Tutorial Week 2

Turing Machines

Tutorial Objectives

- Introduce the Turing Machine
- Learn the formal definition of a Turing Machine
- Learn terms and apply when talking about Turing Machines, such as:
  - Halt, Loop, Accept, Reject, Language, Alphabet, Blank, Input, Output, States, the Tape, the Head
- Go over examples of Turing Machines
  - Learn how to describe a Turing Machine to solve a problem
  - Reason about its behaviour

Limitations of our Computers

We would like to talk about what computers can and cannot do, to understand computability more mechanically.

What are some issues about using the modern Computer?

- Limited memory
- Limited power
- It can be non-deterministic at times; race conditions
- More than one way to create a modern computer.

We'll want a more formal and universal "computer"

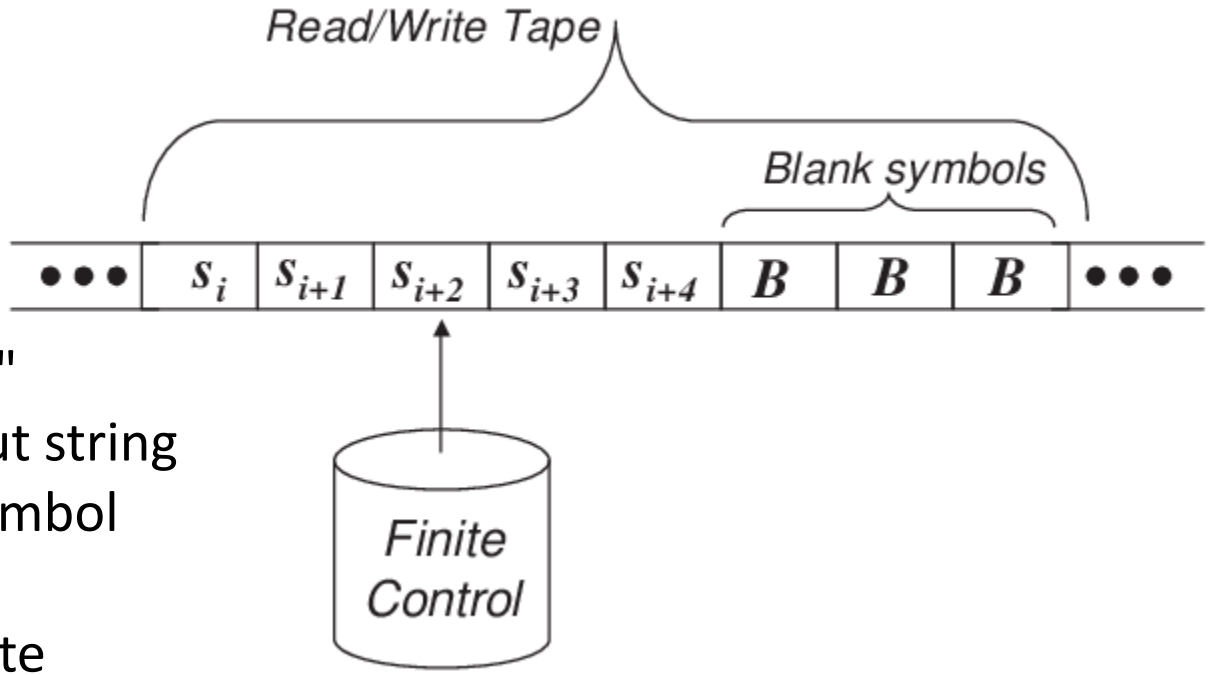
Turing Machine -- Informally

- First proposed by Alan Turing in 1936 as a powerful abstract model of computation
- Key features
  - It has unlimited memory
  - It can read and write to memory
  - It has an unbounded (yet finite) amount of states
  - Input and output strings
- It is an accurate model of a general-purpose computer

Turing Machine -- Informally

Setup – before any computation:

