15-453
TURING MACHINES
TURING MACHINE

INFINITE TAPE

q_1

INPUT

ANPUTFUN
read  write  move

0 → 0, R

□ → □, R

0 → 0, R

q_{accept}

0 → 0, R

□ → □, L

reject
read  write  move

\[
\begin{align*}
0 & \rightarrow 0, R \\
\square & \rightarrow \square, R \\
0 & \rightarrow 0, R \\
\square & \rightarrow \square, R \\
0 & \rightarrow 0, R \\
\square & \rightarrow \square, L
\end{align*}
\]
Definition: A Turing Machine is a 7-tuple $\Gamma = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\square \notin \Sigma$
- $\Gamma$ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state, and $q_{\text{reject}} \neq q_{\text{accept}}$
CONFIGURATIONS

11010q_7001110

corresponds to:

1 1 0 1 0 0 0 1 1 1 0
A Turing Machine $M$ accepts input $w$ if there is a sequence of configurations $C_1, \ldots, C_k$ such that
1. $C_1$ is a start configuration of $M$ on input $w$, ie $C_1$ is $q_0w$
2. each $C_i$ yields $C_{i+1}$, ie $M$ can legally go from $C_i$ to $C_{i+1}$ in a single step

\[
\begin{align*}
uaq_ibv & \quad \text{yields} \quad uq_jacv & \text{if} & \quad \delta(q_i, b) = (q_j, c, L) \\
uaqiqqv & \quad \text{yields} \quad uacq_jv & \text{if} & \quad \delta(q_i, b) = (q_j, c, R)
\end{align*}
\]
A Turing Machine $M$ accepts input $w$ if there is a sequence of configurations $C_1, \ldots, C_k$ such that

1. $C_1$ is a start configuration of $M$ on input $w$, i.e. $C_1$ is $q_0w$

2. each $C_i$ yields $C_{i+1}$, i.e. $M$ can legally go from $C_i$ to $C_{i+1}$ in a single step

3. $C_k$ is an accepting configuration, i.e. the state of the configuration is $q_{\text{accept}}$
A Turing Machine $M$ \textit{rejects} input $w$ if there is a sequence of configurations $C_1, \ldots, C_k$ such that

1. $C_1$ is a \textit{start} configuration of $M$ on input $w$, ie $C_1$ is $q_0 w$
2. each $C_i$ \textit{yields} $C_{i+1}$, ie $M$ can legally go from $C_i$ to $C_{i+1}$ in a single step
3. $C_k$ is a \textit{rejecting} configuration, ie the state of the configuration is $q_{\text{reject}}$
A TM decides a language if it accepts all strings in the language and rejects all strings not in the language.

A language is called **decidable** or **recursive** if some TM decides it.
\{ 0^{2^n} \mid n \geq 0 \} \text{ is decidable.}

**PSEUDOCODE:**

1. Sweep from left to right, cross out every other 0
2. If in stage 1, the tape had only one 0, accept
3. If in stage 1, the tape had an odd number of 0’s, reject
4. Move the head back to the first input symbol.
5. Go to stage 1.
\{ 0^{2^n} \mid n \geq 0 \}

A finite automaton with states:
- \(q_0\)
- \(q_1\)
- \(q_2\)
- \(q_3\)
- \(q_{\text{reject}}\)
- \(q_{\text{accept}}\)

Transitions:
- \(0 \rightarrow \square, R\)
- \(x \rightarrow x, R\)
- \(\square \rightarrow \square, R\)
- \(\square \rightarrow \square, L\)
- \(x \rightarrow x, L\)
- \(0 \rightarrow 0, L\)
- \(0 \rightarrow 0, R\)
- \(0 \rightarrow x, R\)
- \(x \rightarrow x, R\)
- \(x \rightarrow x, R\)
- \(x \rightarrow x, R\)
- \(x \rightarrow x, R\)
- \(\square \rightarrow \square, R\)
- \(\square \rightarrow \square, R\)
- \(\square \rightarrow \square, R\)

The automaton accepts strings where the number of 0s is a power of 2.
\{ 0^{2n} \mid n \geq 0 \}
A TM **decides** a language if it accepts all strings in the language and rejects all strings not in the language

A language is called **decidable** or **recursive** if some TM decides it

**Theorem:** \( L \) **decidable** \( \iff \neg L \) **decidable**

**Proof:** \( L \) has a machine \( M \) that accepts or rejects on all inputs. Define \( M' \) to be \( M \) with accept and reject states swapped. \( M' \) decides \( \neg L \).
Theorem: A, B decidable $\implies$ A union B decidable

Proof: Let M be a TM for A. Let M’ be a TM for B. Make a Union machine implementing the following pseudo-code:
(Intuition: use the even squares to simulate M, and the odd squares to simulate M’)

Double input size by writing each input symbol twice, starting with q_0 symbols. Use cross product construction to allow the finite state control to remember state of each TM. Move pebble around to always be one square left of position of head in M or M’, respectively. Odd phase: Bring head back to start symbol of tape, scan odd squares to find tape head location at pebble... accept if either M or M’ accept.
A TM **recognizes** a language if it accepts all and only those strings in the language.

