Week 7

# Turing Machines, Decidability, & Complexity

Sam Ruggerio

# Outline



Section 1

# Turing Machines

# Turing Machines

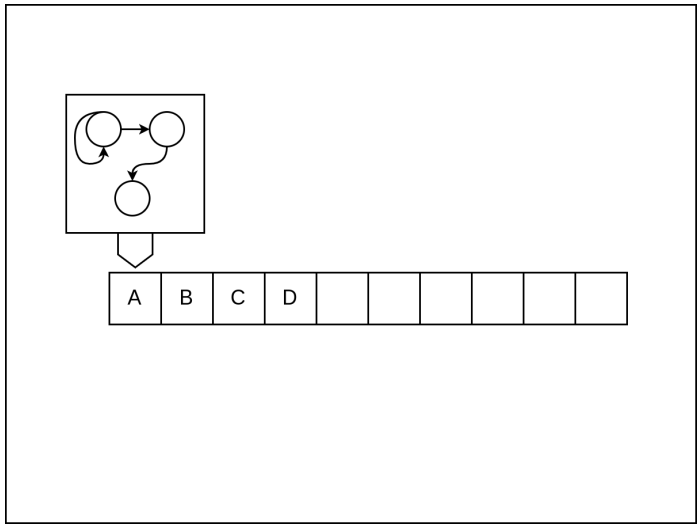
- Turing Machines were conceived by Alan Turing in 1936.
- They define the fundamental basis for all models of computation.
- They aren't necessarily fast, but tell us what we *can* compute with computers.

# Turing Machines

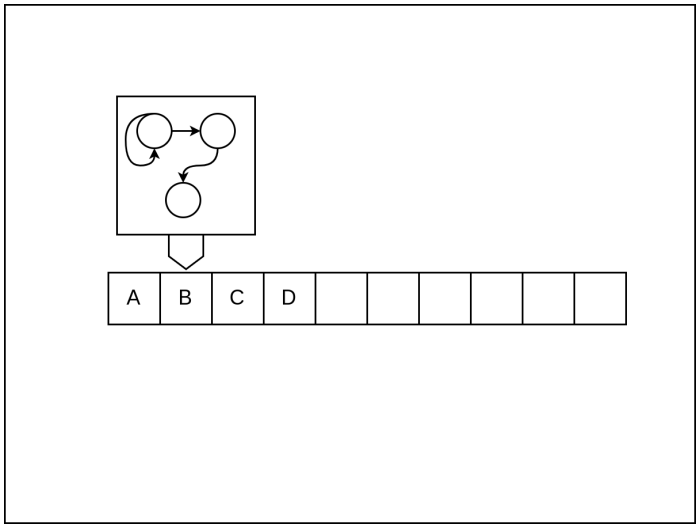
A Turing Machine is made of a few parts:

- A infinitely long, 1D tape of symbols, with blanks for unused cells.
- A state machine, which tells what the Turing Machine should do (think: DFA/NFA)
- The Read/Write head that points to a position on the tape.
- The state machine has actions of what to do upon reaching a particular state (read, write, move left, move right, accept, reject)
- Transitions between states depending on what the Turing Machine read.

