Theory of Computation

Ryan Chao, Yakir Propp

PRIMES Circle

2023

Intro to State Machines

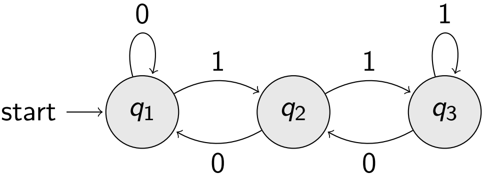
Have you ever been in an elevator before?

Intro to State Machines

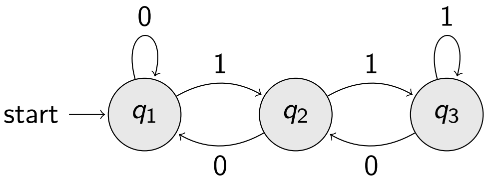
Here is an elevator in a building with three floors.

Intro to State Machines

Here is an elevator in a building with three floors.

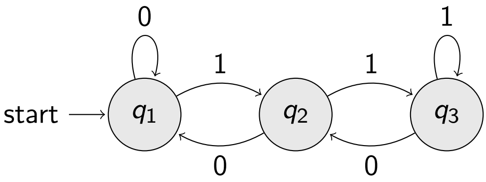


Here is an elevator in a building with three floors.



What is that?

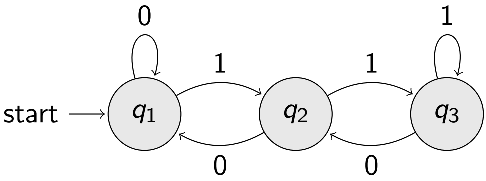
Here is an elevator in a building with three floors.



What is that?

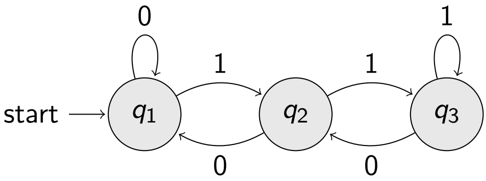
This is a state diagram for an elevator in a building with three floors. q_1, q_2, and q_3, which are called *states* represent the first second and third floors of the building, respectively.

State Machine informal definition



This state diagram exhibits all required features of all state diagrams.

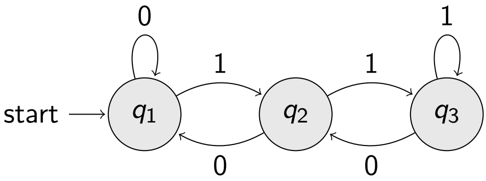
State Machine informal definition



This state diagram exhibits all required features of all state diagrams.

1. Every state diagram has a **start state** which sends off an arrow into our first state, the initial state, q_1, which is in this case, the ground floor.

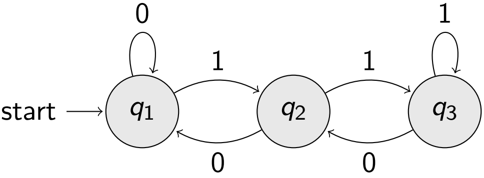
State Machine informal definition



This state diagram exhibits all required features of all state diagrams.

1. Every state diagram has a **start state** which sends off an arrow into our first state, the initial state, q_1, which is in this case, the ground floor.
2. The state diagram has an **alphabet**, which is a set of symbols which are the options for each arrow. In this case, the alphabet is $0, 1$, because we can only go up or down.

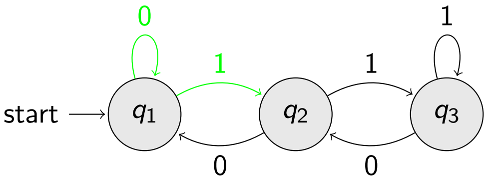
State Machine informal definition



This state diagram exhibits all required features of all state diagrams.

1. Every state diagram has a **start state** which sends off an arrow into our first state, the initial state, q_1, which is in this case, the ground floor.
2. The state diagram has an **alphabet**, which is a set of symbols which are the options for each arrow. In this case, the alphabet is $0, 1$, because we can only go up or down.
3. Each state has exactly one arrow for each symbol of the alphabet coming off from it.

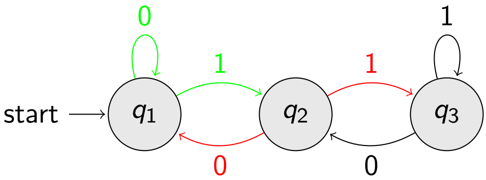
State Machine informal definition



This state diagram exhibits all required features of all state diagrams.