A language is called **Turing-recognizable** or **recursively enumerable**, (or **r.e.** or **semi-decidable**) if some TM recognizes it.

A TM **decides** a language if it accepts all strings in the language and rejects all strings not in the language.

A language is called **decidable** or **recursive** if some TM decides it.
A TM **recognizes** a language if it accepts all and only those strings in the language

A language is called **Turing-recognizable** or **recursively enumerable** (or **r.e.** or **semi-decidable**) if some TM recognizes it

**FALSE:** $L \text{ r.e. } \iff \neg L \text{ r.e.}$

**Proof:** $L$ has a machine $M$ that accepts or rejects on all inputs. Define $M'$ to be $M$ with accept and reject states swapped. $M'$ decides $\neg L$. 
\[ w \in \Sigma^* \]

**TM**

\[ w \in L \ ? \]

- **yes** → **accept**
- **no** → **reject**

**L is decidable**

(Recursive)

**TM**

\[ w \in \Sigma^* \]

\[ w \in L \ ? \]

- **yes** → **accept**
- **no** → **reject or no output**

**L is semi-decidable**

(Recursively enumerable, Turing-recognizable)
A language is called **Turing-recognizable** or **recursively enumerable (r.e.)** or **semi-decidable** if some TM **recognizes** it.

A language is called **decidable** or **recursive** if some TM **decides** it.

**Theorem:** If A and ¬A are r.e. then A is recursive.
**Theorem:** If A and \( \neg A \) are r.e. then A is recursive

Suppose M accepts A. M’ accepts \( \neg A \) decidable. Use Odd squares/ Even squares simulation of M and M’. If x is accepted by the even squares reject it/accepted by the odd squares then accept x.
TURING MACHINE with WRITE ONLY output tape.

Outputs a sequence of strings separated by hash marks. Allows for a well defined infinite sequence of strings in the limit. The machine is said to enumerate the sequence of strings occurring on the tape.
Lex-order has an enumerator strings of length 1, the length 2, ….

Pairs of binary strings have a lex-order enumerator for each \( n > 0 \) list all pairs of strings \( a, b \) as \( \#a\#b\# \) where total length of \( a \) and \( b \) is \( n \).

Let \( \text{BINARY}(w) = \) pair of binary strings be any fixed way of encoding a pair of binary strings with a single binary string
Outputs a sequence of strings separated by hash marks. Allows for a well defined infinite sequence of strings in the limit. The machine is said to enumerate the set of strings occurring on the tape.
From every TM $M$ accepting $A$, there is a TM $M'$ outputting $A$.

For $n = 0$ to forever do
{
    Do $n$ parallel simulations of $M$ for $n$ steps for the first $n$ inputs
}

$M(0)$, $M(1)$, $M(2)$, $M(3)$..

Odd/Even trick becomes “modulo $n$” trick. If $M(x)$ accepts then output($x#$)
From every TM $M$ outputting $A$, there is a TM $M'$ accepting $A$.

$M''(X)$ run $M$, accept if $X$ output on tape.
Let $\mathbb{Z}^+ = \{1,2,3,4\ldots\}$. There exists a bijection between $\mathbb{Z}^+$ and $\mathbb{Z}^+ \times \mathbb{Z}^+$ (or $\mathbb{Q}^+$)

\[
(1,4) \quad (1,2) \quad (1,3) \quad (1,4) \quad (1,5) \quad \ldots \\
(2,1) \quad (2,2) \quad (2,3) \quad (2,4) \quad (2,5) \quad \ldots \\
(3,1) \quad (3,2) \quad (3,3) \quad (3,4) \quad (3,5) \quad \ldots \\
(4,1) \quad (4,2) \quad (4,3) \quad (4,4) \quad (4,5) \quad \ldots \\
(5,1) \quad (5,2) \quad (5,3) \quad (5,4) \quad (5,5) \quad \ldots 
\]
$w \in \Sigma^*$

$w \in L$ ?

