CIS 770: Formal Language Theory

Pavithra Prabhakar

Kansas State University

Spring 2016

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

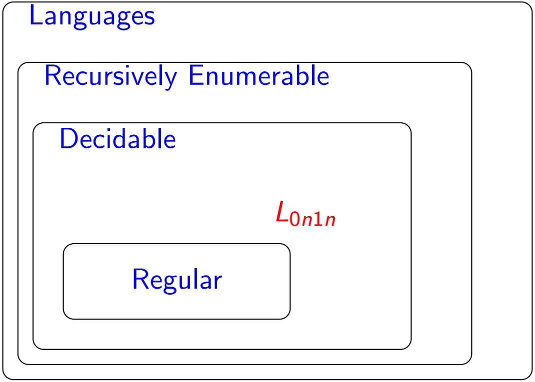
Undecidability

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 $L_d = \{M \mid M \not\in L(M)\}$. 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 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.

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., $L_d = L(j)$.

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 $j \notin L(j)$! $\square$

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.

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.

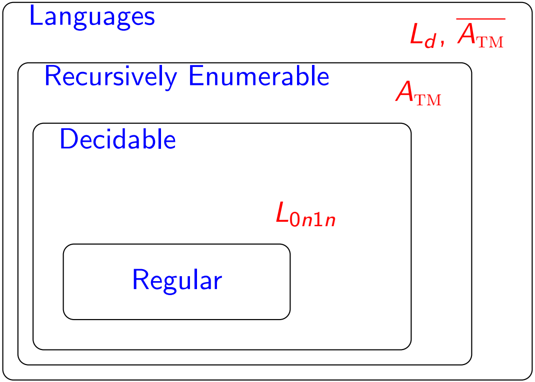
Proof.

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



Reductions

A reduction is a way of converting one problem into another problem such that a solution to the second problem can be used to solve the first problem. We say the first problem reduces to the second problem.

Reductions

A reduction is a way of converting one problem into another problem such that a solution to the second problem can be used to solve the first problem. We say the first problem reduces to the second problem.

- Informal Examples: Measuring the area of rectangle reduces to measuring the length of the sides

Reductions

A reduction is a way of converting one problem into another problem such that a solution to the second problem can be used to solve the first problem. We say the first problem reduces to the second problem.

- Informal Examples: Measuring the area of rectangle reduces to measuring the length of the sides; Solving a system of linear equations reduces to inverting a matrix

Reductions

A reduction is a way of converting one problem into another problem such that a solution to the second problem can be used to solve the first problem. We say the first problem reduces to the second problem.

- Informal Examples: Measuring the area of rectangle reduces to measuring the length of the sides; Solving a system of linear equations reduces to inverting a matrix
- The problem L_d reduces to the problem A_{TM} as follows: "To see if $w \in L_d$ check if $\langle w, w \rangle \in A_{\mathrm{TM}}$."

Undecidability using Reductions

Proposition

Suppose L_1 reduces to L_2 and L_1 is undecidable. Then L_2 is undecidable.

Undecidability using Reductions

Proposition

Suppose L_1 reduces to L_2 and L_1 is undecidable. Then L_2 is undecidable.

Proof Sketch.

Suppose for contradiction L_2 is decidable. Then there is a M that always halts and decides L_2. Then the following algorithm decides L_1

Undecidability using Reductions

Proposition

Suppose L_1 reduces to L_2 and L_1 is undecidable. Then L_2 is undecidable.

Proof Sketch.

Suppose for contradiction L_2 is decidable. Then there is a M that always halts and decides L_2. Then the following algorithm decides L_1

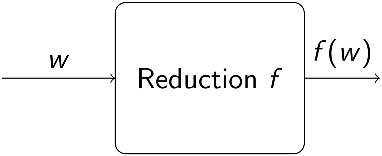
- On input w, apply reduction to transform w into an input w' for problem 2
- Run M on w', and use its answer.

Schematic View

$\xrightarrow{w}$

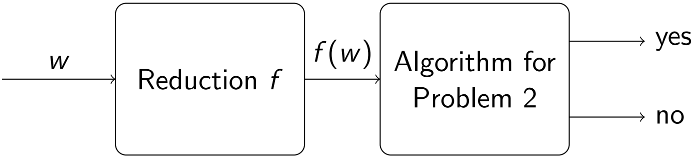
Reductions schematically

Schematic View



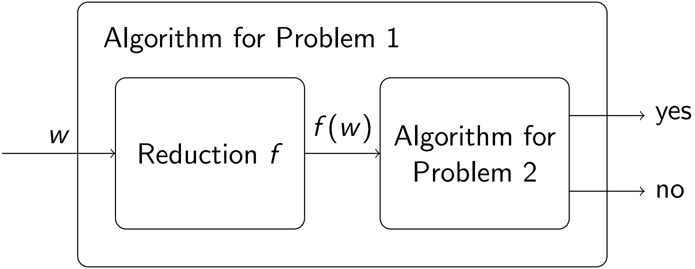
Reductions schematically

Schematic View



Reductions schematically

Schematic View



Reductions schematically

The Halting Problem

Proposition

The language $HALT = \{\langle M, w\rangle \mid M \text{ halts on input } w\}$ *is undecidable.*

The Halting Problem

Proposition

The language $HALT = \{\langle M, w\rangle \mid M \text{ halts on input } w\}$ *is undecidable.*

Proof.