1. Every state diagram has a **start state** which sends off an arrow into our first state, the initial state, q_1, which is in this case, the ground floor.
2. The state diagram has an **alphabet**, which is a set of symbols which are the options for each arrow. In this case, the alphabet is $0, 1$, because we can only go up or down.
3. Each state has exactly one arrow for each symbol of the alphabet coming off from it.

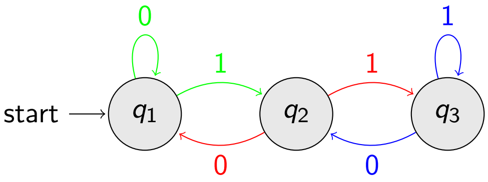
State Machine informal definition



This state diagram exhibits all required features of all state diagrams.

1. Every state diagram has a **start state** which sends off an arrow into our first state, the initial state, q_1, which is in this case, the ground floor.
2. The state diagram has an **alphabet**, which is a set of symbols which are the options for each arrow. In this case, the alphabet is $0, 1$, because we can only go up or down.
3. Each state has exactly one arrow for each symbol of the alphabet coming off from it.

State Machine informal definition



This state diagram exhibits all required features of all state diagrams.

1. Every state diagram has a **start state** which sends off an arrow into our first state, the initial state, q_1, which is in this case, the ground floor.
2. The state diagram has an **alphabet**, which is a set of symbols which are the options for each arrow. In this case, the alphabet is $0, 1$, because we can only go up or down.
3. Each state has exactly one arrow for each symbol of the alphabet, coming off from it.

Input Strings and Accept States

If you were to walk into the elevator and decide to go up, down, up, down, down, and then up, the state diagram would tell you where you would end up after all of that. If you follow the steps in your head, you will end up on the second floor (remember, there is no floor -1).

Input Strings and Accept States

If you were to walk into the elevator and decide to go up, down, up, down, down, and then up, the state diagram would tell you where you would end up after all of that. If you follow the steps in your head, you will end up on the second floor (remember, there is no floor -1).

By the way, not recommended. If you want exercise, just take the stairs!

Input Strings and Accept States

If you were to walk into the elevator and decide to go up, down, up, down, down, and then up, the state diagram would tell you where you would end up after all of that. If you follow the steps in your head, you will end up on the second floor (remember, there is no floor -1).

Input Strings and Accept States

If you were to walk into the elevator and decide to go up, down, up, down, down, and then up, the state diagram would tell you where you would end up after all of that. If you follow the steps in your head, you will end up on the second floor (remember, there is no floor -1). Converting "down" into 0, and "up" into 1, you follow the sequence: 101001.

Input Strings and Accept States

Also important to know, an empty string is denoted as ϵ. It is just a blank character. For example, 101001 and $101\epsilon00\epsilon1\epsilon$ are the same.

Input Strings and Accept States

Also important to know, an empty string is denoted as ϵ. It is just a blank character. For example, 101001 and $101\epsilon00\epsilon1\epsilon$ are the same.

Hopefully you can now see how this elevator example and state machines relate to computer science. They work by parsing in input strings to find the output in the state machine.

Input Strings and Accept States

Also important to know, an empty string is denoted as ϵ. It is just a blank character. For example, 101001 and $101\epsilon00\epsilon1\epsilon$ are the same.

Hopefully you can now see how this elevator example and state machines relate to computer science. They work by parsing in input strings to find the output in the state machine. This brings of to our fourth property of state machines.

Input Strings and Accept States

Also important to know, an empty string is denoted as ϵ. It is just a blank character. For example, 101001 and $101\epsilon00\epsilon1\epsilon$ are the same.

Hopefully you can now see how this elevator example and state machines relate to computer science. They work by parsing in input strings to find the output in the state machine. This brings of to our fourth property of state machines.

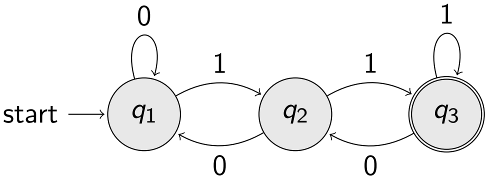
4. In the theory of computation, we want to find if the state machine *rejects* or *accepts* a string. The machine will *accepts* a string if it ends at an accept state, and reject if it does not. Any of the states can be accept states.

Input Strings and Accept States

Here is the state diagram with an accept state

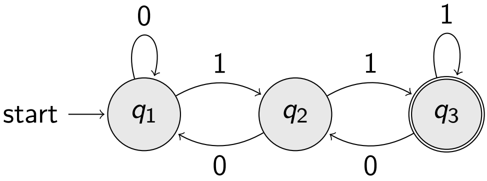
Input Strings and Accept States

Here is the state diagram with an accept state



Input Strings and Accept States

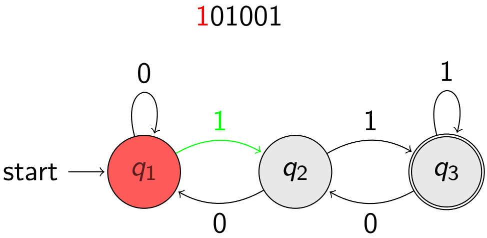
Here is the state diagram with an accept state



Say we really want to know if we end up on the third floor after all of our movement. This state machine will accept a string if it lands us at state 3.