TM

yes
accept

no
reject

$L$ is decidable (recursive)

$L$ is semi-decidable (recursively enumerable, Turing-recognizable)
$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
We can encode a TM as a string of 0s and 1s

\[ 0^n10^m10^k10^s10^t10^r10^u1\ldots \]

- **n states**
- **m tape symbols** (first k are input symbols)
- **start state**
- **accept state**
- **reject state**
- **blank symbol**

\[
( (p, a), (q, b, L) ) = 0^p10^a10^q10^b10
\]

\[
( (p, a), (q, b, R) ) = 0^p10^a10^q10^b11
\]
THE CHURCH-TURING THESIS

Intuitive Notion of Algorithms EQUALS Turing Machines
THE ACCEPTANCE PROBLEM

\[ A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \} \]

**Theorem:** \( A_{TM} \) is semi-decidable (r.e.)

but **NOT** decidable

\( A_{TM} \) is r.e. :

Define a TM \( U \) as follows:

On input \( (M, w) \), \( U \) runs \( M \) on \( w \). If \( M \) ever accepts, accept. If \( M \) ever rejects, reject.

**NB.** When we write “input \( (M, w) \)” we really mean “input code for (code for \( M, w \))”
Similarly, we can encode DFAs, NFAs, CFGs, etc. into strings of 0s and 1s.
So we can define the following languages:

\[ A_{DFA} = \{ (B, w) \mid B \text{ is a DFA that accepts string } w \} \]

**Theorem:** \( A_{DFA} \) is decidable

**Proof Idea:** Simulate \( B \) on \( w \)

\[ A_{NFA} = \{ (B, w) \mid B \text{ is an NFA that accepts string } w \} \]

**Theorem:** \( A_{NFA} \) is decidable

\[ A_{CFG} = \{ (G, w) \mid G \text{ is a CFG that generates string } w \} \]

**Theorem:** \( A_{CFG} \) is decidable

**Proof Idea:** Transform \( G \) into Chomsky Normal Form. Try all derivations of length up to \( 2|w|-1 \)
UNDECIDABLE PROBLEMS

THURSDAY Feb 13
There are languages over \{0,1\} that are not decidable

If we believe the Church-Turing Thesis, this is **MAJOR**: it means there are things that computers inherently cannot do.

We can prove this using a **counting argument**. We will show there is no **onto** function from the set of all Turing Machines to the set of all languages over \{0,1\}. **(Works for any \(\Sigma\))** Hence there are languages that have no decider.

Then we will prove something stronger: There are **semi-decidable (r.e.)** languages that are NOT decidable.
Turing Machines

Languages over \{0, 1\}
Let $L$ be any set and $2^L$ be the power set of $L$

**Theorem:** There is no onto map from $L$ to $2^L$

**Proof:** Assume, for a contradiction, that there is an onto map $f : L \rightarrow 2^L$

Let $S = \{ x \in L \mid x \notin f(x) \}$

If $S = f(y)$ then $y \in S$ if and only if $y \notin S$

Can give a more constructive argument!
Theorem: There is no onto function from the positive integers to the real numbers in (0, 1).

Proof: Suppose $f$ is any function mapping the positive integers to the real numbers in (0, 1):

\[
\begin{align*}
1 & \rightarrow 0.28347279\ldots \\
2 & \rightarrow 0.88388384\ldots \\
3 & \rightarrow 0.77635284\ldots \\
4 & \rightarrow 0.11111111\ldots \\
5 & \rightarrow 0.12345678\ldots \\
\vdots & \quad \vdots
\end{align*}
\]

Let $r = 11121\ldots$.

\[
\left[n\text{-th digit of } r\right] = \begin{cases} 
1 & \text{if } \left[n\text{-th digit of } f(n)\right] \neq 1 \\
2 & \text{otherwise}
\end{cases}
\]

$f(n) \neq r$ for all $n$ (Here, $r = 11121\ldots$)
THE MORAL:
No matter what $L$ is, $2^L$ always has more elements than $L$. 
Not all languages over \{0,1\} are decidable, in fact: not all languages over \{0,1\} are semi-decidable.

\{decidable languages over \{0,1\}\}
\{semi-decidable languages over \{0,1\}\}

\{Turing Machines\}
\{Strings of 0s and 1s\}

Set \(L\)  
Set of all subsets of \(L\): \(2^L\)
**THE ACCEPTANCE PROBLEM**

\[ A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \} \]

**Theorem:** \( A_{TM} \) is semi-decidable (r.e.) but **NOT** decidable

\( A_{TM} \) is r.e. :

Define a TM \( U \) as follows:

On input \((M, w)\), \( U \) runs \( M \) on \( w \). If \( M \) ever accepts, accept. If \( M \) ever rejects, reject.

