BBM402-Lecture 6: Decidable Languages and the Halting Problem

Lecturer: Lale Özkahya

Resources for the presentation:
https://courses.engr.illinois.edu/cs373/fa2010/lectures
Decidable and Recognizable Languages

Recall: Definition

A Turing machine $M$ is said to recognize a language $L$ if $L = L(M)$.
Decidable and Recognizable Languages

Recall: Definition

A Turing machine $M$ is said to **recognize** a language $L$ if $L = L(M)$. A Turing machine $M$ is said to **decide** a language $L$ if $L = L(M)$ and $M$ halts on every input.
Decidable and Recognizable Languages

Recall: Definition

A Turing machine $M$ is said to recognize a language $L$ if $L = L(M)$. A Turing machine $M$ is said to decide a language $L$ if $L = L(M)$ and $M$ halts on every input.

$L$ is said to be Turing-recognizable (or simply recognizable) if there exists a TM $M$ which recognizes $L$. 

We just saw some example algorithms all of which terminate in a finite number of steps, and output yes or no (accept or reject). i.e., They decide the corresponding languages.
Decidability and Recognizability of Languages

Recall: Definition

A Turing machine $M$ is said to **recognize** a language $L$ if $L = L(M)$. A Turing machine $M$ is said to **decide** a language $L$ if $L = L(M)$ and $M$ halts on every input.

$L$ is said to be **Turing-recognizable** (or simply recognizable) if there exists a TM $M$ which recognizes $L$. $L$ is said to be **Turing-decidable** (or simply decidable) if there exists a TM $M$ which decides $L$. 

- Every finite language is decidable: For example, by a TM that has all the strings in the language "hard-coded" into it.
- We just saw some example algorithms all of which terminate in a finite number of steps, and output yes or no (accept or reject). i.e., They decide the corresponding languages.
Decidable and Recognizable Languages

Recall: Definition

A Turing machine $M$ is said to **recognize** a language $L$ if $L = L(M)$. A Turing machine $M$ is said to **decide** a language $L$ if $L = L(M)$ and $M$ halts on every input.

$L$ is said to be **Turing-recognizable** (or simply recognizable) if there exists a TM $M$ which recognizes $L$. $L$ is said to be **Turing-decidable** (or simply decidable) if there exists a TM $M$ which decides $L$.

- Every finite language is decidable
Decidable and Recognizable Languages

Recall: Definition
A Turing machine $M$ is said to recognize a language $L$ if $L = L(M)$. A Turing machine $M$ is said to decide a language $L$ if $L = L(M)$ and $M$ halts on every input.

$L$ is said to be Turing-recognizable (or simply recognizable) if there exists a TM $M$ which recognizes $L$. $L$ is said to be Turing-decidable (or simply decidable) if there exists a TM $M$ which decides $L$.

- Every finite language is decidable: For e.g., by a TM that has all the strings in the language “hard-coded” into it.
Decidable and Recognizable Languages

<table>
<thead>
<tr>
<th>Recall: Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Turing machine $M$ is said to <strong>recognize</strong> a language $L$ if $L = L(M)$. A Turing machine $M$ is said to <strong>decide</strong> a language $L$ if $L = L(M)$ and $M$ halts on every input.</td>
</tr>
</tbody>
</table>

$L$ is said to be **Turing-recognizable** (or simply recognizable) if there exists a TM $M$ which recognizes $L$. $L$ is said to be **Turing-decidable** (or simply decidable) if there exists a TM $M$ which decides $L$.

- Every finite language is decidable: For e.g., by a TM that has all the strings in the language “hard-coded” into it
- We just saw some example algorithms all of which terminate in a finite number of steps, and output yes or no (accept or reject).
Decidable and Recognizable Languages

Recall: Definition
A Turing machine $M$ is said to recognize a language $L$ if $L = L(M)$. A Turing machine $M$ is said to decide a language $L$ if $L = L(M)$ and $M$ halts on every input.

$L$ is said to be Turing-recognizable (or simply recognizable) if there exists a TM $M$ which recognizes $L$. $L$ is said to be Turing-decidable (or simply decidable) if there exists a TM $M$ which decides $L$.