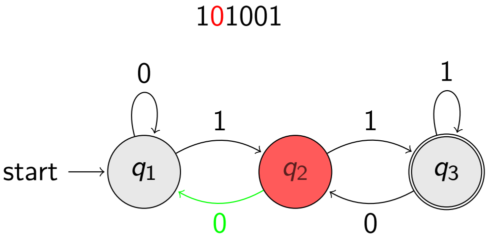
Input Strings and Accept States

Let's see what this now complete state machine does with our previous input: 101001:



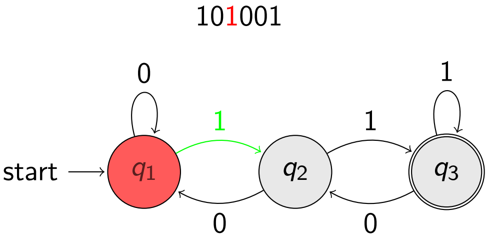
Inputs Strings and Accept States

Let's see what this now complete state machine does with our previous input: 101001:



Input Strings and Accept States

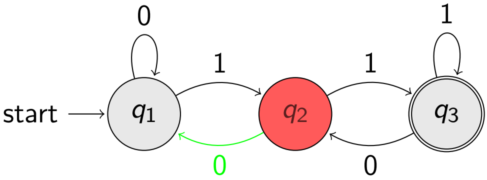
Let's see what this now complete state machine does with our previous input: 101001:



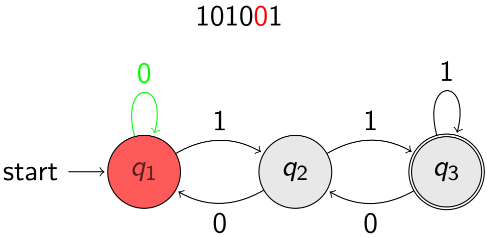
Input Strings and Accept States

Let's see what this now complete state machine does with our previous input: 101001:

101001



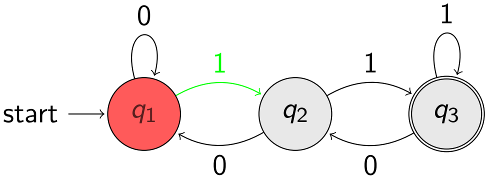
Let's see what this now complete state machine does with our previous input: 101001:



Input Strings and Accept States

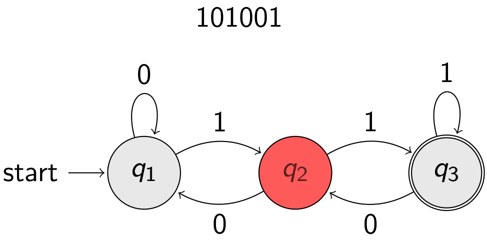
Let's see what this now complete state machine does with our previous input: 101001:

101001



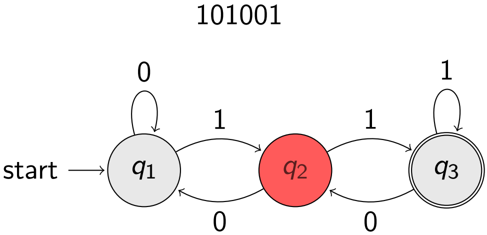
Input Strings and Accept States

Let's see what this now complete state machine does with our previous input: 101001:



Input Strings and Accept States

Let's see what this now complete state machine does with our previous input: 101001:



As you can see, the string in our example: 101001, will not be accepted, because it ends at state 2, which is not an accept state.

Languages

Now that we've introduced accept states, let us present the two following definitions.

Definition

The **Language** of a state machine the set of strings that it accepts.

Definition

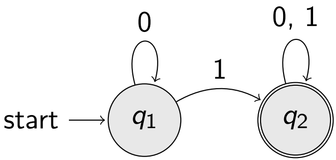
A state machine "recognizes" a language if the set of strings that it accepts is *exactly* equal to the language.

Definition

A set of strings which recognized by *some* DFA is called a **regular language**.

Example

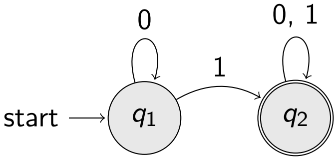
Take this state machine for example:



Can you figure out what the language of this state machine is?

Example

Take this state machine for example:



Can you figure out what the language of this state machine is?