We will reduce A_{TM} to HALT. Based on a machine M, let us consider a new machine $f(M)$ as follows:

The Halting Problem

Proposition

The language $HALT = \{\langle M, w\rangle \mid M \text{ halts on input } w\}$ *is undecidable.*

Proof.

We will reduce A_{TM} to HALT. Based on a machine M, let us consider a new machine $f(M)$ as follows:

```
On input x
    Run M on x
    If M accepts then halt and accept
    If M rejects then go into an infinite loop
```

The Halting Problem

Proposition

The language $HALT = \{\langle M, w\rangle \mid M \text{ halts on input } w\}$ is undecidable.

Proof.

We will reduce A_{TM} to HALT. Based on a machine M, let us consider a new machine $f(M)$ as follows:

```
On input x
    Run M on x
    If M accepts then halt and accept
    If M rejects then go into an infinite loop
```

Observe that $f(M)$ halts on input w if and only if M accepts w $\cdots\!\longrightarrow$

Proof (contd).

Suppose HALT is decidable. Then there is a Turing machine H that always halts and $L(H) = \mathrm{HALT}$.

Proof (contd).

Suppose HALT is decidable. Then there is a Turing machine H that always halts and $L(H) = \text{HALT}$. Consider the following program T

```
On input ⟨M, w⟩
    Construct program f(M)
    Run H on ⟨f(M), w⟩
    Accept if H accepts and reject if H rejects
```

Proof (contd).

Suppose HALT is decidable. Then there is a Turing machine H that always halts and $L(H) = \text{HALT}$. Consider the following program T

```
On input ⟨M, w⟩
    Construct program f(M)
    Run H on ⟨f(M), w⟩
    Accept if H accepts and reject if H rejects
```

T decides A_{TM}.

Proof (contd).

Suppose HALT is decidable. Then there is a Turing machine H that always halts and $L(H) = \text{HALT}$. Consider the following program T

```
On input ⟨M, w⟩
    Construct program f(M)
    Run H on ⟨f(M), w⟩
    Accept if H accepts and reject if H rejects
```

T decides A_{TM}. But, A_{TM} is undecidable, which gives us the contradiction. □

Mapping Reductions

Definition

A function $f : \Sigma^* \to \Sigma^*$ is computable if there is some Turing Machine M that on every input w halts with $f(w)$ on the tape.

Mapping Reductions

Definition

A function $f : \Sigma^* \to \Sigma^*$ is computable if there is some Turing Machine M that on every input w halts with $f(w)$ on the tape.

Definition

A mapping/many-one reduction from A to B is a computable function $f : \Sigma^* \to \Sigma^*$ such that

$$w \in A \text{ if and only if } f(w) \in B$$

Mapping Reductions

Definition

A function $f : \Sigma^* \to \Sigma^*$ is computable if there is some Turing Machine M that on every input w halts with $f(w)$ on the tape.

Definition

A mapping/many-one reduction from A to B is a computable function $f : \Sigma^* \to \Sigma^*$ such that

$$w \in A \text{ if and only if } f(w) \in B$$

In this case, we say A is mapping/many-one reducible to B, and we denote it by $A \leq_m B$.

Convention

In this course, we will drop the adjective “mapping” or “many-one”, and simply talk about reductions and reducibility.

Reductions and Recursive Enumerability

Proposition

If $A \leq_m B$ and B is recursively enumerable then A is recursively enumerable.

Reductions and Recursive Enumerability

Proposition

If $A \leq_m B$ and B is recursively enumerable then A is recursively enumerable.

Proof.

Let f be the reduction from A to B and let M_B be the Turing Machine recognizing B.

Reductions and Recursive Enumerability

Proposition

If $A \leq_m B$ and B is recursively enumerable then A is recursively enumerable.

Proof.

Let f be the reduction from A to B and let M_B be the Turing Machine recognizing B. Then the Turing machine recognizing A is

```
On input w
    Compute f(w)
    Run M_B on f(w)
    Accept if M_B does and reject if M_B rejects
```

□

Reductions and non-r.e.

Corollary

If $A \leq_m B$ and A is not recursively enumerable then B is not recursively enumerable.

Reductions and Decidability

Proposition

If $A \leq_m B$ and B is decidable then A is decidable.

Reductions and Decidability

Proposition

If $A \leq_m B$ and B is decidable then A is decidable.

Proof.

Let M_B be the Turing machine deciding B and let f be the reduction. Then the algorithm deciding A, on input w, computes $f(w)$ and runs M_B on $f(w)$. $\square$

Reductions and Decidability

Proposition

If $A \leq_m B$ and B is decidable then A is decidable.

Proof.

Let M_B be the Turing machine deciding B and let f be the reduction. Then the algorithm deciding A, on input w, computes $f(w)$ and runs M_B on $f(w)$. $\square$

Corollary

If $A \leq_m B$ and A is undecidable then B is undecidable.