- Every finite language is decidable: For e.g., by a TM that has all the strings in the language “hard-coded” into it
- We just saw some example algorithms all of which terminate in a finite number of steps, and output yes or no (accept or reject). i.e., They decide the corresponding languages.
But not all languages are decidable!
But not all languages are decidable! In the next class we will see an example:

$A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$ is undecidable
But not all languages are decidable! In the next class we will see an example:

- $A_{TM} = \{ \langle M, w \rangle \mid M$ is a TM and $M$ accepts $w \}$ is undecidable

However $A_{TM}$ is Turing-recognizable!
But not all languages are decidable! In the next class we will see an example:

- $A_{TM} = \{ \langle M, w \rangle \mid M$ is a TM and $M$ accepts $w \}$ is undecidable

- However $A_{TM}$ is Turing-recognizable!

**Proposition**

There are languages which are recognizable, but not decidable
Recognizing $A_{TM}$

Program $U$ for recognizing $A_{TM}$:

On input $\langle M, w \rangle$
- simulate $M$ on $w$
- if simulated $M$ accepts $w$, then accept
- else reject (by moving to $q_{rej}$)
Recognizing $A_{TM}$

Program $U$ for recognizing $A_{TM}$:

On input $\langle M, w \rangle$
  simulate $M$ on $w$
  if simulated $M$ accepts $w$, then accept
  else reject (by moving to $q_{\text{rej}}$)

$U$ (the Universal TM) accepts $\langle M, w \rangle$ iff $M$ accepts $w$. i.e.,

$$L(U) = A_{TM}$$
Recognizing $A_{TM}$

Program $U$ for recognizing $A_{TM}$:

- **On input** $\langle M, w \rangle$
  - simulate $M$ on $w$
  - if simulated $M$ accepts $w$, then accept
  - else reject (by moving to $q_{rej}$)

$U$ (the Universal TM) accepts $\langle M, w \rangle$ iff $M$ accepts $w$. i.e.,

$$L(U) = A_{TM}$$

But $U$ does not decide $A_{TM}$
Recognizing $A_{TM}$

Program $U$ for recognizing $A_{TM}$:

On input $\langle M, w \rangle$

- simulate $M$ on $w$
  - if simulated $M$ accepts $w$, then accept
  - else reject (by moving to $q_{\text{rej}}$)

$U$ (the Universal TM) accepts $\langle M, w \rangle$ iff $M$ accepts $w$. i.e.,

$$L(U) = A_{TM}$$

But $U$ does not decide $A_{TM}$: If $M$ rejects $w$ by not halting, $U$ rejects $\langle M, w \rangle$ by not halting.
Recognizing $A_{TM}$

Program $U$ for recognizing $A_{TM}$:

On input $\langle M, w \rangle$

simulate $M$ on $w$

if simulated $M$ accepts $w$, then accept
else reject (by moving to $q_{\text{rej}}$)

$U$ (the Universal TM) accepts $\langle M, w \rangle$ iff $M$ accepts $w$. i.e.,

$$L(U) = A_{TM}$$

But $U$ does not decide $A_{TM}$: If $M$ rejects $w$ by not halting, $U$ rejects $\langle M, w \rangle$ by not halting. Indeed (as we shall see) no TM decides $A_{TM}$. 
Proposition

If $L$ and $\bar{L}$ are recognizable, then $L$ is decidable

Proof.

Program $P$ for deciding $L$, given programs $P_L$ and $P_{\bar{L}}$ for recognizing $L$ and $\bar{L}$:
Proposition

If \( L \) and \( \bar{L} \) are recognizable, then \( L \) is decidable

Proof.

Program \( P \) for deciding \( L \), given programs \( P_L \) and \( P_{\bar{L}} \) for recognizing \( L \) and \( \bar{L} \):

- On input \( x \), simulate \( P_L \) and \( P_{\bar{L}} \) on input \( x \).
Deciding vs. Recognizing

Proposition

If $L$ and $\overline{L}$ are recognizable, then $L$ is decidable

Proof.

Program $P$ for deciding $L$, given programs $P_L$ and $P_{\overline{L}}$ for recognizing $L$ and $\overline{L}$:

- On input $x$, simulate $P_L$ and $P_{\overline{L}}$ on input $x$. Whether $x \in L$ or $x \not\in L$, one of $P_L$ and $P_{\overline{L}}$ will halt in finite number of steps.
- Which one to simulate first?
Deciding vs. Recognizing

Proposition

If $L$ and $\overline{L}$ are recognizable, then $L$ is decidable

Proof.