It is simply all strings that contain a 1 in them.

1. If a 1 is detected by the machine, it moves to state 2, and stays there, since, at state 2, it loops for both 0 and 1.
2. If there is not a 1 in the string, the machine will loop at state 1, and not accept.

Vocab Debrief

We've been using the word state machine to describe these diagrams so far, because they are a machine of states.

Vocab Debrief

We've been using the word state machine to describe these diagrams so far, because they are a machine of states.

- What you have seen with the elevator and the most recent example are examples of **Automata**.

Vocab Debrief

We've been using the word state machine to describe these diagrams so far, because they are a machine of states.

- What you have seen with the elevator and the most recent example are examples of **Automata**. **Automata** are simply models for machines. They take in strings an accept or reject.
- The previous examples are all **Finite Automata**, since they do not possess an infinite memory.

Vocab Debrief

We've been using the word state machine to describe these diagrams so far, because they are a machine of states.

- What you have seen with the elevator and the most recent example are examples of **Automata**. **Automata** are simply models for machines. They take in strings an accept or reject.
- The previous examples are all **Finite Automata**, since they do not possess an infinite memory. In fact, **Finite Automata** do not have any memory, but you will see that some **Automata** do.

Vocab Debrief

We've been using the word state machine to describe these diagrams so far, because they are a machine of states.

- What you have seen with the elevator and the most recent example are examples of **Automata**. **Automata** are simply models for machines. They take in strings an accept or reject.
- The previous examples are all **Finite Automata**, since they do not possess an infinite memory. In fact, **Finite Automata** do not have any memory, but you will see that some **Automata** do.

There was a slight lie told earlier. Item #3 for state diagrams does not always hold:

Vocab Debrief

We've been using the word state machine to describe these diagrams so far, because they are a machine of states.

- What you have seen with the elevator and the most recent example are examples of **Automata**. **Automata** are simply models for machines. They take in strings an accept or reject.
- The previous examples are all **Finite Automata**, since they do not possess an infinite memory. In fact, **Finite Automata** do not have any memory, but you will see that some **Automata** do.

There was a slight lie told earlier. Item #3 for state diagrams does not always hold:

3. Each state has exactly one arrow for each symbol of the alphabet, coming off from it.

Vocab Debrief

We've been using the word state machine to describe these diagrams so far, because they are a machine of states.

- What you have seen with the elevator and the most recent example are examples of **Automata**. **Automata** are simply models for machines. They take in strings an accept or reject.
- The previous examples are all **Finite Automata**, since they do not possess an infinite memory. In fact, **Finite Automata** do not have any memory, but you will see that some **Automata** do.

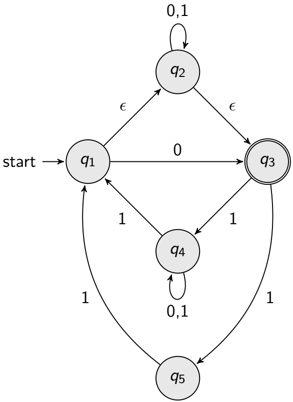
There was a slight lie told earlier. Item #3 for state diagrams does not always hold:

3. Each state has exactly one arrow for each symbol of the alphabet, coming off from it.

What you have just seen are **Deterministic Finite Automata**. They are called deterministic this third rule makes the state machine take a set path for any input.

NFA Example

The other type of **Finite Automata** is the Nondeterministic finite automata (NFA). Here is an example of the an NFA:



NFA informal definition

An NFA is like a DFA with one key difference:

NFA informal definition

An NFA is like a DFA with one key difference:

3. - Each state has any number of arrows for each symbol of the alphabet, coming off from it.

NFA informal definition

An NFA is like a DFA with one key difference:

3. - Each state has any number of arrows for each symbol of the alphabet, coming off from it.
   - At any stage in computation, meaning at any point in the string, we can choose to traverse any one of the arrows with the corresponding symbol. If any of the branches of computation end at an accept state, the NFA accepts.

NFA informal definition

An NFA is like a DFA with one key difference:

3. - Each state has any number of arrows for each symbol of the alphabet, coming off from it.
   - At any stage in computation, meaning at any point in the string, we can choose to traverse any one of the arrows with the corresponding symbol. If any of the branches of computation end at an accept state, the NFA accepts.
   - We are also allowed a new type of arrow called an ϵ arrow. The way that this arrow works is it transitions without taking in any input symbol, or in other words, taking in the empty symbol.

