The Dynamics of Turing Machines

1 Abstract

We propose to study the behavior of Turing Machines in the context of dynamical systems. A *Turing Machine* (abbreviated "TM") is a model of computation involving a *transition function* on a space of *configurations*. We think of TMs as beginning with a finite *input string* and evolving according to the rules of the transition function; TMs can either run forever or halt in one of the *accepting* or *rejecting* state. Hence, a TM can be naturally understood as a discrete dynamical system with two fixed points. In this context the asymptotic behavior of TMs is of great interest in the field of Computability Theory; however, the problem has only recently begun being examined from the dynamical perspective. For our project we plan to examine the connection between these two disciplines, both in terms of how tools from abstract dynamical systems can help us interpret the behavior of TMs as well as how continuous dynamical systems can be approximated discretely to create analog models of computation. Particular topics of focus might include studying randomized approximation algorithms using the techniques of stability theory or investigating connections between the halting problem and limit cycles.

2 Introduction

A *Turing machine* is an abstract model of computation that allows us to encode computer programs as collections of sets with well-defined maps. There are many "equivalent" definitions for Turing machines; we'll first give the canonical one and then discuss second that will be more natural in the dynamical systems context.[1]

Definition 1 (Turing Machine)**.** A *Turing machine* is a 5-tuple

$$M = (Q, F, \Gamma, \Sigma, \delta)$$

such that the following conditions hold:

1) Q, F, Γ, Σ and F are finite sets. Note, we choose the following naming conventions:
   i) Q is called the set of *states* for M,
   ii) $F = \{\checkmark, \text{✗}\}$ is called the set of *final states* (read "accept" and "reject" respectively),
   iii) Γ is called the *work alphabet*, and
   iv) Σ is called the *input alphabet*;
2) $F \subseteq Q$ and $\Sigma \subseteq \Gamma$;
3) Q contains a distinguished element q_0, called the *start state* or *initial state*;
4) Γ contains a distinguished element $\textvisiblespace$ called the *blank character* (we require $\textvisiblespace \notin \Sigma$);
5) δ (called the *transition function*) has the following properties:

[1] Here, "equivalent" means that the models of computation they define are equally powerful. That is, given two distinct definitions D_0, D_1 for Turing machines, the behavior of any machine defined with D_0 can be simulated using a machine defined by D_1 and vice versa.

i) δ is a partial function $\delta : (Q \setminus F) \times \Gamma \nrightarrow Q \times \Gamma \times \{L, R\}$. Note the inclusion of the word *partial* — in general we do not require δ to be defined on all of $(Q \setminus F) \times \Gamma$.

ii) L, R are understood as "shift" directions, as explained below.

We think of Turing machines as operating on an infinite *work tape*. The tape starts with some finite input string and blanks everywhere else:

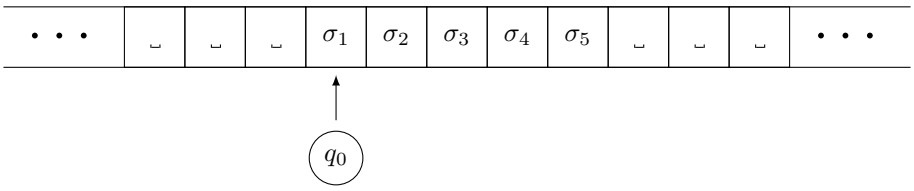

Figure 1: Turing machine with the input string $\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5$

Suppose $\delta(q_0, \sigma_1) = (q_i, \gamma_1, R)$. Then we interpret this as the Turing machine writing a γ_1 on the current tape, updating its state to q_1, and moving one space to the right on the tape.