Program $P$ for deciding $L$, given programs $P_L$ and $P_{\overline{L}}$ for recognizing $L$ and $\overline{L}$:

- On input $x$, simulate $P_L$ and $P_{\overline{L}}$ on input $x$. Whether $x \in L$ or $x \notin L$, one of $P_L$ and $P_{\overline{L}}$ will halt in finite number of steps.
- Which one to simulate first? Either could go on forever.
Proposition

If $L$ and $\overline{L}$ are recognizable, then $L$ is decidable

Proof.

Program $P$ for deciding $L$, given programs $P_L$ and $P_{\overline{L}}$ for recognizing $L$ and $\overline{L}$:

- On input $x$, simulate $P_L$ and $P_{\overline{L}}$ on input $x$. Whether $x \in L$ or $x \notin L$, one of $P_L$ and $P_{\overline{L}}$ will halt in finite number of steps.
- Which one to simulate first? Either could go on forever.
- On input $x$, simulate in parallel $P_L$ and $P_{\overline{L}}$ on input $x$ until either $P_L$ or $P_{\overline{L}}$ accepts.
Deciding vs. Recognizing

**Proposition**

*If \( L \) and \( \overline{L} \) are recognizable, then \( L \) is decidable*

**Proof.**

Program \( P \) for deciding \( L \), given programs \( P_L \) and \( P_{\overline{L}} \) for recognizing \( L \) and \( \overline{L} \):

- On input \( x \), simulate \( P_L \) and \( P_{\overline{L}} \) on input \( x \). Whether \( x \in L \) or \( x \notin L \), one of \( P_L \) and \( P_{\overline{L}} \) will halt in finite number of steps.
- Which one to simulate first? Either could go on forever.
- On input \( x \), simulate in parallel \( P_L \) and \( P_{\overline{L}} \) on input \( x \) until either \( P_L \) or \( P_{\overline{L}} \) accepts.
- If \( P_L \) accepts, accept \( x \) and halt. If \( P_{\overline{L}} \) accepts, reject \( x \) and halt.
Deciding vs. Recognizing

Proof (contd).

In more detail, $P$ works as follows:

On input $x$
for $i = 1, 2, 3, \ldots$
    simulate $P_L$ on input $x$ for $i$ steps
    simulate $P_{\overline{L}}$ on input $x$ for $i$ steps
    if either simulation accepts, break
if $P_L$ accepted, accept $x$ (and halt)
if $P_{\overline{L}}$ accepted, reject $x$ (and halt)
Proof (contd).

In more detail, $P$ works as follows:

On input $x$
for $i = 1, 2, 3, \ldots$

simulate $P_L$ on input $x$ for $i$ steps
simulate $P_{\overline{L}}$ on input $x$ for $i$ steps
if either simulation accepts, break
if $P_L$ accepted, accept $x$ (and halt)
if $P_{\overline{L}}$ accepted, reject $x$ (and halt)

(Alternately, maintain configurations of $P_L$ and $P_{\overline{L}}$, and in each iteration of the loop advance both their simulations by one step.)
So far:

- $A_{TM}$ is undecidable (next lecture)
- But it is recognizable
So far:

- $A_{\text{TM}}$ is undecidable (next lecture)
- But it is recognizable
- Is every language recognizable?
Deciding vs. Recognizing

So far:

- $A_{TM}$ is undecidable (next lecture)
- But it is recognizable
- Is every language recognizable? No!

Note: Decidable languages are closed under complementation, but recognizable languages are not.
Deciding vs. Recognizing

So far:

- $A_{TM}$ is undecidable (next lecture)
- But it is recognizable
- Is every language recognizable? No!

Proposition

$\overline{A_{TM}}$ is unrecognizable
Deciding vs. Recognizing

So far:
- $A_{TM}$ is undecidable (next lecture)
- But it is recognizable
- Is every language recognizable? No!

**Proposition**

$\overline{A_{TM}}$ is unrecognizable

**Proof.**

If $\overline{A_{TM}}$ is recognizable, since $A_{TM}$ is recognizable, the two languages will be decidable too!
Deciding vs. Recognizing

So far:

- $A_{TM}$ is undecidable (next lecture)
- But it is recognizable
- Is every language recognizable? No!

**Proposition**

$\overline{A_{TM}}$ is unrecognizable

**Proof.**

If $A_{TM}$ is recognizable, since $A_{TM}$ is recognizable, the two languages will be decidable too!