NFA informal definition

An NFA is like a DFA with one key difference:

3.
- Each state has any number of arrows for each symbol of the alphabet, coming off from it.
- At any stage in computation, meaning at any point in the string, we can choose to traverse any one of the arrows with the corresponding symbol. If any of the branches of computation end at an accept state, the NFA accepts.
- We are also allowed a new type of arrow called an ϵ arrow. The way that this arrow works is it transitions without taking in any input symbol, or in other words, taking in the empty symbol.
- Another way to think about it is that for any transition, the NFA goes through all available transition arrows. The ends of these arrows branch out similarly. If any of the final states is an accept state, the NFA accepts.

DFA vs. NFA

NFAs seem much more powerful than their counterpart, because of all of the extra possibilities allowed. Yet, it can be proven that any NFA has an equivalent DFA, or in other words, any language that is recognized by some NFA can also be recognized by some NFA.

DFA/NFA Equivalence

Here is a rough sketch of this proof:
The NFA can be viewed as a *deterministic* machine from sets of states. At any point in computation, the machine is in a set of states, and from each of these states, after undergoing every transition, we end up in another set.

DFA/NFA Equivalence

Here is a rough sketch of this proof:
The NFA can be viewed as a *deterministic* machine from sets of states. At any point in computation, the machine is in a set of states, and from each of these states, after undergoing every transition, we end up in another set.

From an NFA, N, we can create DFA D, where D's states are the sets of the states in N, and D's accept states are any set which contains an accept state of N.

Introducing Turing Machine

Do you use a computer?

Introducing Turing Machine

Do you use a computer? Well then you're in luck! I sure do!

Introducing Turing Machine

Do you use a computer? Well then you're in luck! I sure do! Turing machines happen to be the first version of modern day computers! Here's what the computers which you now know and love would have looked like all those years ago (all the way back in 1936)

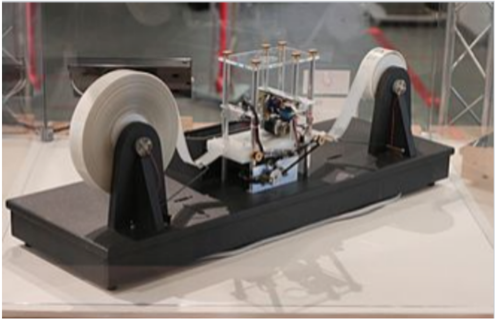


Figure: Turing Machine in 1936

Church-Turing Thesis

Church-Turing Thesis

Any real-world computation can be translated into an equivalent problem involving a Turing Machine

Church-Turing Thesis

Church-Turing Thesis

Any real-world computation can be translated into an equivalent problem involving a Turing Machine

Any computation you could do, a Turing Machine could as well!

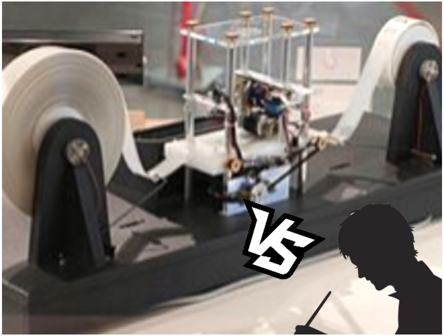


Figure: You vs a Turing Machine

Turing Machine Definition

So what actually are these things called Turing Machines? Turing Machines consist of a few main elements:

Definition

A Turing Machine...

1. Has an infinite "**tape**" which stores symbols sequentially

Turing Machine Definition

So what actually are these things called Turing Machines? Turing Machines consist of a few main elements:

Definition

A Turing Machine...

1. Has an infinite "**tape**" which stores symbols sequentially
2. Begins by holding just the input string

Turing Machine Definition

So what actually are these things called Turing Machines? Turing Machines consist of a few main elements:

Definition

A Turing Machine...

1. Has an infinite "**tape**" which stores symbols sequentially
2. Begins by holding just the input string
3. Has a **tape head** which can move either left or right one space at a time

Turing Machine Definition

So what actually are these things called Turing Machines? Turing Machines consist of a few main elements:

Definition

A Turing Machine...

1. Has an infinite "**tape**" which stores symbols sequentially
2. Begins by holding just the input string
3. Has a **tape head** which can move either left or right one space at a time
4. Has **transition functions** that decide where the tape position is

Turing Machine Definition

So what actually are these things called Turing Machines? Turing Machines consist of a few main elements:

Definition

A Turing Machine...

1. Has an infinite "**tape**" which stores symbols sequentially
2. Begins by holding just the input string
3. Has a **tape head** which can move either left or right one space at a time
4. Has **transition functions** that decide where the tape position is
5. During a transition can add or remove a symbol in current location

Turing Machine Definition

So what actually are these things called Turing Machines? Turing Machines consist of a few main elements:

Definition

A Turing Machine...

1. Has an infinite "**tape**" which stores symbols sequentially
2. Begins by holding just the input string
3. Has a **tape head** which can move either left or right one space at a time
4. Has **transition functions** that decide where the tape position is
5. During a transition can add or remove a symbol in current location
6. Has only one accept state and reject state

A Closer Look

Let's take a closer look!

