1.4 Variants of Turing Machines

The definition of a Turing machine that we gave earlier in this lecture is by no means a canonical definition. When we chose to give our machine a deterministic transition function, or a single tape, or a two-way infinite tape, we made those choices simply to fix *some* definition of a Turing machine. Since we are just introducing Turing machines for the first time in this lecture, we decided to go with a relatively easy-to-understand definition: the single, two-way infinite tape allowed us to use our "two stacks" perspective to motivate the definition, and deterministic transition functions are more straightforward to reason about.

Just like we defined different types of finite automata, nothing is stopping us from modifying our definition of a Turing machine. However, one of the most remarkable results in theoretical computer science is that we can make pretty much *any* modification to our definition of a Turing machine, and it won't affect the recognition power of the model. The Turing machine is, in a sense, the Platonic form of a computer; nearly any variant definition we choose grants us sufficient power to recognize the large classes of decidable and semidecidable languages.

To illustrate, here we will make a number of modifications to our Turing machine definition, and we will then prove that each modified definition is equivalent to our original definition. This will give us a wide array of variant Turing machine models that we can choose from when we're trying to recognize certain languages.

Nondeterministic Turing Machines

The first natural modification we can make to our definition is to take the transition function δ to be nondeterministic. Just like with our other models of computation, a nondeterministic transition function for a Turing machine will map a pair of state and tape symbol to the power set of tuples of state, tape symbol, and input head movement.

Definition 9 (Nondeterministic Turing machine)**.** A nondeterministic Turing machine is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where everything is defined as in Definition 1 except for the transition function, which is

$$\delta : (Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

We saw that deterministic and nondeterministic finite automata are equivalent in terms of recognition power, while nondeterministic pushdown automata are able to recognize more languages than deterministic pushdown automata. With Turing machines, we return to equivalence: adding nondeterminism to a Turing machine does not give it more recognition power.

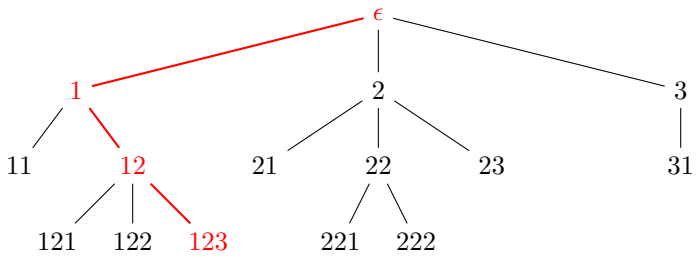
Theorem 10. *Given a nondeterministic Turing machine $\mathcal{M}$, we can convert it to a deterministic Turing machine $\mathcal{M}'$.*

Proof. Let $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be a nondeterministic Turing machine. We will construct a deterministic Turing machine $\mathcal{M}'$ that uses three tapes to simulate the nondeterministic computation of $\mathcal{M}$.

- The first tape will be called the *input tape*, and it will contain the input word given to $\mathcal{M}$. The contents of the input tape will not be changed during the computation.
- The second tape will be called the *simulation tape*, and it will simulate the contents of $\mathcal{M}$'s tape as $\mathcal{M}$ performs its nondeterministic computation.
- The third tape will be called the *address tape*, and it will keep track of where we are in the nondeterministic computation tree of $\mathcal{M}$.

Before we proceed, let's consider how we can represent the nondeterministic computation tree of $\mathcal{M}$ on a linear form of storage like a tape. If we take b to denote the maximum number of branches in the tree (that is, the maximum number of nondeterministic transitions $\mathcal{M}$ can follow at any given point in its computation), then we can assign a unique address to each vertex of the tree over the alphabet $\Sigma_b = \{1, 2, \ldots, b\}$. The address is determined by tracing the branches we must follow in order to get from the root to that vertex; for example, the vertex with address 123 can be reached by starting at the root of the tree and taking the first

branch, followed by the second branch, followed by the third branch. As a consequence of this convention, the root of the tree receives the address ϵ.



The address tape, then, contains symbols from the alphabet Σ_b. Each symbol on the address tape will tell $\mathcal{M}'$ which branch of the nondeterministic computation tree it must follow in its next computation step. Note that the contents of the address tape do not necessarily need to correspond to a vertex of the tree; if the address tape contains an invalid address, then $\mathcal{M}'$ simply aborts its attempted simulation of that branch.