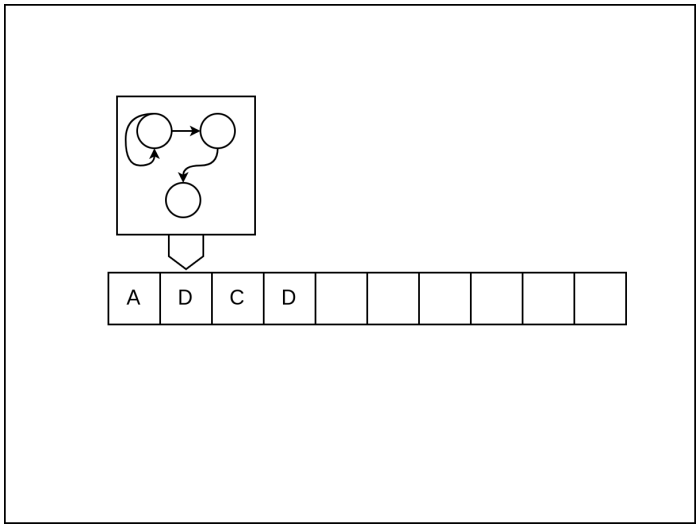
# Turing Machine



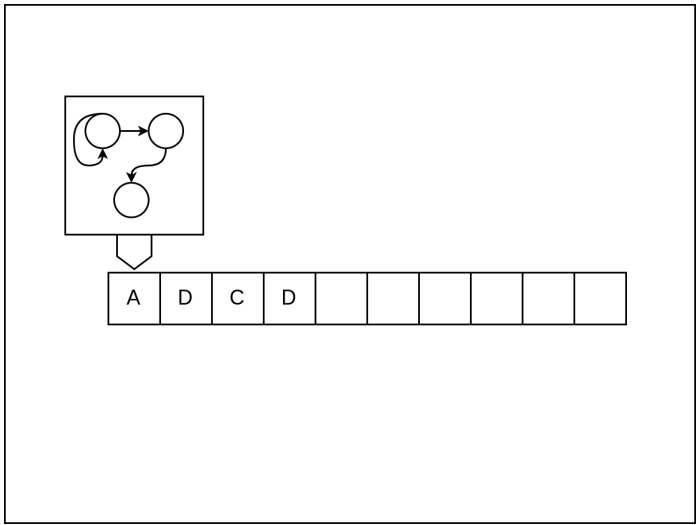

# Turing Machine



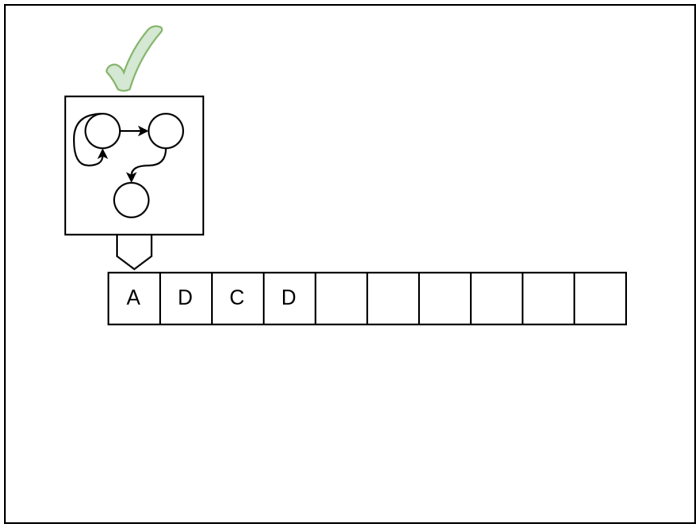
# Turing Machine



# Turing Machine



# Turing Machine




# Formal Definitions

- Let the Turing Machine $M$ be defined by $(\Gamma, \Sigma, Q, \delta)$
- $\Gamma$ is the tape alphabet, the set of symbols that can appear on the tape. The blank symbol is included: $\epsilon \in \Gamma$
- $\Sigma$ is the input alphabet, the set of symbols that can *initially* appear on the tape: $\Sigma \subseteq \Gamma \setminus \{\epsilon\}$
- $Q$ is the set of states. Three states are required to be in $Q$: START, ACCEPT, REJECT. Let $Q'$ be $Q$ without accept/reject
- $\delta$ is the transition function: $Q' \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$

# Tedium

- Turing machines are as fundamental as you can get w.r.t computation.
- As such, the programs are very long and complex for simple problems.
- Alonzo Church and Turing effectively proved that any problem that can be decided, can be decided with a Turing Machine (Church-Turing Thesis).

Questions?

Section 2

# Decision Problems

# Decision Problems

Decision problems are strictly true and false queries:

"What is **the index** of an element less than $k$?"

vs.

"**Does this** array contain a value less than $k$?"

# Decision Problems

Given an **oracle**, SmallestIDecision, which tells you whether an array has an element less than $k$, we can figure out which index that value is stored:

```
SmallestI(A[1..n], k):
    ind ← 0
    for i ← 1 to n + 1:
        if SmallestIDecision(A[i..n], k) = False:
            ind ← i − 1
            break
    return ind
```

# Questions!

Try solving *using* the decision problem, *without* comparing characters. Specifically, you should only solve these by checking if something equals True or False. (No, '1' does not typecheck to True)

1. How many 0s are in a string $w$?
   Does the string $w$ have $k$ 0s: DecisionKZero$(w, k)$
2. How many pairs of 1s are in a string $w$?
   Is the number of $1s$ in a string $w$ even? Decision1Even$(w)$

# Questions!

```
CountZeros(w[1..n]):
  for k ← 1 to n:
      if DecisionKZero(w, k) = True:
          return k
```

```
Count1Pairs(w[1..n]):
  c ← 0
  for i ← 1 to n:
      if Decision1Even(w[i]) = False:
          c ← c + 0.5
  return ⌊c⌋
```

# Decision Languages

If a problem only has a true or false response, we can define a language of that problem:

$$\text{SAT} = \{w \in \Sigma^* \mid w \text{ is a satisfiable boolean formula}\}$$

$$\text{SORT} = \{w \in \Sigma^* \mid w \text{ defines a sorted integer array}\}$$

$$\text{HALT} = \{w \in \Sigma^* \mid w \text{ is a program that halts}\}$$

# Turing Machines & Problems

For any Turing machine $M$ and language $L$:

- We say that $M$ ***recognizes/accepts*** $L$ if for any input $w \in L$, $M$ accepts $w$.
  - It's fine for $M$ to never stop on inputs that aren't in $L$, but it must accept every element of $L$.
- $M$ ***decides*** $L$ if for any input $w$, $M$ accepts if $w \in L$ and rejects otherwise.

