CS301: Theory of Computation
Sections 3.2 & 3.3 - Turing Variants & Algorithms

Daniel White

Gettysburg College
Computer Science

Spring 2024

Many definitions of Turing machines exist, including versions with multiple tapes or nondeterminism. These **variants** have the same power as our original model.

The invariance to such changes is known as **robustness**, which will be demonstrated for Turing machines soon.

These results will provide easier avenues for proving languages are Turing-recognizable or decidable.

A **multitape Turing machine** is like an ordinary Turing machine with several tapes, each with its own head for reading and writing which may move left, right, or be stationary. The initial input appears on the first tape.

Formally, the transition function appears as

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

exercise: Show that the language $\{a^p \mid p \text{ prime}\}$ is decidable by a multitape Turing machine.

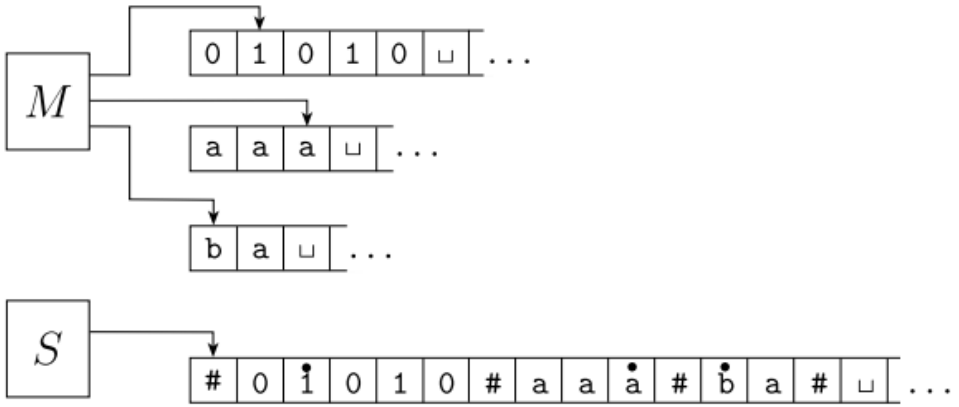
Theorem 3.13 (pg. 177)

Every multitape Turing machine has an equivalent single-tape Turing machine.

Proof. We show how to convert a multitape TM M to an equivalent single-tape TM, S. The key idea is to show how to simulate M with S ... *(see next slide)*

Corollary 3.15 (pg. 178)

A language is Turing-recognizable if and only if some multitape Turing machine recognizes it.



On input $w = w_1 w_2 \dots w_n$,

1. S puts its tape into the format that represents all tapes of M.

$$\#\dot{w}_1 w_2 \dots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \dots$$

2. To simulate a single move, S scans its tape from the first # to the last to determine the symbols under the virtual heads ...

exercise: Finish this proof.

homework: Know this proof.

A **nondeterministic Turing machine** is like an ordinary Turing machine except, at any point in a computation, the machine may proceed according to several possibilities. If any branch leads to the accept state, the machine accepts its input.

Formally, the transition function appears as

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

exercise: Prove that the concatenation of two Turing-recognizable languages is recognizable by a nondeterministic Turing machine.

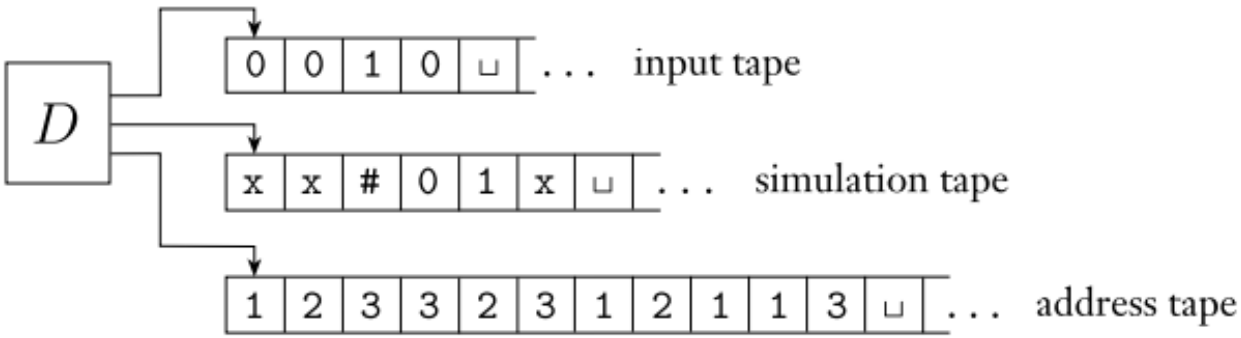
Theorem 3.16 (pg. 178)

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof. We can simulate any nondeterministic TM N with a deterministic TM D, where the simulation is to have D try all possible branches of N's computation. If D finds the accept state on any branch, D accepts; otherwise, D's simulation will not halt ... *(see next slide)*

Corollary 3.18 (pg. 180)

A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.



The deterministic simulation machine D has three tapes. The first tape is used to store the input string (and is never changed), the second tape is used to simulate N's tape in some branch, and the third tape stores the "address" of the state of computation in the tree of all possible branches of computation ...

exercise: Finish this proof.

homework: Know this proof.

We can modify the proof of Theorem 3.16 so that if N always halts on all branches of computation, D will always halt. *(see homework)*

Corollary 3.19 (pg. 180)

A language is decidable if and only if some nondeterministic Turing machine decides it.

We call a nondeterministic Turing machine a **decider** if all branches halt on all inputs.

homework: 3.3

Informal Definition (pg. 182)

An **algorithm** is a collection of simple instructions for carrying out some task.

The notion of algorithm itself was not defined precisely until the twentieth century. Before that, we used only an intuitive notion of what algorithms were.

Unfortunately, that was insufficient for gaining a deeper understanding of algorithms.

In 1936, Alonzo Church (λ-calculus) and Alan Turing (TMs) wrote papers demonstrating that their formal approaches were equivalent and were aligned with the informal notion of algorithm.

The connection between our intuitive notion of algorithm and the formal definitions is called the **Church-Turing thesis**.

i.e. When one asks if an algorithm exists to determine something, it is equivalent to ask if a TM exists that decides some particular language.

The standard ways of expressing TMs:

- **formal description** - Complete description of the machine's states, transition function, etc. *e.g.* diagrams and 7-tuples.
- **implementation description** - English prose to describe the way that the machine moves its head and stores data.
- **high-level description** - English prose to describe an algorithm, ignoring implementation details.

The input to a TM is a string. If we wish to provide an object other than a string, we can encode that object into a string. We denote an arbitrary (but fixed) encoding of object $\mathcal{O}$ as $\langle \mathcal{O} \rangle$. High-level descriptions assume a TM will first check for valid encoding.

exercise: Let G be an undirected graph. Come up with a reasonable encoding, $\langle G \rangle$, of G as a string.

exercise: Let A be the language of all strings representing undirected graphs that are connected. Provide a high-level description of a TM that decides A.

1. Select the first node of G and mark it.
2. For each node in G, mark it if ...
3. ...

exercise: Walk through *some* implementation details of the TM described above with respect to your encoding scheme.

homework: Repeat for directed, strongly-connected graphs.