Having defined the three tapes, we can describe how $\mathcal{M}'$ simulates the computation of $\mathcal{M}$:

1. Initialize the tapes in the following way:
   (a) Write to the input tape the input word w given to $\mathcal{M}$.
   (b) Leave both the simulation and address tapes empty.
2. Copy the contents of the input tape to the simulation tape.
3. Use the simulation tape to perform the following steps:
   (a) For each computation step of $\mathcal{M}$, read the next symbol of the address tape to determine which branch of the nondeterministic computation tree to follow.
   (b) If there are no more symbols remaining on the address tape, or if the address tape contains an invalid address, or if $\mathcal{M}$ enters a rejecting configuration, then abort the attempted simulation of this branch and go to step 4.
   (c) If $\mathcal{M}$ enters an accepting configuration, then accept w.
4. Write to the address tape the sequence of symbols over Σ_b that comes next in lexicographic order and go to step 2. □

Since every deterministic Turing machine is a nondeterministic Turing machine that doesn't use nondeterminism during its computation, we immediately obtain the other direction of the relationship between these models. Therefore, we can conclude that deterministic and nondeterministic Turing machines are equivalent.

Multitape Turing Machines

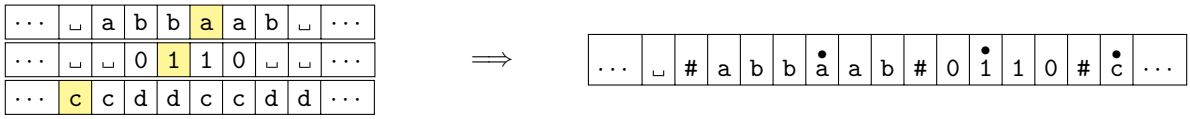
In the proof of Theorem 10, we saw that we could simulate the computation of a nondeterministic Turing machine using a deterministic Turing machine with multiple tapes. It's natural to wonder whether including additional tapes gave us some kind of advantage in this simulation process—after all, giving our computational model two stacks instead of one stack led us to the idea of a Turing machine—but as it turns out, we can perform exactly the same computations using only a single tape.

First, let's define our *multitape Turing machine* model. Again, we only need to modify the transition function: instead of mapping a pair of state and *one* tape symbol to a tuple of state, *one* tape symbol, and *one* input head movement, we will transition on k tape symbols and k input head movements, where k denotes the number of tapes used by the machine.

Definition 11 (k-tape Turing machine)**.** A k-tape Turing machine is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where everything is defined as in Definition 1 except for the transition function, which is

$$\delta : (Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k.$$

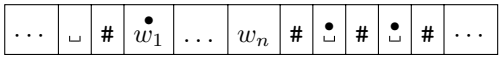
The idea allowing us to establish one direction of the equivalence between multitape and single-tape Turing machines is that we can simulate having many tapes T_i by storing all of the contents on a single tape T and separating "tape i" from the other "tapes" using a special symbol. Since our single tape has only one input head, we will also simulate the position of each input head on "tape i" using a special marker on the corresponding tape symbol to act as a virtual input head.



Theorem 12. *Given a k-tape Turing machine $\mathcal{M}$, we can convert it to a single-tape Turing machine $\mathcal{M}'$.*

Proof. Suppose the given Turing machine $\mathcal{M}$ has k tapes, and the tape alphabet is denoted by Γ. Our single-tape Turing machine $\mathcal{M}'$ will simulate $\mathcal{M}$'s computation on an input word $w = w_1 \dots w_n$ in the following way:

1. Take the tape alphabet of $\mathcal{M}'$ to be $\Gamma' = \Gamma \cup \dot{\Gamma} \cup \{\#\}$, where $\dot{\Gamma}$ consists of all alphabet symbols of Γ augmented with a dot and # is a special boundary marker.
2. Write the boundary marker # and the symbols of w to the tape of $\mathcal{M}'$, including the dotted symbol w_1. Then, write $k + 1$ copies of # each separated by a dotted blank space.



3. For each step of the computation, $\mathcal{M}'$ scans its entire tape from the first occurrence of # to the $(k+1)$st occurrence of # to read the symbols on all k tapes of $\mathcal{M}$. Then, $\mathcal{M}'$ makes a second pass along its tape to update any symbols that were changed by the transition function of $\mathcal{M}$. This update includes changing occurrences of dotted symbols in accordance with the changed positions of each virtual input head.