# Turing Machines & Languages

For any Turing machine $M$ and input $w$:

- $\textsc{Accept}(M)$: the language of all inputs $w$ where $M$ accepts.
- $\textsc{Reject}(M)$: the language of all inputs $w$ where $M$ rejects.
- $\textsc{Halt}(M)$: the language of all inputs $w$ where $M$ gives a response (halts).
- $\textsc{Diverge}(M)$: the language of all inputs $w$ where $M$ never halts.

# Code is Data

- We can represent a Turing machine as a string encoding. (Enumerate the states, what state to go to, map the symbols to binary, etc.)
- The input to a Turing machine can be a Turing machine itself! (Think: programs running programs/VMs/Compilers/etc.)
- We use $M$ to say "the Turing Machine $M$", and $\langle M \rangle$ to say "the encoding of Turing Machine $M$"

# Simulating Machines

- Assume we have a Turing Machine $U$ designed to simulate another Turing machine $M$ on a given input $w$.
- This instance is denoted like so: $\langle M, w \rangle$
- The language of machines that accept their input is defined as:

$$\text{ACCEPT} = \{\langle M, w \rangle \mid M \text{ accepts } w\}$$

# Halting Problem

- We cannot decide (accepting/rejecting all inputs) whether a Turing program will halt.
- This result was proven by Turing in his original paper.
- Walkthrough of this proof in a future meeting!

Questions?

# Questions!

Sanity Check:

1. Is the language following language **recognizable** by a Turing Machine: HALT $= \{\langle M, w\rangle | M \text{ halts on input } w\}$
2. Can $\langle M\rangle$ be an input to Turing Machine $M$?
3. Can $M$ decide what will happen with itself? (i.e. $\langle M, \langle M\rangle\rangle$)

Section 3

# Complexity

# Runtime

Turing machines give us metrics to measure algorithm runtime. If given an input of length $n$:

1. How many steps does the Turing Machine take until it halts? (Runtime)
2. How many cells on the tape does the Turing Machine use? (Space)

# Complexity Classes

- Define P to be the set of decision *problems* that are decidable in polynomial time.
- Define NP to be the set of decision problems that are decidable in polynomial time via non-determinism.
- Define EXP to be the set of decision problems that are decidable in exponential time.
- Define NEXP to be the set of decision problems that are decidable in exponential time via non-determinism.

# Complexity Classes

What's above NEXP?

- Define $R$ to be the set of decision problems that are decidable.
- Define $RE$ to be the set of decision problems that are recursively enumerable/recognizable. (Halts, if the answer is Yes).

# Complexity Classes

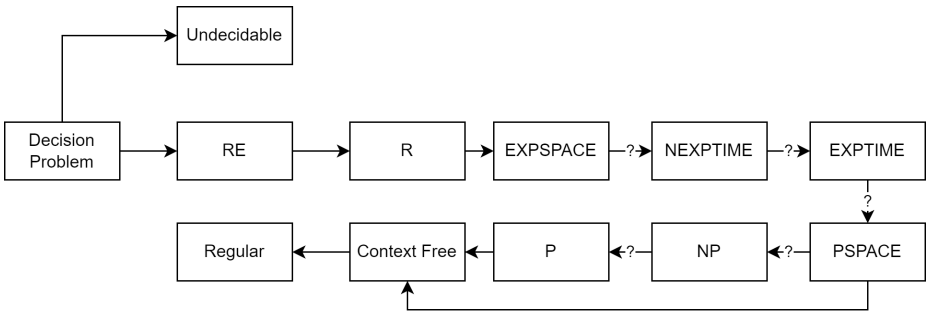
What about space?

- Define PSPACE to be the set of decision problems that are decidable in polynomial space.
- Define EXPSPACE to be the set of decision problems that are decidable in exponential space.

# Relations

We know that P and NP are subsets of PSPACE, but are any of these sets equal?
Is PSPACE = EXPTIME?
Is EXPTIME = NEXPTIME = EXPSPACE?

# Complexity Subsets




Questions?