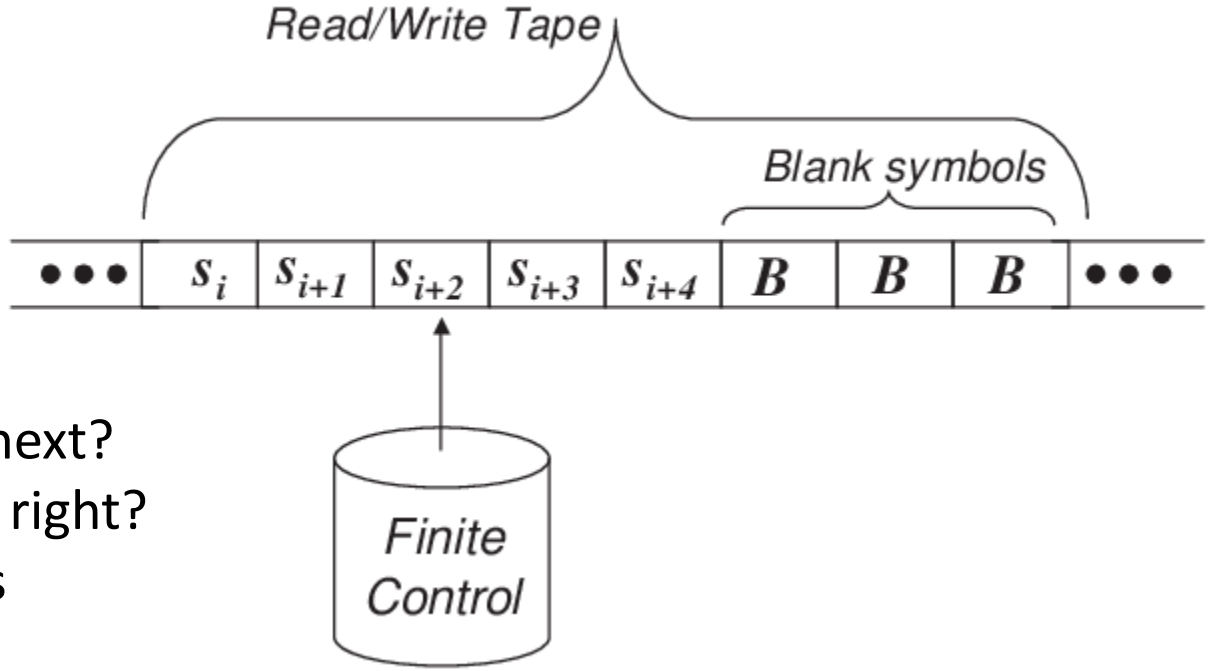
- The Turing Machine is given an input string
- The input is written to the tape letter by letter
  - So that each letter occupies one cell
- The read/write head is pointing to the "first cell"
  - The cell which has the first letter of the input string
- Any tape cell that is not occupied by an input symbol will be empty by default – the blank symbol □
- Finally, the Turing Machine is set to its initial state
  - These states govern the machine's behavior, it must begin at some state to begin computation.



Turing Machine -- Informally

During Computation:

- The head will read in the cell's symbol.
- The machine – using its internal state AND the symbol it has read – will make three decisions:
  - What should I write to the current cell?
  - What internal state should I transition into next?
  - Should I move the head to the left or to the right?
- After all decisions are done, the process repeats indefinitely!