   If any of the virtual input heads of $\mathcal{M}'$ move onto an occurrence of #, then $\mathcal{M}$ must have moved the corresponding input head of that tape onto a blank space. In this case, $\mathcal{M}'$ writes a blank space to this cell of its tape and shifts the symbols of all subsequent cells rightward by one position. □

Naturally, a k-tape Turing machine can simulate the computation of a single-tape Turing machine by using only one of its k tapes. This again gives us the other direction of the relationship between the models, and again establishes the equivalence of the models.

One-Way Infinite Tape Turing Machines

In our definition of a Turing machine, we assumed that the tape used by the machine is *two-way infinite*; that is, there is an infinite number of cells to the left and to the right of the input head, and the input head can therefore move to an infinite number of positions of the tape in either direction.

We didn't have to define our storage in this way, though. We could have alternatively defined the tape to act more like the stack of a pushdown automaton: just like the stack has a fixed bottom boundary forcing us to push symbols only above that boundary, the tape could have a fixed left boundary forcing us to write symbols only to the right of that boundary. We call such a tape *one-way infinite*, since at any position along the tape, the input head has a finite number of cells to its left and an infinite number of cells to its right.

Because of the way the computation of the Turing machine is initialized, the initial position of the input head of a Turing machine with a one-way infinite tape will be on the first symbol of the input word, and

the left boundary of the tape will be to the input head's immediate left. In this cell, the input head cannot make a leftward move; if the transition function tells the input head to move left, then the input head will simply remain in the same cell.[3]

If we want to simulate the computation of a one-way infinite tape Turing machine using a two-way infinite tape Turing machine, then the required conversion seems straightforward: we just need to write a special symbol to one cell of the two-way infinite tape to act as the left boundary, and modify the transition function to handle the case where the input head moves onto this left boundary symbol.

Theorem 13. *Given a one-way infinite tape Turing machine $\mathcal{M}$, we can convert it to a two-way infinite tape Turing machine $\mathcal{M}'$.*

Proof. Suppose the given one-way infinite tape Turing machine $\mathcal{M}$ receives as input a word $w = w_1 \dots w_n$ and has a tape alphabet Γ. Our two-way infinite tape Turing machine $\mathcal{M}'$ will simulate $\mathcal{M}$'s computation on w in the following way:

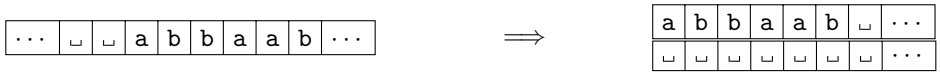
1. Take the tape alphabet of $\mathcal{M}'$ to be $\Gamma' = \Gamma \cup \{\vdash\}$, where $\vdash$ is a special left boundary marker.
2. Write the symbols of w to the tape of $\mathcal{M}'$, and write the left boundary marker $\vdash$ in the cell to the immediate left of the cell containing w_1.
3. Take the transition function of $\mathcal{M}'$ to be $\delta' = \delta$ and, for each state $q \in Q$, add a new transition $\delta'(q, \vdash) = (q, \vdash, R)$ to the transition function of $\mathcal{M}'$.
4. Create new states q'_{accept} and q'_{a}, and add new transitions to the transition function of $\mathcal{M}'$ as follows:
   - $\delta'(q_{\text{accept}}, \vdash) = (q'_{\text{accept}}, \vdash, R)$;
   - $\delta'(q_{\text{accept}}, c) = (q'_{\text{a}}, c, R)$ for all $c \in \Gamma' \setminus \{\vdash\}$; and
   - $\delta'(q'_{\text{a}}, d) = (q'_{\text{accept}}, d, L)$ for all $d \in \Gamma' \setminus \{\vdash\}$.

   Also, create new states q'_{reject} and q'_{r}, and add similar transitions on these states. Take the accepting and rejecting states of $\mathcal{M}'$ to be q'_{accept} and q'_{reject}, respectively. □

To obtain the other direction of the equivalence—that is, to simulate the computation of a two-way infinite tape Turing machine with a one-way infinite tape Turing machine—we can use our previous result establishing the equivalence of single-tape and multitape Turing machines.