**Note:** Decidable languages are closed under complementation, but recognizable languages are not.
A decision problem requires checking if an input (string) has some property.
A decision problem requires checking if an input (string) has some property. Thus, a decision problem is a function from strings to boolean.
A decision problem requires checking if an input (string) has some property. Thus, a decision problem is a function from strings to boolean.

A decision problem is represented as a formal language consisting of those strings (inputs) on which the answer is “yes”.

---

Agha-Viswanathan
CS373
A Turing Machine on an input $w$ either (halts and) accepts, or (halts and) rejects, or never halts.
Recursive Enumerability

- A Turing Machine on an input $w$ either (halts and) accepts, or (halts and) rejects, or never halts.
- The language of a Turing Machine $M$, denoted as $L(M)$, is the set of all strings $w$ on which $M$ accepts.
A Turing Machine on an input $w$ either (halts and) accepts, or (halts and) rejects, or never halts.

The language of a Turing Machine $M$, denoted as $L(M)$, is the set of all strings $w$ on which $M$ accepts.

A language $L$ is recursively enumerable/Turing recognizable if there is a Turing Machine $M$ such that $L(M) = L$. 
Decidability

A language $L$ is **decidable** if there is a Turing machine $M$ such that $L(M) = L$ and $M$ halts on every input.
Decidability

- A language $L$ is **decidable** if there is a Turing machine $M$ such that $L(M) = L$ and $M$ halts on every input.
- Thus, if $L$ is decidable then $L$ is recursively enumerable.
A language $L$ is *undecidable* if $L$ is not decidable.

That is, there is no Turing machine $M$ such that $L(M)$ = $L$, or $L$ is recursively enumerable but not decidable. That is, any Turing machine $M$ such that $L(M)$ = $L$, $M$ does not halt on some inputs.
A language $L$ is **undecidable** if $L$ is not decidable. Thus, there is no Turing machine $M$ that halts on every input and $L(M) = L$. 
A language \( L \) is **undecidable** if \( L \) is not decidable. Thus, there is no Turing machine \( M \) that halts on every input and \( L(M) = L \).

This means that either \( L \) is not recursively enumerable. That is, there is no Turing machine \( M \) such that \( L(M) = L \), or
Undecidability

Definition

A language $L$ is **undecidable** if $L$ is not decidable. Thus, there is no Turing machine $M$ that halts on every input and $L(M) = L$.

- This means that either $L$ is not recursively enumerable. That is there is no turing machine $M$ such that $L(M) = L$, or

- $L$ is recursively enumerable but not decidable. That is, any Turing machine $M$ such that $L(M) = L$, $M$ does not halt on some inputs.
Big Picture

Languages

Recursively Enumerable

Decidable

Regular

$L_{0n1n}$

Relationship between classes of Languages
For the rest of this lecture, let us fix the input alphabet to be \( \{0, 1\} \)
For the rest of this lecture, let us fix the input alphabet to be \( \{0, 1\} \); a string over any alphabet can be encoded in binary.
For the rest of this lecture, let us fix the input alphabet to be \{0, 1\}; a string over any alphabet can be encoded in binary.

Any Turing Machine/program $M$ can itself be encoded as a binary string.
For the rest of this lecture, let us fix the input alphabet to be \{0, 1\}; a string over any alphabet can be encoded in binary.

Any Turing Machine/program \( M \) can itself be encoded as a binary string. Moreover every binary string can be thought of as encoding a TM/program.
For the rest of this lecture, let us fix the input alphabet to be \{0, 1\}; a string over any alphabet can be encoded in binary.

Any Turing Machine/program \( M \) can itself be encoded as a binary string. Moreover every binary string can be thought of as encoding a TM/program. (If not the correct format, considered to be the encoding of a default TM.)
For the rest of this lecture, let us fix the input alphabet to be \{0, 1\}; a string over any alphabet can be encoded in binary.

Any Turing Machine/program $M$ can itself be encoded as a binary string. Moreover every binary string can be thought of as encoding a TM/program. (If not the correct format, considered to be the encoding of a default TM.)

We will consider decision problems (language) whose inputs are Turing Machine (encoded as a binary string)
The Diagonal Language

Definition
Define \( L_d = \{ M \mid M \not\in L(M) \} \).
The Diagonal Language

Definition

Define $L_d = \{M \mid M \not\in L(M)\}$. Thus, $L_d$ is the collection of Turing machines (programs) $M$ such that $M$ does not halt and accept when given itself as input.
A non-Recursively Enumerable Language

Proposition

$L_d$ is not recursively enumerable.
A non-Recursively Enumerable Language

Proposition

$L_d$ is not recursively enumerable.