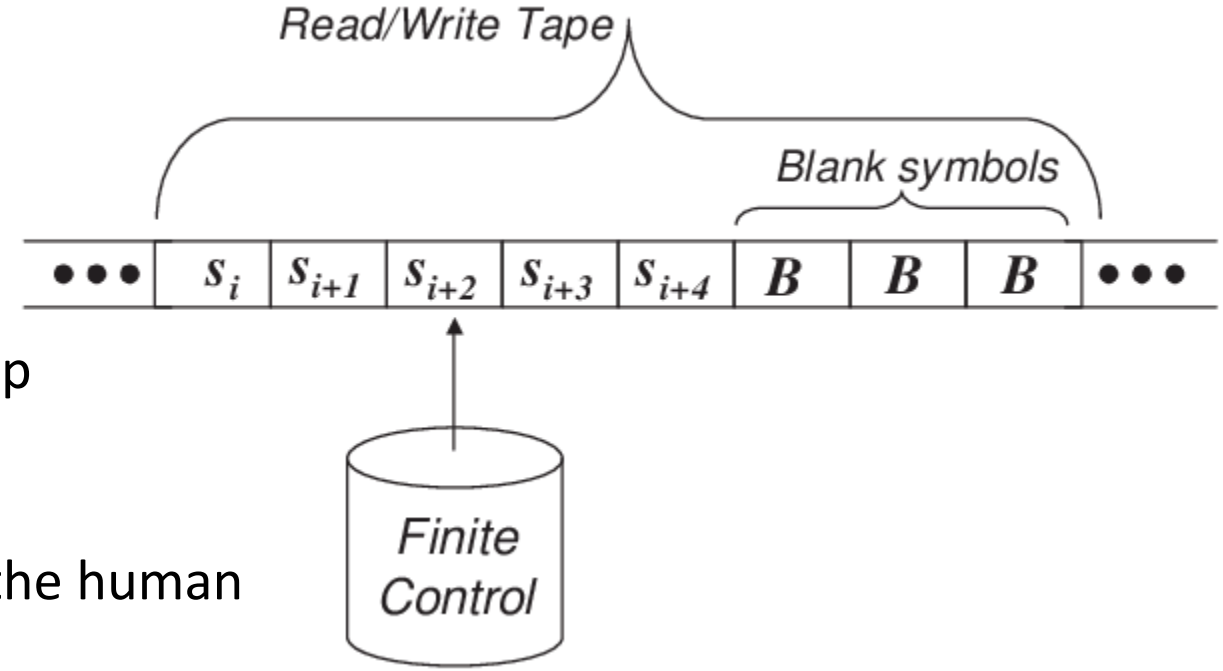


So far it will just run infinitely, how do we terminate the Turing Machine?

Turing Machine -- Informally

Stop the computation:

- At any point in its computation, the Turing Machine may choose to transition to a special ***HALTING*** state.
- In this special state, the Turing Machine will stop reading symbols from the tape which stops the computation cycle described in the prev. slide
- The output can then be read from the tape by the human

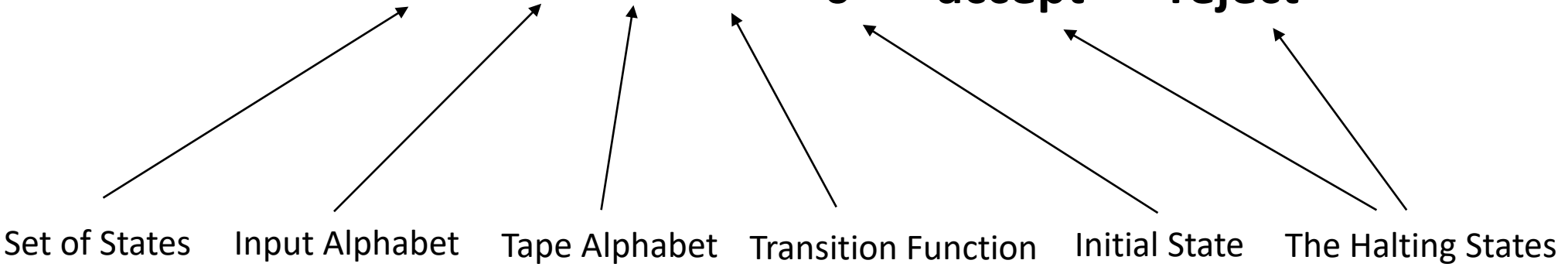


To program a Turing Machine, we will need to learn its formal definition

Turing Machines -- Formally

We won't dwell on the formal definition of the Turing Machine much, but we will briefly go over it. We present the 7-tuple formal definition.

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$



Example 1

We'll do these together:

- Describe the output of M if we give it the string "CSC363".
- Describe the output of M if we give it the string "CSC3Z3"
- Describe the output of M if we give it the string "CCC3Z3"
- Describe the output of M if we give it the empty string

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where:

- $Q = \{q_0, q_1, q_{accept}, q_{reject}\}$
- $\Sigma = \{C, S, Z, 3, 6\}$
- $\Gamma = \{C, S, Z, 0, 3, 6, \square\}$
- δ is defined by the table:

(q_0, C)	$(q_0, 3, R)$
(q_0, S)	$(q_0, 6, R)$
(q_0, Z)	$(q_1, 0, L)$
$(q_0, 0)$	$(q_0, 0, R)$
$(q_0, 3)$	(q_0, C, R)
$(q_0, 6)$	(q_0, S, R)
$(q_0, \square)$	$(q_{accept}, \square, L)$
(q_1, C)	$(q_1, 0, L)$
(q_1, S)	$(q_1, 0, L)$
(q_1, Z)	$(q_1, 0, L)$
$(q_1, 0)$	$(q_1, 0, L)$
$(q_1, 3)$	$(q_1, 0, L)$
$(q_1, 6)$	$(q_0, 6, R)$
$(q_1, \square)$	$(q_1, \square, R)$

Input ‘CSC363’

We'll do these together:

- Start at C and state q0. Write 3, stay on state q0, and move Right.
  3SC363
- Read S in state q0, write 6, stay in q0, and move Right.
  36C363
- Read C in state q0, write 3, stay in q0, and move Right.
  363363
- Read 3 in state q0, write C, stay in q0, move Right.
  363C63
- Read 6 in state q0, write S, stay in q0, move Right
  363CS3
- Read 3 in state q0, write C, stay in q0, move Right.
  363CSC
- Read blank in state q0, transition to accepting state.