While we won't go through the full proof here, the idea of the proof is to split the two-way infinite tape into a pair of one-way infinite tapes, where the "split point" occurs between the first symbol of the input word and the infinite blank spaces to the left of the input word. This action produces two one-way infinite tapes: one containing the input word, and one containing only blank spaces. Then, we modify the transition function of the one-way infinite tape Turing machine to operate on either the first or second tape, switching between the tapes each time the input head of the two-way infinite tape Turing machine crosses the "split point" on its tape.

Splitting the two-way infinite tape into a pair of one-way infinite tapes gives us the following setup:



Ultimately, this construction completes the proof and establishes the equivalence between one-way and two-way infinite tape Turing machines.

[3] Some authors alternatively assert that, if the input head of a one-way infinite tape Turing machine moves beyond the left boundary of the tape, then the machine "crashes" and the computation cannot continue.

2 Universal Turing Machines

Up to now, we have had to construct different specific Turing machines for each language we wished to recognize. In fact, we have had to construct specific machines for *every* language we wished to recognize in this course, whether that machine be a finite automaton, or a pushdown automaton, or indeed, a Turing machine.

However, we know that Turing machines are capable of performing quite complicated computations, and we know also that we can construct Turing machines that can simulate the computations of other variant Turing machines. What if we took this idea and generalized it as much as possible? That is, what if we constructed some Turing machine that could simulate the computation of *any* other Turing machine?

Alan Turing considered this exact idea in the paper that introduced the model of computation that would eventually be named after him. Turing described the process of constructing a machine $\mathcal{U}$ that is capable of simulating the computation of any other machine $\mathcal{M}$, so long as we give an appropriate encoding of $\mathcal{M}$ as part of the input to $\mathcal{U}$:

> "It is possible to invent a single machine which can be used to compute any computable sequence. If this machine $\mathcal{U}$ is supplied with a tape on the beginning of which is written the S.D. [standard description] of some computing machine $\mathcal{M}$, then $\mathcal{U}$ will compute the same sequence as $\mathcal{M}$."
> — Alan Turing, *On Computable Numbers, with an Application to the Entscheidungsproblem* Proceedings of the London Mathematical Society, Series 2, 42(1), 1937.

Here, like Turing did before us, we will show how to construct such a machine $\mathcal{U}$, which is nowadays called a *universal Turing machine*.[4] The main benefit of having such a machine is that we will no longer have to construct specific Turing machines for each language we consider; now, we can just give a high-level description of the Turing machine's computation, and we can feasibly "program" the universal Turing machine to perform its computation in a similar way. These high-level descriptions will be quite similar to what we saw in the proofs of Theorems 10, 12, and 13, where we simply listed the steps of the machine's computation instead of explicitly writing out each component of the machine.

Suppose that the input we give to our universal Turing machine $\mathcal{U}$ is of the form $\langle \mathcal{M}, w \rangle$, where $\mathcal{M}$ is an encoding of the Turing machine we wish to simulate and w is the input word given to $\mathcal{M}$. Given an input of this form, our machine $\mathcal{U}$ must satisfy three criteria:

1. $\mathcal{U}$ halts its computation on input $\langle \mathcal{M}, w \rangle$ if and only if $\mathcal{M}$ halts its computation on input w;
2. $\mathcal{U}$ enters its accepting state if and only if $\mathcal{M}$ enters its accepting state; and
3. $\mathcal{U}$ enters its rejecting state if and only if $\mathcal{M}$ enters its rejecting state.

We now go through with the construction of this machine $\mathcal{U}$.

Theorem 14. *There exists a universal Turing machine $\mathcal{U}$ that, given an input $\langle \mathcal{M}, w \rangle$, is capable of simulating the computation of a Turing machine $\mathcal{M}$ on an input word w.*

Proof. We will construct $\mathcal{U}$ in the form of a multitape Turing machine, just as we did in our procedure to determinise a nondeterministic Turing machine.

For this construction, we need only three tapes:

- The first tape will initially contain the input $\langle \mathcal{M}, w \rangle$, and after the computation of $\mathcal{U}$ begins, it will simulate the contents of the tape of $\mathcal{M}$.
- The second tape will contain the encoding of the machine $\mathcal{M}$.
- The third tape will keep track of which state we are currently in during the computation of $\mathcal{M}$ by maintaining the current state as a binary number.

[4] It's important to note that, in this context, "universal" does not mean that the Turing machine $\mathcal{U}$ can compute *everything*. It only means that $\mathcal{U}$ can compute whatever other Turing machines can compute.