Proof.

Recall that,
A non-Recursively Enumerable Language

Proposition

$L_d$ is not recursively enumerable.

Proof.

Recall that,

- Inputs are strings over \{0, 1\}
A non-Recursively Enumerable Language

Proposition

\( L_d \) is not recursively enumerable.

Proof.

Recall that,

- Inputs are strings over \( \{0, 1\} \)
- Every Turing Machine can be described by a binary string and every binary string can be viewed as Turing Machine
A non-Recursively Enumerable Language

Proposition

$L_d$ is not recursively enumerable.

Proof.

Recall that,

- Inputs are strings over \{0, 1\}
- Every Turing Machine can be described by a binary string and every binary string can be viewed as Turing Machine
- In what follows, we will denote the $i^{th}$ binary string (in lexicographic order) as the number $i$. 
Proposition

\[ L_d \text{ is not recursively enumerable.} \]

Proof.

Recall that,

- Inputs are strings over \( \{0, 1\} \)
- Every Turing Machine can be described by a binary string and every binary string can be viewed as a Turing Machine.
- In what follows, we will denote the \( i \)th binary string (in lexicographic order) as the number \( i \). Thus, we can say \( j \in L(i) \), which means that the Turing machine corresponding to \( i \)th binary string accepts the \( j \)th binary string.
Completing the proof
Diagonalization: Cantor

Proof (contd).

We can organize all programs and inputs as a (infinite) matrix, where the \((i, j)\)th entry is \(Y\) if and only if \(j \in L(i)\).

<table>
<thead>
<tr>
<th>TMs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>6</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Suppose \(L_d\) is recognized by a Turing machine, which is the \(j\)th binary string. i.e., \(L_d = L(j)\). But \(j \in L_d\) iff \(j \not\in L(j)!\) □
Completing the proof
Diagonalization: Cantor

Proof (contd).
We can organize all programs and inputs as a (infinite) matrix, where the \((i,j)\)th entry is \(Y\) if and only if \(j \in L(i)\).

<table>
<thead>
<tr>
<th>Inputs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>(\cdots)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMs</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

Suppose \(L_d\) is recognized by a Turing machine, which is the \(j\)th binary string. i.e., \(L_d = L(j)\).
Completing the proof

Proof (contd).

We can organize all programs and inputs as a (infinite) matrix, where the \((i,j)\)th entry is Y if and only if \(j \in L(i)\).

<table>
<thead>
<tr>
<th>TMs</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>(\ldots)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>6</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Suppose \(L_d\) is recognized by a Turing machine, which is the \(j\)th binary string. i.e., \(L_d = L(j)\). But \(j \in L_d\) iff \(j \notin L(j)\)! □
Consider the following program

On input $i$
  Run program $i$ on $i$
  Output ‘‘yes’’ if $i$ does not accept $i$
  Output ‘‘no’’ if $i$ accepts $i$
Consider the following program

On input \( i \)
- Run program \( i \) on \( i \)
- Output ‘‘yes’’ if \( i \) does not accept \( i \)
- Output ‘‘no’’ if \( i \) accepts \( i \)

Does the above program recognize \( L_d \)?
Consider the following program

On input \(i\)
- Run program \(i\) on \(i\)
- Output ‘‘yes’’ if \(i\) does not accept \(i\)
- Output ‘‘no’’ if \(i\) accepts \(i\)

Does the above program recognize \(L_d\)? No, because it may never output “yes” if \(i\) does not halt on \(i\).
Question

Is there a machine model such that

- all programs in the model halt on all inputs, and
- for each problem decidable by a TM, there is a program in the model that decides it?
There is no such model!

Suppose there is a programming language in which all programs always halt. Programs in this language can be described by binary strings, and can be simulated by TMs. Consider the Turing Machine $M_d$ on input $i$:

- Run program $i$ on $i$.
- Output "yes" if $i$ does not accept $i$.
- Output "no" if $i$ accepts $i$.

$M_d$ always halts and solves a problem not solved by any program in our language! Inability to halt is essential to capture all computation.

