Lecture #7: Simulations and Turing Machine Variants
Key Concepts

Simulations

Recall that a ***simulation*** is something that can be presented to relate the power of two models of computation. In order to show that the machines described by a ***second*** model of computation are (in some sense) at least as powerful or efficient as the machines described by a ***first*** model of computation, we generally do the following:

(a) Consider an arbitrary machine M, of the type described by the ***first*** model of computation.

(b) Use M to define another machine $\widehat{M}$, of the type described by the ***second*** model of computation.

(c) Prove that $\widehat{M}$ solves the same problem as M.

Step (b) can be expanded as follows:

- Begin by describing — as clearly and completely as you can — how a configuration of the first machine, M can be represented when a machine, $\widehat{M}$, of the second type is being used.

  When $\widehat{M}$ is a kind of Turing machine this will, generally, including describing how many tapes $\widehat{M}$ has, and what they are used for. It might also include describing $\widehat{M}$'s tape alphabet, $\widehat{\Gamma}$.

This will, ideally, make it significantly easier, to describe the following:

- ***Initialization:*** Let Σ be the input alphabet for M — so that it must be the input alphabet for $\widehat{M}$, as well. Describe how $\widehat{M}$ begins with its initial configuration for an input string $\omega \in \Sigma^\star$ and moves to a representation of M's initial configuration for the same input string.

- ***Step-by-Step Simulation:***

  For configurations $\mathcal{C}_1$ and $\mathcal{C}_2$ of M, such that M moves from configuration $\mathcal{C}_1$ to configuration $\mathcal{C}_2$ using a single step, describe how $\widehat{M}$ moves from a representation of $\mathcal{C}_1$ to a representation of $\mathcal{C}_2$.

- ***Cleanup:***

  Describe anything more that $\mathcal{M}$ must do, to end its computation, after simulating M's final step.

  Note: For Turing machines that recognize (or decide) languages, there might not be anything, here, to describe.

If $\widehat{M}$ is a Turing machine then it is possible that $\widehat{M}$s's transition function, $\widehat{\delta}$, has been described in detail once these steps have been completed. Alternatively, if the simulation is more complex, then enough information has been given so that it *could* be completed, if you had time.

If it is not obvious then a ***proof of correctness*** of the simulation should be given. When M and $\widehat{M}$ are types of Turing machines, that each have an input alphabet Σ, then this should imply the following: For every string $\omega \in \Sigma^\star$ and for every non-negative integer t, the following property is satisfied: If the execution of M on input ω uses at least t steps, and M is in configuration $\mathcal{C}$ after the first t steps of its execution on input ω, then $\widehat{M}$'s execution on input ω includes a simulation of at least t steps of M and — after the simulation of t steps of M — $\widehat{M}$ is in a configuration that gives a representation of $\mathcal{C}$.

Multi-Tape Turing Machines

For any (fixed) positive integer k, a ***k*-tape Turing machine** is a generalization of a Turing machine that has k tapes — all of whose tape heads can move independently. Transitions now allow tape heads to move ***left***, move ***right*** or ***stay*** — so that a k-tape Turing machine can be modelled as a 7-tuple

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

where Q, Σ, Γ, q_0, q_{accept} and q_{reject} are all as they are for standard Turing machines, and where

$$\delta : Q \times \Gamma^k \to Q \times (\Gamma \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\})^k$$

is a partial function such that $\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k)$ is defined whenever $q \in Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}$ and $\sigma_1, \sigma_2, \ldots, \sigma_k \in \Gamma$, and $\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k)$ is undefined whenever $q \in \{q_{\text{accept}}, q_{\text{reject}}\}$ and $\sigma_1, \sigma_2, \ldots, \sigma_k \in \Gamma$.

A ***supplemental document*** (whose first few pages should be examined, as needed) gives more information about how configurations for these Turing machines can be represented, how they process strings, and how the languages of these machines are defined.