**NB.** When we write “input \((M, w)\)” we really mean “input code for (code for \( M, w \))”
THE ACCEPTANCE PROBLEM

$A_{TM} = \{ (M, w) \mid M \text{ is a TM that accepts string } w \}$

Theorem: $A_{TM}$ is semi-decidable (r.e.)

but NOT decidable

$A_{TM}$ is r.e.: Define a TM $U$ as follows: $U$ is a universal TM

On input $(M, w)$, $U$ runs $M$ on $w$. If $M$ ever accepts, accept. If $M$ ever rejects, reject.

Therefore, $U$ accepts $(M, w) \iff M$ accepts $w \iff (M, w) \in A_{TM}$

Therefore, $U$ recognizes $A_{TM}$
$A_{TM} = \{ (M,w) \mid M \text{ is a TM that accepts string } w \}$

$A_{TM}$ is undecidable: (proof by contradiction)

Assume machine $H$ decides $A_{TM}$

$H( (M,w) ) = \begin{cases} 
\text{Accept} & \text{if } M \text{ accepts } w \\
\text{Reject} & \text{if } M \text{ does not accept } w 
\end{cases}$

Construct a new TM $D$ as follows: on input $M$, run $H$ on $(M,M)$ and output the opposite of $H$

$D( D ) = \begin{cases} 
\text{Reject} & \text{if } D \text{ accepts } D \\
\text{Accept} & \text{if } D \text{ does not accept } D 
\end{cases}$
<table>
<thead>
<tr>
<th>M_1</th>
<th>M_2</th>
<th>M_3</th>
<th>M_4</th>
<th>…</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_1</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
</tr>
<tr>
<td>M_2</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>M_3</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
</tr>
<tr>
<td>M_4</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
</tr>
</tbody>
</table>

D | reject | reject | accept | accept | ?
**Theorem:** $A_{TM}$ is r.e. but NOT decidable

**Cor:** $\neg A_{TM}$ is not even r.e.!
Let machine $H$ semi-decides $A_{TM}$ (Such $\exists$, why?)

$$H( (M,w) ) = \begin{cases} 
    \text{Accept} & \text{if } M \text{ accepts } w \\
    \text{Reject or} & \\
    \text{No output} & \text{if } M \text{ does not accept } w 
\end{cases}$$

Construct a new TM $D$ as follows: on input $M$, run $H$ on $(M,M)$ and output

$$D( D ) = \begin{cases} 
    \text{Reject} & \text{if } H( (D,D) ) \text{ Accepts} \\
    \text{Accept} & \text{if } H( (D,D) ) \text{ Rejects} \\
    \text{No output} & \text{if } H( (D,D) ) \text{ has No output} 
\end{cases}$$

$$H( (D,D) ) = \text{No output} \quad \text{No Contradictions !}$$
We have shown:
Given any machine $H$ for semi-deciding $A_{TM}$, we can effectively construct a TM $D$ such that $(D,D) \notin A_{TM}$ but $H$ fails to tell us that.

That is, $H$ fails to be a decider on instance $(D,D)$.

In other words,
Given any “good” candidate for deciding the *Acceptance Problem*, we can effectively construct an instance where the candidate fails.
THE classical HALTING PROBLEM

\( \text{HALT}_{\text{TM}} = \{ (M, w) \mid M \text{ is a TM that halts on string } w \} \)

Theorem: \( \text{HALT}_{\text{TM}} \) is undecidable

Proof: Assume, for a contradiction, that TM \( H \) decides \( \text{HALT}_{\text{TM}} \)

We use \( H \) to construct a TM \( D \) that decides \( \text{A}_{\text{TM}} \)

On input \((M, w)\), \( D \) runs \( H \) on \((M, w)\):

If \( H \) rejects then reject

If \( H \) accepts, run \( M \) on \( w \) until it halts:

Accept if \( M \) accepts, ie halts in an accept state
Otherwise reject
If $M$ doesn't halt:

REJECT

If $M$ halts:

Does $M$ halt on $w$?

If $M$ doesn't halt: REJECT

ACCEPT if halts in accept state

REJECT otherwise
In many cases, one can show that a language $L$ is undecidable by showing that if it is decidable, then so is $A_{TM}$.

We reduce deciding $A_{TM}$ to deciding the language in question.

$A_{TM} \leq L$

We just showed: $A_{TM} \leq Halt_{TM}$

Is $Halt_{TM} \leq A_{TM}$?