BBM401-Lecture 7: 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.

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.

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

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).

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.

Decidable and Recognizable Languages

- But not all languages are decidable!

Decidable and Recognizable Languages

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

Decidable and Recognizable Languages

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

Decidable and Recognizable Languages

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

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

Recognizing A_{TM}

Program U for recognizing A_{TM}:

```
On input ⟨M, w⟩
    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.,

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

But U does not decide A_{TM}

Recognizing A_{TM}

Program U for recognizing A_{TM}:

```
On input ⟨M, w⟩
    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.,

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

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 ⟨M, w⟩
    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.,

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

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}.

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 $\overline{L}$:

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.

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?

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.

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

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)

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)

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

Deciding vs. Recognizing

So far:

- A_{TM} is undecidable (next lecture)
- But it is recognizable

Deciding vs. Recognizing

So far:

- A_{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!

Deciding vs. Recognizing

So far:

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

Proposition

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

Deciding vs. Recognizing

So far:

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

Proposition

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

Proof.

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

Deciding vs. Recognizing

So far:

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

Proposition

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

Proof.

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

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

Decision Problems and Languages

- A decision problem requires checking if an input (string) has some property.

Decision Problems and Languages

- A decision problem requires checking if an input (string) has some property. Thus, a decision problem is a function from `strings` to `boolean`.

Decision Problems and Languages

- 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".

Recursive Enumerability

- 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.

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 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.

Undecidability

Definition

A language L is undecidable if L is not decidable.

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 $L(M) = L$.

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

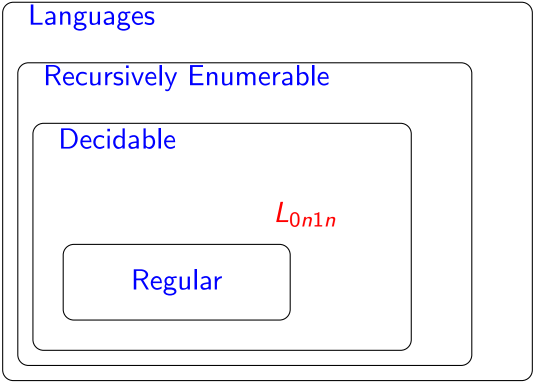
Undecidability

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



Relationship between classes of Languages

Machines as Strings

- For the rest of this lecture, let us fix the input alphabet to be $\{0, 1\}$

Machines as Strings

- 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.

Machines as Strings

- 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.

Machines as Strings

- 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.

Machines as Strings

- 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.)

Machines as Strings

- 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 $L_d = \{M \mid M \notin L(M)\}$.

The Diagonal Language

Definition

Define $L_d = \{M \mid M \notin L(M)\}$. 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 ith binary string (in lexicographic order) as the number i.

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 ith 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 ith binary string accepts the jth 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 $j \in L(i)$.

Inputs $\longrightarrow$

TMs $\downarrow$		1	2	3	4	5	6	7	$\cdots$
	1	N	N	N	N	N	N	N	
	2	N	N	N	N	N	N	N	
	3	Y	N	Y	N	Y	Y	Y	
	4	N	Y	N	Y	Y	N	N	
	5	N	Y	N	Y	Y	N	N	
	6	N	N	Y	N	Y	N	Y	

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 $j \in L(i)$.

Inputs $\longrightarrow$

TMs $\downarrow$		1	2	3	4	5	6	7	$\cdots$
	1	N	N	N	N	N	N	N	
	2	N	N	N	N	N	N	N	
	3	Y	N	Y	N	Y	Y	Y	
	4	N	Y	N	Y	Y	N	N	
	5	N	Y	N	Y	Y	N	N	
	6	N	N	Y	N	Y	N	Y	

Suppose L_d is recognized by a Turing machine, which is the jth binary string. i.e., $L_d = L(j)$.

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 $j \in L(i)$.

Inputs $\longrightarrow$

TMs $\downarrow$

	1	2	3	4	5	6	7	$\cdots$
1	$\boxed{\text{N}}$	N	N	N	N	N	N	
2	N	$\boxed{\text{N}}$	N	N	N	N	N	
3	Y	N	$\boxed{\text{Y}}$	N	Y	Y	Y	
4	N	Y	N	$\boxed{\text{Y}}$	Y	N	N	
5	N	Y	N	Y	$\boxed{\text{Y}}$	N	N	
6	N	N	Y	N	Y	$\boxed{\text{N}}$	Y	

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

Acceptor for 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
```

Acceptor for 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?

Acceptor for 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.

Models for Decidable Languages

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?

Models for Decidable Languages

Answer

There is no such model!

Models for Decidable Languages

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.

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. 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
```

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. 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!

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. 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, $A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

The Universal Language

Proposition

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

The Universal Language

Proposition

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

Proof.

We have already seen that A_{TM} is r.e.

The Universal Language

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 $L(M) = A_{\mathrm{TM}}$.

The Universal Language

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 $L(M) = A_{\mathrm{TM}}$. Consider a TM D as follows:

```
On input i
    Run M on input ⟨i, i⟩
    Output ‘‘yes’’ if i rejects i
    Output ‘‘no’’ if i accepts i
```

The Universal Language

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 $L(M) = A_{\mathrm{TM}}$. Consider a TM D as follows:

```
On input i
    Run M on input ⟨i, i⟩
    Output ‘‘yes’’ if i rejects i
    Output ‘‘no’’ if i accepts i
```

Observe that $L(D) = L_d$!

The Universal Language

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 $L(M) = A_{\mathrm{TM}}$. Consider a TM D as follows:

```
On input i
    Run M on input ⟨i, i⟩
    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. $\square$

A more complete Big Picture