The first claim should not be surprising, since any standard Turing machine is a 1-tape Turing machine:

Claim. *Let $\subseteq \Sigma^\star$ (for some alphabet Σ).*

(a) If L is Turing-recognizable then there is a k-tape Turing machine, for some integer $k \geq 1$, that recognizes L.

(b) If L is Turing-decidable then there is a k-tape Turing machine, for some integer $k \geq 1$, that decides L.

A rather complicated ***simulation*** can be used to establish the following too:

Claim. *Let $L \subseteq \Sigma^\star$ (for some alphabet Σ). Let k be any integer such that $k \geq 1$.*

(a) If there is a k-tape Turing machine M that recognizes L then there is also a (standard, one-tape) Turing machine $\widehat{M}$ that recognizes L, so that L is Turing-recognizable.

(b) If there is a k-tape Turing machine M that decides L then there is also a (standard, one-tape) Turing machine $\widehat{M}$ that decides L, so that L is Turing-decidable.

A ***multi-tape Turing machine*** is a k-tape Turing machine for some positive integer k.

The above claims imply that the sets of "Turing-recognizable" and "Turing-decidable" languages would not be changed if they were defined using multi-tape Turing machines instead of standard Turing machines.

Multi-tape Turing machines that compute functions $f : \Sigma_1^\star \to \Sigma_2^\star$ were also defined: When $f(\omega)$ is defined, then the Turing machine's execution, on input ω, must halt with $f(\omega)$ written on the ***k^{th} tape***, with all other tapes filled with copies of "$\sqcup$", and with all tape heads at the leftmost cells of the tape.

It can be proved, for all alphabets Σ_1 and Σ_2, that a (partial or total) function $f : \Sigma_1^\star \to \Sigma_2^\star$ is Turing-computable if and only if there exists a k-tape Turing machine that computes f, for some positive integer k.

Nondeterministic Turing Machines

A ***nondeterministic*** Turing machine is the same as a regular Turing machine except that there can be **zero, one, or *many*** moves that might be possible at any time — so that the transition function is now a *total* function

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathtt{L}, \mathtt{R}\}).$$

— such that $\delta(q_{\text{accept}}, \sigma) = \delta(q_{\text{reject}}, \sigma) = \emptyset$ for every symbol $\sigma \in \Gamma$.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, qR)$ be a nondeterministic Turing machine and let $\omega \in \Sigma^\star$.

- M ***accepts*** ω if *there exists* at least one (finite) sequence of moves, beginning in M's initial configuration for ω, that ends with M in state q_{accept}.
- M ***rejects*** ω if *every* sequence of moves of M, that begins with the initial configuration for ω, is finite — and there are no sequences of moves that end with M in state q_{accept}.
- M ***loops on*** ω otherwise.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be a nondeterministic Turing machine and let $L \subseteq \Sigma^\star$.

- M ***recognizes*** L if M accepts every string $\omega \in L$ and M either rejects or loops on every string $\omega \in \Sigma^\star$ such that $\omega \notin L$.
- M ***decides*** L if M accepts every string $\omega \in L$ and M rejects every string $\omega \in \Sigma^\star$ such that $\omega \notin L$ — so that M does not loop on any string in $\Sigma^\star$.[1]

Claim. *Let $\subseteq \Sigma^\star$ (for some alphabet Σ).*

(a) L is Turing-recognizable if and only if there exists a nondeterministic Turing machine M such that M recognizes L.

(b) L is Turing-decidable if and only if there exists a nondeterministic Turing machine M such that M decides L.

An optional supplemental document includes a sketch of a proof of this claim.

The Church-Turing Thesis

The ***Church-Turing Thesis*** is a widely held belief that Turing machines (and the sets of languages and functions defined using them) really *do* model computability. Additional details about — quite important — thesis are given in a supplemental document about this.

[1] An even *stronger* condition is sometimes required, in order to say that a nondeterministic Turing machine "decides" a language — but adding this does not change the set of languages that are "decidable" by nondeterministic Turing machines.