Completing the Proof

Designing a Multi-Tape Turing Machine for Additional of Binary Numbers

During Lecture #6 (and in a supplemental document for it) ***Turing machines that compute functions*** were introduced. Now that ***multi-tape Turing machines*** (that recognize languages) have been introduced, ***multi-tape Turing machines that compute functions*** can be introduced too — as described in the supplemental document, "More about Multi-Tape Turing Machines", for this lecture.

Let $\Sigma_{\text{binary}} = \{0, 1\}$, and let $f_{\text{b\_inc}} : \Sigma^\star_{\text{binary}} \to \Sigma^\star_{\text{binary}}$ such that the following properties are satisfied for every string $\omega \in \Sigma^\star_{\text{binary}}$:

- If ω is the unpadded binary representation of a non-negative integer n then $f_{\text{b\_inc}}(\omega)$ is the unpadded binary representation of $n + 1$.
- On the other hand, if ω is *not* the unpadded binary representation of any non-negative integer, then $f_{\text{b\_inc}}(\omega) = \lambda$, the empty string.

Let $f_{\text{b\_dec}} : \Sigma^\star_{\text{binary}} \to \Sigma^\star_{\text{binary}}$ such that the following properties are satisfied for every string $\omega \in \Sigma^\star_{\text{binary}}$:

- If ω is the unpadded binary representation of a *positive* integer n, then $f_{\text{b\_dec}}(\omega)$ is the unpadded binary representation of $n - 1$.
- On the other hand, if ω is *not* the unpadded binary representation of any positive integer, then $f_{\text{b\_dec}}(\omega) = \lambda$, the empty string.

In the supplemental document for Lecture #6 about Turing machines that compute functions, a Turing machine that computes the above function $f_{\text{b\_inc}}$ was presented, along with a proof of its correctness. This was continued in the additional practice problems for Tutorial #6; if these practice problems were solved then a Turing machine that computes the function $f_{\text{b\_dec}}$ was obtained, and proved to be correct, as well.

Now let $\Sigma_{\text{pair}} = \{0, 1, \#\}$ and consider a function $f_{\text{add}} : \Sigma^\star_{\text{pair}} \to \Sigma^\star_{\text{binary}}$ such that the following properties are satisfied for every string $\omega \in \Sigma^\star_{\text{pair}}$:

- If $\omega = \mu \# \nu$, where $\mu, \nu \in \Sigma^\star_{\text{binary}}$ are the unpadded binary representations of a pair of non-negative integers n and m, respectively, then $f_{\text{add}}(\omega)$ is the unpadded binary representation of their sum, $n + m$.
- Otherwise $f_{\text{add}}(\omega) = \lambda$, the empty string.

Consider the algorithm shown in Figure 1, on the following page.

On input $\omega \in \Sigma_{\text{pair}}$:

1. If ω has the form $\mu \# \nu$, where μ and ν are the unpadded binary representations of non-negative integers then write μ onto Tape #2 and write ν onto Tape #3, erasing Tape #1 and moving all tape heads to their leftmost positions. That is, go from the configuration

$$q_0 \, \omega \, \sharp \, q_0 \, \sharp \, q_0 \;\; = \;\; q_0 \, \mu \# \nu \, \sharp \, q_0 \, \sharp \, q_0$$

to a configuration

$$q_1 \, \sharp \, q_1 \, \mu \, \sharp \, q_1 \, \nu$$

and continue to the next step. On the other hand, if ω does not have this form then erase the first tape, moving the tape head back to its leftmost position, and halt — that is, go from the configuration $q_0 \, \omega \, \sharp \, q_0 \, \sharp \, q_0$ to the configuration

$$q_{\text{halt}} \, \sharp \, q_{\text{halt}} \, \sharp \, q_{\text{halt}}$$

Let n and m be the non-negative integers whose representations are on Tapes #2 and #3, respectively.

2. `while` $(n > 0)$ `{`
3. $n := n - 1$ (updating Tape #2 to make this change)
4. $m := m + 1$ (updating Tape #3 to make this change)
   `}`
5. Erase the copy of "0" that is now on Tape #2 — so that the tape head for this points to its leftmost cell — and go to state q_{halt}, in order to return the unpadded binary representation of m.

Figure 1: An Algorithm to Compute the Function f_{add}

Suppose, in particular, that this is being implemented using a 3-tape Turing machine, with input alphabet Σ_{pair}, whose tape alphabet also includes "dotted" copies of the symbols in Σ_{pair}, that is, symbols $\dot{0}$, $\dot{1}$, and $\dot{\#}$.

Proving the Correctness of This Algorithm