Theory of Computation Notes: Turing Machine Variants

Arjun Chandrasekhar

1 Turing Completeness

Here we will study several different models of computation and show that all of them are equivalent to Turing machines. To show that a model of computation is equivalent to a Turing machine, we must show that a language L is Turing-recognizable if and only L can be recognized on that equivalent model of computation. This involves two directions:

1. If L can be recognized by a Turing machine, it can be recognized by the model of computation in question.
2. If L can be recognized by the model of computation in question, it can be recognized by a Turing machine.

The typical technique for these proofs is to show that the model of computation in question can simulate a Turing machine, and vice-versa.

2 Stationary Turing Machine

A **stationary Turing machine** is a Turing machine that has the option to stay in place, rather than moving left or right. The transition function is $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$.

Proposition 2.1. Stationary Turing machines are equivalent to Turing machines.

Proof. We will show that a language L is Turing-recognizable if and only if L can be recognized by a stationary Turing machine.

1. Suppose L can be recognized by a Turing machine M. Note that M **is** a stationary TM that simply chooses not to stay in place.
2. Suppose L can be recognized by a stationary TM called M. We will design a TM called M_2 that recognizes L by simulating M. We simply make M_2 behave as M would, but if M is supposed to stay in place, M_2 will move left and then move right before proceeding.

□

3 2-hop Turing Machine

A **2-hop Turing machine** is a Turing machine that has the option to move two spaces to the left or right when it transitions. The transition functions is $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, LL, RR\}$.

Proposition 3.1. 2-hop Turing machines are equivalent to Turing machines.

Proof. We will show that a language L is Turing-recognizable if and only if L can be recognized by a 2-hop Turing machine.

1. Suppose L can be recognized by a Turing machine M. Note that M **is** a 2-hop TM that simply chooses to only move one space at a time on each transition.

2. Suppose L can be recognized by a 2-hop TM called M. We will design a TM called M_2 that recognizes L by simulating M. We simply make M_2 behave as M would, but if M is supposed to jump two spaces at once, M_2 will move two spaces over, going one square at a time.

□

4 2-tape Turing Machine

A **2-tape Turing machine** is a Turing machine that has two separate tapes and tape heads. The tape heads share a common state, but they may move independently. The transition function is $\delta : Q \times \Gamma^2 \to Q \times \Gamma^2 \times \{L, R\}^2$.

Proposition 4.1. 2-tape Turing machines are equivalent to Turing machines.

Proof. We will show that a language L is Turing-recognizable if and only if L can be recognized by a 2-tape Turing machine.

1. Suppose L can be recognized by a Turing machine M. Note that M **is** a 2-tape TM that simply chooses to only use one tape.

2. Suppose L can be recognized by a 2-tape TM called M. We will design a TM called M_2 that recognizes L by simulating M. We will use the infinite tape to keep track of both of the tapes of M. At each step, M_2 simulates both of the tape heads of M. If needed, M can always move the contents of the second tape further downstream in order to make more room for tape 1.

□

5 Nondeterministic Turing Machines

A **Nondeterministic Turing machine** is a Turing machine that can make multiple choices at every step. The transition function is $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$. The TM can have many different computation paths for the same input string. The macbine accepts if at least one computation path accepts.

Proposition 5.1. Nondeterministic Turing machines are equivalent to Turing machines.

Proof. We will show that a language L is Turing-recognizable if and only if L can be recognized by a nondeterministic Turing machine.

1. Suppose L can be recognized by a Turing machine M. Note that M **is** a nondeterministic TM that has only one possible computation path.

2. Suppose L can be recognized by a nondeterministic TM called M. We will design a TM called M_2 that recognizes L by simulating M. To do this, M_2 will run M and try out every possible choice at each step. Through the course of the computation, M_2 will keep track of all possible computation paths that M could currently be in.

   A naive solution would be to simply try running each computation path to completion in a "depth first" manner. However, we do not want to get stuck on a computation path that loops forever, and never get the chance to try other choices. So we simulate each computation path in a "breadth first" manner. For each active computation path, we run M for one more step and update each computation path accordingly. This ensures that we never get stuck, because even if one computation path gets stuck in a loop it will not interfere with our ability to keep track of the progress of all the other paths. We accept if any computation path ever reaches an accepting configuration.

□