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where:

- $Q = \{q_0, q_1, q_{accept}, q_{reject}\}$
- $\Sigma = \{C, S, Z, 3, 6\}$
- $\Gamma = \{C, S, Z, 0, 3, 6, \square\}$
- δ is defined by the table:

(q_0, C)	$(q_0, 3, R)$
(q_0, S)	$(q_0, 6, R)$
(q_0, Z)	$(q_1, 0, L)$
$(q_0, 0)$	$(q_0, 0, R)$
$(q_0, 3)$	(q_0, C, R)
$(q_0, 6)$	(q_0, S, R)
$(q_0, \square)$	$(q_{accept}, \square, L)$
(q_1, C)	$(q_1, 0, L)$
(q_1, S)	$(q_1, 0, L)$
(q_1, Z)	$(q_1, 0, L)$
$(q_1, 0)$	$(q_1, 0, L)$
$(q_1, 3)$	$(q_1, 0, L)$
$(q_1, 6)$	$(q_0, 6, R)$
$(q_1, \square)$	$(q_1, \square, R)$

Abstracting Turing Machines

- It is tedious to construct a Turing Machine using its *formal description*.
- Instead, we will construct Turing Machines using its *implementation description*
- Use English prose in a way to describe how the Turing Machine will function during its computation.
- Eventually when we are more confident with Turing Machines, we will construct Turing Machines using a *high-level description*.
- Use English prose to describe the algorithm we want the Turing Machine to do, with little implementation details.

How much detail is enough?

- We want to be accurate in our description of a Turing Machine's behaviour, and not assert anything that we do not know how to do.
- Turing Machines accept strings. If we want it to accept anything that is not a string (like integers, graphs, sets), then we will need to encode it as a string.
- Can we write a Turing Machine to output x+y when given (x,y) as input?
- Can we write a Turing Machine to output a path given a graph **G**, a starting node *v* and a destination node *w*?

Accept and Reject

We would like to use Turing Machines to tell us if an input string ***x*** belongs in a set ***S***.

In this case, we care very little about the output, and more about whether it halts in the accepting state or the rejecting state.

The language of a Turing Machine **M** is the set of all input strings such that if they are given to M, then **M** will halt in the accepting state. We call this set **L(M)**.

Any questions?

Many examples next

Example 2

- Describe how we can construct a Turing Machine such that it accepts the string “CSC” and rejects all other input.

- Have three states, q_0, q_1, q_2. Start at q_0.
- Check that the first letter is a “C”.
  - If it is, move head to the right, and go to state q_1.
  - Otherwise, reject.
- Check that the second letter is a “S”.
  - If is, move the head to the right, and go to state q_2.
  - Otherwise, reject.
- Check that the third letter is a “C”. If it is, then accept. Otherwise, reject.

What is the language of this Turing Machine? I.e. what is the set of all inputs that this Turing Machine accepts?

Example 3

- Describe how we can construct a Turing Machine **M** that has the following language:

$$L(M) = \{0^n 1^m, n,m \in N\}$$

Example 4

- Find the error in the description of this Turing Machine **M** that has the following language:

$$L(M) = \{0^n 1^n, n \in N\}$$

M(x):

- Start in state count0, scan the input from left to right. Count the amount of zeroes read until it sees a one. When a one or a blank is read, keep the head still and move to state count1.
- In state count1, keep scanning the input from left to right counting the amount of ones. If a zero is read at any point, reject the input. Otherwise, when it reads a blank, accept if the counts match, and reject otherwise.

Example 5

- Properly describe a Turing Machine **M** that has the following language:

$$L(M) = \{0^n1^n, n \in N\}$$

- To do this, we will use a method where it removes a zero, then a one, then a zero, and so on... you can begin with a high-level description, then refine it to be implementable as a Turing Machine.

Example 6

- Describe a few ways that we can force a Turing Machine to loop.
- If a Turing Machine M is given the input x, and it fails to halt after a finite amount of time, can we conclude that it is looping? Why or why not?

Example 7

- Suppose we want to construct a Turing Machine that accept integers. What are some ways we can encode an integer into a string?

Snap back to reality

If a Turing Machine does not halt on an input **x**, then we say it loops on **x**. Are we able to tell if a Turing Machine **M** will loop on an input **x**?

For some specific Turing Machines, sure. We can prove that a certain **M** will halt or loop on **x**
But what we really want to know: can we write a general program **H** that can tell if any given Turing Machine **M** will halt on any given input **x**? After all, we as programmers do not like infinite loops all that much!

We want the following behavior:

- **H(M,x):**
  - If M would loop if given the input x:
    - H will reject
  - Otherwise, if M would halt if given the input x:
    - H will accept

The Halting Problem

End of Tutorial 1