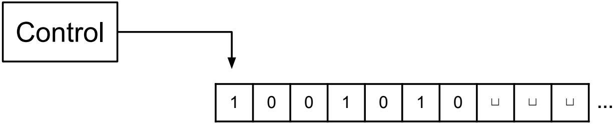


Figure: Example Turing Machine

A Closer Look

Let's take a closer look!

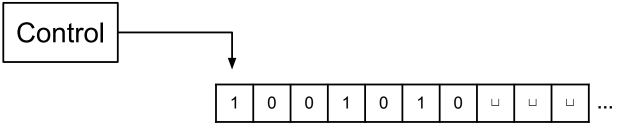


Figure: Example Turing Machine

The Control block symbolizes where the transition functions and states are held, while the arrow represents the tape head, pointing to which character of the input string it is at. Finally, we can see the tape holding the string "1001010," the $\sqcup$ symbol telling us that there is an empty space.

Note the "..." is showing us that the tape is infinite

Example

Example

Find if input has equal number of 1's and 0's

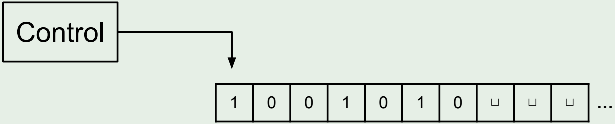


Figure: Example Turing Machine

1. Scan input left to right

Example

Example

Find if input has equal number of 1's and 0's

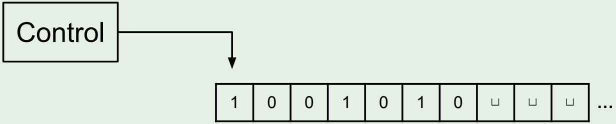


Figure: Example Turing Machine

1. Scan input left to right
2. Replace the first 1 with an "x" and the first 0 with an "x"

Example

Example

Find if input has equal number of 1's and 0's

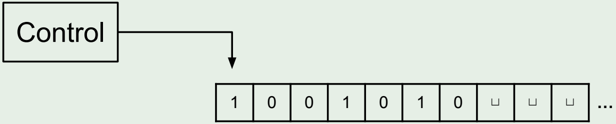


Figure: Example Turing Machine

1. Scan input left to right
2. Replace the first 1 with an "x" and the first 0 with an "x"
3. If the entire string is full of x's, accept

Example

Example

Find if input has equal number of 1's and 0's

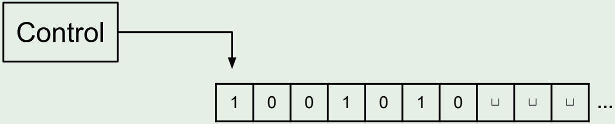


Figure: Example Turing Machine

1. Scan input left to right
2. Replace the first 1 with an "x" and the first 0 with an "x"
3. If the entire string is full of x's, accept
4. If all non-x symbols are the same, reject

Example

Example

Find if input has equal number of 1's and 0's

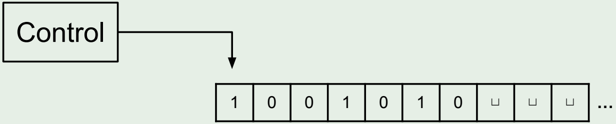


Figure: Example Turing Machine

1. Scan input left to right
2. Replace the first 1 with an "x" and the first 0 with an "x"
3. If the entire string is full of x's, accept
4. If all non-x symbols are the same, reject
5. Return the head to the left end of the tape

Example

Example

Find if input has equal number of 1's and 0's

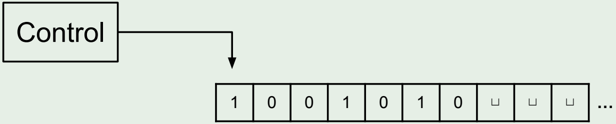


Figure: Example Turing Machine

1. Scan input left to right
2. Replace the first 1 with an "x" and the first 0 with an "x"
3. If the entire string is full of x's, accept
4. If all non-x symbols are the same, reject
5. Return the head to the left end of the tape
6. Go to stage 1