Agha-Viswanathan CS373
Answer

There is no such model! Suppose there is a programming language in which all programs always halt.
Models for Decidable Languages

Answer

There is no such model! Suppose there is a programming language in which all programs always halt. Programs in this language can be described by binary strings, and can be simulated by TMs.
There is no such model! Suppose there is a programming language in which all programs always halt. Programs in this language can be described by binary strings, and can be simulated by TMs. Consider the Turing Machine $M_d$

On input $i$
- Run program $i$ on $i$
- Output ‘‘yes’’ if $i$ does not accept $i$
- Output ‘‘no’’ if $i$ accepts $i$
There is no such model! Suppose there is a programming language in which all programs always halt. Programs in this language can be described by binary strings, and can be simulated by TMs. Consider the Turing Machine $M_d$

On input $i$

- Run program $i$ on $i$
- Output "yes" if $i$ does not accept $i$
- Output "no" if $i$ accepts $i$

$M_d$ always halts and solves a problem not solved by any program in our language!
Answer

There is no such model! Suppose there is a programming language in which all programs always halt. Programs in this language can be described by binary strings, and can be simulated by TMs. Consider the Turing Machine $M_d$

On input $i$

- Run program $i$ on $i$
- Output ‘‘yes’’ if $i$ does not accept $i$
- Output ‘‘no’’ if $i$ accepts $i$

$M_d$ always halts and solves a problem not solved by any program in our language! Inability to halt is essential to capture all computation.
Recursively Enumerable but not Decidable

- $L_d$ not recursively enumerable, and therefore not decidable.
Recursively Enumerable but not Decidable

- $L_d$ not recursively enumerable, and therefore not decidable. Are there languages that are recursively enumerable but not decidable?
Recursively Enumerable but not Decidable

- $L_d$ not recursively enumerable, and therefore not decidable. Are there languages that are recursively enumerable but not decidable?

- Yes, $A_{TM} = \{\langle M, w \rangle \mid M$ is a TM and $M$ accepts $w\}$
The Universal Language

Proposition

$A_{TM}$ is r.e. but not decidable.
**Proposition**

\( A_{TM} \) is r.e. but not decidable.

**Proof.**

We have already seen that \( A_{TM} \) is r.e.
Proposition

\[ A_{TM} \text{ is r.e. but not decidable.} \]

Proof.

We have already seen that \( A_{TM} \) is r.e. Suppose (for contradiction) \( A_{TM} \) is decidable. Then there is a TM \( M \) that always halts and \( L(M) = A_{TM} \).
Proposition

\( A_{TM} \) is r.e. but not decidable.

Proof.

We have already seen that \( A_{TM} \) is r.e. Suppose (for contradiction) \( A_{TM} \) is decidable. Then there is a TM \( M \) that always halts and \( L(M) = A_{TM} \). Consider a TM \( D \) as follows:

On input \( i \)
- Run \( M \) on input \( \langle i, i \rangle \)
- Output ‘yes’ if \( i \) rejects \( i \)
- Output ‘no’ if \( i \) accepts \( i \)
Proposition

\( A_{TM} \) is r.e. but not decidable.

Proof.

We have already seen that \( A_{TM} \) is r.e. Suppose (for contradiction) \( A_{TM} \) is decidable. Then there is a TM \( M \) that always halts and \( L(M) = A_{TM} \). Consider a TM \( D \) as follows:

On input \( i \)

- Run \( M \) on input \( \langle i, i \rangle \)
- Output ‘‘yes’’ if \( i \) rejects \( i \)
- Output ‘‘no’’ if \( i \) accepts \( i \)

Observe that \( L(D) = L_d \)!
Proposition

$A_{TM}$ is r.e. but not decidable.

Proof.

We have already seen that $A_{TM}$ is r.e. Suppose (for contradiction) $A_{TM}$ is decidable. Then there is a TM $M$ that always halts and $L(M) = A_{TM}$. Consider a TM $D$ as follows:

On input $i$

Run $M$ on input $\langle i, i \rangle$

Output ‘‘yes’’ if $i$ rejects $i$

Output ‘‘no’’ if $i$ accepts $i$

Observe that $L(D) = L_d$! But, $L_d$ is not r.e. which gives us the contradiction. □
A more complete Big Picture

Languages

Recursively Enumerable

Decidable

Regular

$L_d, \overline{A_{TM}}$

$L_{0^n1^n}$

$A_{TM}$