Lecture #6: Computation on Strings — Intermediate Models

Lecture Presentation

What We Will Discuss

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 the ***second*** model of computation are (in some sense) at least as powerful as the machines described by the ***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.

Where Have You — Ideally — Seen Simulations, Before This (in "Theory" Courses)?

Examples:

- A simulation was — ideally — used in CPSC 351 (or CPSC 313) to show that every language of a ***deterministic multi-tape Turing machine*** is also the language of a ***deterministic single-tape Turing machine***
- A simulation is described, in the lecture notes, to prove "Claim #3" — that is, to show that if $f : (\Sigma^\star)^k \to \Sigma^\star$ is a (partial or total) function, that is $\mathcal{S}_n$-computable, for some positive integer n, then there exists a Post-Turing program that strongly computes f.
- Another simulation is described, in the lecture notes, to prove "Claim #4" — that is, to show that if $f : \mathbb{N}^k \to \mathbb{N}$ is a (partial or total) function that is weakly compute by a Post-Turing program, then f is $\mathcal{S}$-computable as well.

Complication

Different machines —corresponding to different models of computation — represent information in different ways.

First Example: Simulating a Multi-Tape Turing Machine with a Single-Tape Turing Machine

During an execution of a k-tape deterministic Turing machine, information is represented using

- its current state
- the contents of the non-blank parts of its k tapes (each beginning with the symbol at the leftmost cell on the tape)
- the locations of the tape heads for each of its k tapes

During an execution of a single-tape deterministic Turing machine, information is represented using

- its current state
- the contents of the non-blank part of its (single) tape —beginning with the symbol at the leftmost cell on the tape
- the locations of the tape head for its (single) tape)

Something We Can Take Advantage Of: The set of states, and the tape alphabet, for the single-tape Turing machine (being developed) can depend on — but can also be very different from — the set of states, and the tape alphabet, for the k-tape Turing machine (that we are given, or whose existence we have assumed).

Second Example: Simulating an $\mathcal{S}_n$-Program with a Post-Turing Machine

During the execution of an $\mathcal{S}_n$-program, information is represented using

-
-

During the execution a Post-Turing program, information is represented using

-
-
-

Third Example: Simulating a Post-Turing Machine with an $\mathcal{S}$-Program

During the execution a Post-Turing program, information is represented using

-
-
-

During the execution of an $\mathcal{S}$-program, information is represented using

-
-

What This Implies

When developing a simulation we must generally start by describing how the information that is represented, at some point during an execution of the first machine, can be represented during an execution of the second machine, instead.

First Example: Simulating a Multi-Tape Turing Machine with a Single-Tape Turing Machine

Consider an execution of a k-tape Turing machine

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

At this point during the execution, $\mathcal{M}$ might be in a state $q \in Q$ and its tapes might be as follows:

Suppose we simulate $\mathcal{M}$ using a single-tape Turing machine

$$\widehat{\mathcal{M}} = (\widehat{Q}, \Sigma, \widehat{\Gamma}, \widehat{\delta}, \widehat{q}_{\text{accept}}, \widehat{q}_{\text{reject}})$$

How Should $\widehat{Q}$ Be Related To Q?

How Might $\widehat{\Gamma}$ Be Related to Γ?

How Should $\widehat{\mathcal{M}}$ Represent $\mathcal{M}$'s Information?

Second Example: Simulating an $\mathcal{S}_n$-Program with a Post-Turing Machine

Third Example: Simulating a Post-Turing Machine with an $\mathcal{S}$-Program

Initialization

Given an input — which should be the same input for both the machine being simulated, and the machine carrying out the simulation — the "simulating" machine should begin by producing its representation of the simulated machine's initial configuration for this input.

First Example: Simulating a Multi-Tape Turing Machine with a Single-Tape Turing Machine

Consider an execution of a k-tape Turing machine

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

on an input string $\omega \in \Sigma^\star$. Suppose that $\mathcal{M}$ is being simulated using a single-tape Turing machine

$$\widehat{\mathcal{M}} = (\widehat{Q}, \Sigma, \widehat{\Gamma}, \widehat{\delta}, \widehat{q}_{\text{accept}}, \widehat{q}_{\text{reject}})$$

as described above.

$\mathcal{M}$'s initial configuration, on input ω, is as follows.

$\widehat{\mathcal{M}}$'s initial configuration, on input ω, is as follows, instead:

Thus, in its initialization phase, $\mathcal{M}$ should begin in its initial configuration and produce the following representation of $\widehat{\mathcal{M}}$'s initial configuration:

The following additional information about this initialization phase would (probably) also be expected:

Second Example: Simulating an $\mathcal{S}_n$-Program with a Post-Turing Machine

Third Example: Simulating a Post-Turing Machine with an $\mathcal{S}$-Program

Step-by-Step Simulation

During one step in an execution of a "simulated" machine, there is generally a fixed (finite) set of different kinds of instructions that can be carried out. This gives a fixed (finite) set of ***cases*** that should be identified, and dealt with, when we describe how a single step of the "simulated" machine can be simulated by the "simulating" machine.

While there might be lots of details to consider, dealing with them can be straightforward once you have used this information to organize this part of the simulation.

First Example: Simulating a Multi-Tape Turing Machine with a Single-Tape Turing Machine

This example is trickier than the others! The simulation of a step is reasonably easy to consider if you break this down into stages:

- Start by using the "simulating" machine's information to discover the transition that the "simulated" machine would be carrying out:

- At this point, the information corresponding to one of the "simulated" machine's tapes can be updated — one tape at a time:

- There is only a little bit more to do, here:

- There are also a few easy “special cases” that can be handled differently:

Second Example: Simulating an $\mathcal{S}_n$-Program with a Post-Turing Machine

Third Example: Simulating a Post-Turing Machine with an $\mathcal{S}$-Program

Cleanup on Termination

If the simulated machine is computing a function with nontrivial output (instead of simply deciding whether to accept its input) then it may be necessary for the simulating machine to "clean things up", after it has been confirmed that the "simulated" machine would halt, to make sure that the desired output is represented, before the "simulating" machine can halt as well.

First Example: Simulating a Multi-Tape Turing Machine with a Single-Tape Turing Machine

Second Example: Simulating an $\mathcal{S}_n$-Program with a Post-Turing Machine

Third Example: Simulating a Post-Turing Machine with an $\mathcal{S}$-Program

Developing a Proof of Equivalence

If you have got this far then you have done most of the work that you need to! However, should make sure that your description of a “simulation” can be read by another human being.

What Can Be Learned From This?