Example

Let's see what would happen after the first pass of our algorithm

Example

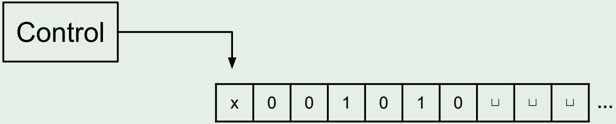


Figure: Example Turing Machine

Example

Let's see what would happen after the first pass of our algorithm

Example

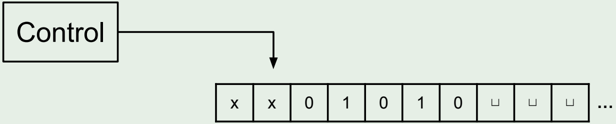


Figure: Example Turing Machine

Example

Let's see what would happen after the first pass of our algorithm

Example

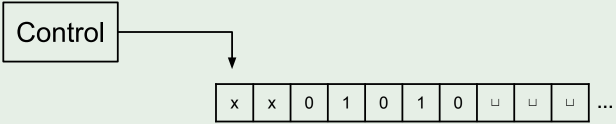


Figure: Example Turing Machine

Example

Let's see what would happen after the second pass of our algorithm

Example

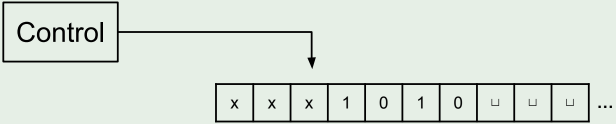


Figure: Example Turing Machine

Example

Let's see what would happen after the second pass of our algorithm

Example

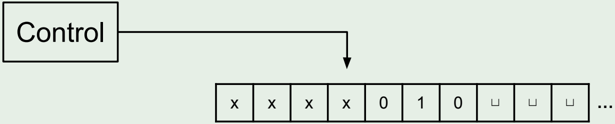


Figure: Example Turing Machine

Example

Let’s see what would happen after the second pass of our algorithm

Example

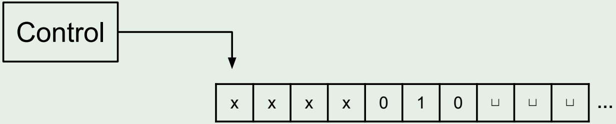


Figure: Example Turing Machine

Example

Let's see what would happen after the third pass of our algorithm

Example

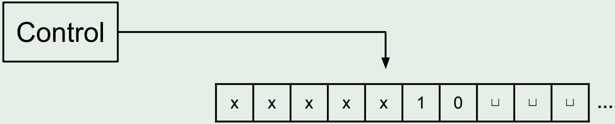


Figure: Example Turing Machine

Example

Let's see what would happen after the third pass of our algorithm

Example

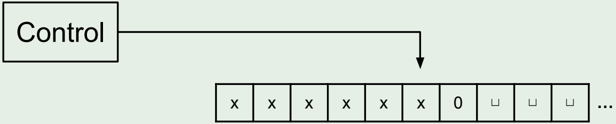


Figure: Example Turing Machine

Example

Let's see what would happen after the third pass of our algorithm

Example

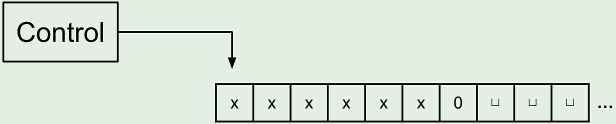


Figure: Example Turing Machine

Example

We see that at the end once we have filled out all the x's we possibly could we are left with only one non-x symbol, a 0, so we reject the string "1001010"

Example

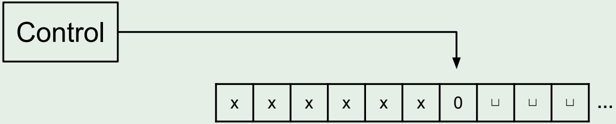


Figure: Example Turing Machine

Varients

What's next?

Varients

What's next?

There are many variants of Turing Machines, the ones which we will talk about today being the **Multitape Turing machine** and **Nondeterministic Turing Machine**. So far, we have only seen **Deterministic Turing Machines**.

Multitape Turing Machine

Definition

A Multitape Turing Machine...

1. Exactly like Deterministic Turing Machine except...

Multitape Turing Machine

Definition

A Multitape Turing Machine...

1. Exactly like Deterministic Turing Machine except...
2. It has multiple tapes that work simultaneously!

Multitape Turing Machine

Definition

A Multitape Turing Machine...

1. Exactly like Deterministic Turing Machine except...
2. It has multiple tapes that work simultaneously!

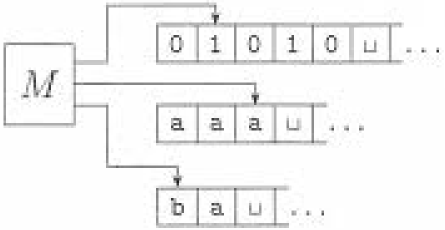


Figure: Multitape TM M

Theorem

Theorem

A Deterministic Turing Machine is equivalent to a Multitape Turing Machine

Theorem

A Deterministic Turing Machine is equivalent to a Multitape Turing Machine

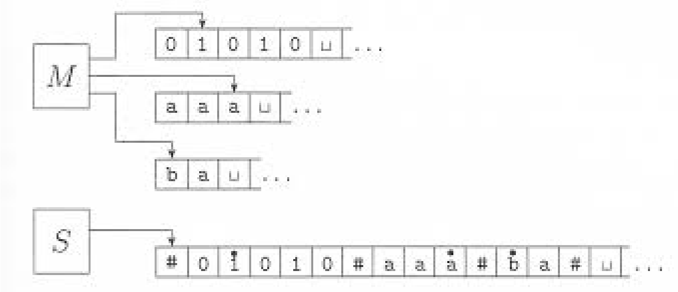


Figure: Multitape TM M and Deterministic TM S simulating it

Nondeterministic Turing Machine

Definition

A Nondeterministic Turing Machine...

1. Exactly like Deterministic Turing Machine except...

Nondeterministic Turing Machine

Definition

A Nondeterministic Turing Machine...

1. Exactly like Deterministic Turing Machine except...
2. Introduces Nondeterminism, can move to any of the states available

Nondeterministic Turing Machine

Definition

A Nondeterministic Turing Machine...

1. Exactly like Deterministic Turing Machine except...
2. Introduces Nondeterminism, can move to any of the states available
3. Tape head has many possibilities for where it can go next

Nondeterministic Turing Machine

Definition

A Nondeterministic Turing Machine...

1. Exactly like Deterministic Turing Machine except...
2. Introduces Nondeterminism, can move to any of the states available
3. Tape head has many possibilities for where it can go next

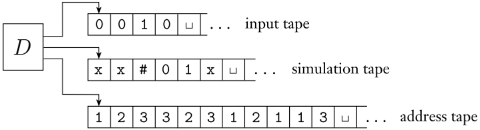


Figure: Nondeterministic TM being simulated on a Multitape TM

Nondeterministic Turing Machine

Definition

A Nondeterministic Turing Machine...

1. Exactly like Deterministic Turing Machine except...
2. Introduces Nondeterminism, can move to any of the states available
3. Tape head has many possibilities for where it can go next

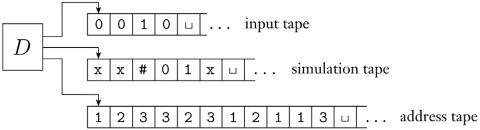


Figure: Nondeterministic TM being simulated on a Multitape TM

Deterministic TM = Multitape TM = Nondeterministic TM

Similarities

So what is the relationship between Turing Machines and Finite Automata?

1. Like a Finite Automata it either accepts a String or rejects a String, it has states, transition functions (the fundamentals of a Finite Automata)

Similarities

So what is the relationship between Turing Machines and Finite Automata?

1. Like a Finite Automata it either accepts a String or rejects a String, it has states, transition functions (the fundamentals of a Finite Automata)
2. Saw Determinism and Nondeterminism and the equivalence between those two models

Similarities

So what is the relationship between Turing Machines and Finite Automata?

1. Like a Finite Automata it either accepts a String or rejects a String, it has states, transition functions (the fundamentals of a Finite Automata)
2. Saw Determinism and Nondeterminism and the equivalence between those two models
3. Main difference: a Turing Machine has an infinite input tape, whereas a Finite Automata has a finite one

Similarities

So what is the relationship between Turing Machines and Finite Automata?

1. Like a Finite Automata it either accepts a String or rejects a String, it has states, transition functions (the fundamentals of a Finite Automata)
2. Saw Determinism and Nondeterminism and the equivalence between those two models
3. Main difference: a Turing Machine has an infinite input tape, whereas a Finite Automata has a finite one

Ultimately for this reason Turing Machines are a lot more powerful than their finite counterparts.

Hierarchy

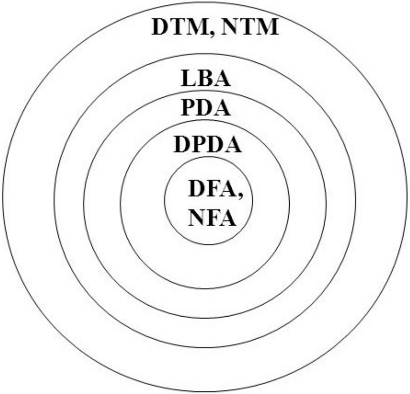


Figure: Hierarchy of Automata

Thank you!

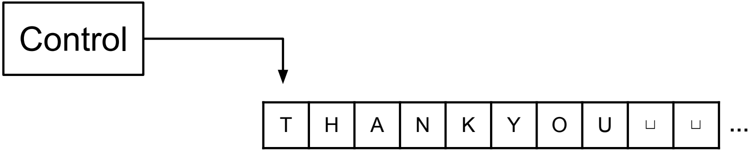


Figure: Thank you!