BBM401-Lecture 8: Reducibility

Lecturer: Lale Özkahya

Resources for the presentation:
https://courses.engr.illinois.edu/cs373/fa2010/lectures

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

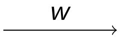
Suppose L_1 reduces to L_2 and L_1 is undecidable. Then L_2 is undecidable.

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

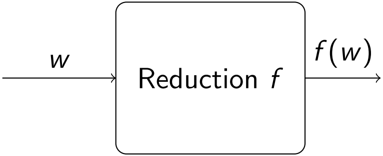
- On input w, apply reduction to transform w into an input w' for problem 2
- Run M on w', and use its answer.

Schematic View



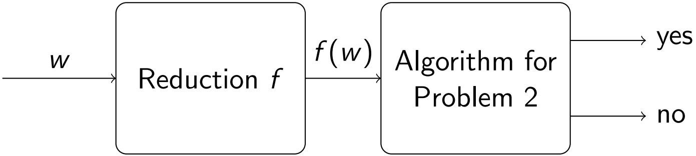
Reductions schematically

Schematic View



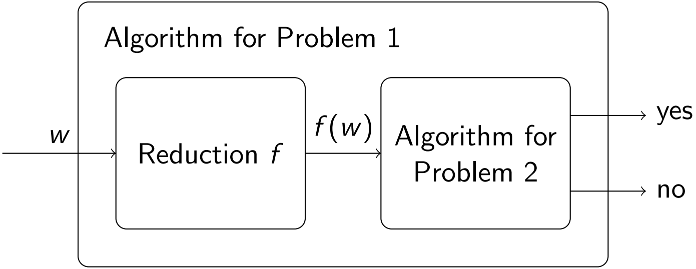
Reductions schematically

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 …→

The Halting Problem

Completing the proof

Proof (contd).

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

The Halting Problem

Completing the proof

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
```

The Halting Problem

Completing the proof

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}.

The Halting Problem

Completing the proof

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 $f : \Sigma^* \rightarrow \Sigma^*$ 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 $f(w)$. □

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 $f(w)$. □

Corollary

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

Emptiness of Turing Machines

Proposition

The language $E_{\mathrm{TM}} = \{M \mid L(M) = \emptyset\}$ *is not r.e.*

Emptiness of Turing Machines

Proposition

The language $E_{\mathrm{TM}} = \{M \mid L(M) = \emptyset\}$ *is not r.e.*

Proof.

Recall $L_d = \{M \mid M \notin L(M)\}$ is not r.e.
L_d is reducible to E_{TM} as follows.

Emptiness of Turing Machines

Proposition

The language $E_{\mathrm{TM}} = \{M \mid L(M) = \emptyset\}$ *is not r.e.*

Proof.

Recall $L_d = \{M \mid M \notin L(M)\}$ is not r.e.
L_d is reducible to E_{TM} as follows. Let $f(M) = N$ where N is a TM that behaves as follows:

On input x
    Run M on $\langle M \rangle$ for $|x|$ steps
    Accept x only if M accepts $\langle M \rangle$ within $|x|$ steps

Emptiness of Turing Machines

Proposition

The language $E_{\mathrm{TM}} = \{M \mid L(M) = \emptyset\}$ *is not r.e.*

Proof.

Recall $L_d = \{M \mid M \notin L(M)\}$ is not r.e.
L_d is reducible to E_{TM} as follows. Let $f(M) = N$ where N is a TM that behaves as follows:

```
On input x
    Run M on ⟨M⟩ for |x| steps
    Accept x only if M accepts ⟨M⟩ within |x| steps
```

Observe that $L(N) = \emptyset$ if and only if M does not accept $\langle M \rangle$

Emptiness of Turing Machines

Proposition

The language $E_{\mathrm{TM}} = \{M \mid L(M) = \emptyset\}$ *is not r.e.*

Proof.

Recall $L_d = \{M \mid M \notin L(M)\}$ is not r.e.
L_d is reducible to E_{TM} as follows. Let $f(M) = N$ where N is a TM that behaves as follows:

```
On input x
    Run M on ⟨M⟩ for |x| steps
    Accept x only if M accepts ⟨M⟩ within |x| steps
```

Observe that $L(N) = \emptyset$ if and only if M does not accept $\langle M \rangle$ if and only if $\langle M \rangle \in L_d$. □

Checking Regularity

Proposition

The language $REGULAR = \{M \mid L(M) \text{ is regular}\}$ *is undecidable.*

Checking Regularity

Proposition

The language $REGULAR = \{M \mid L(M) \text{ is regular}\}$ *is undecidable.*

Proof.

We give a reduction f from A_{TM} to REGULAR.

Checking Regularity

Proposition

The language REGULAR $= \{M \mid L(M)$ *is regular*$\}$ *is undecidable.*

Proof.

We give a reduction f from A_{TM} to REGULAR. Let $f(\langle M, w\rangle) = N$, where N is a TM that works as follows:

```
On input x
    If x is of the form 0^n1^n then accept x
    else run M on w and accept x only if M does
```

Checking Regularity

Proposition

The language REGULAR $= \{M \mid L(M)$ *is regular*$\}$ *is undecidable.*

Proof.

We give a reduction f from A_{TM} to REGULAR. Let $f(\langle M, w\rangle) = N$, where N is a TM that works as follows:

```
On input x
    If x is of the form 0^n1^n then accept x
    else run M on w and accept x only if M does
```

If $w \in L(M)$ then $L(N) =$

Checking Regularity

Proposition

The language REGULAR $= \{M \mid L(M)$ *is regular*$\}$ *is undecidable.*

Proof.

We give a reduction f from A_{TM} to REGULAR. Let $f(\langle M, w\rangle) = N$, where N is a TM that works as follows:

```
On input x
    If x is of the form 0^n 1^n then accept x
    else run M on w and accept x only if M does
```

If $w \in L(M)$ then $L(N) = \Sigma^*$.

Checking Regularity

Proposition

The language $REGULAR = \{M \mid L(M) \text{ is regular}\}$ *is undecidable.*

Proof.

We give a reduction f from A_{TM} to REGULAR. Let $f(\langle M, w\rangle) = N$, where N is a TM that works as follows:

```
On input x
    If x is of the form 0^n 1^n then accept x
    else run M on w and accept x only if M does
```

If $w \in L(M)$ then $L(N) = \Sigma^*$. If $w \notin L(M)$ then $L(N) =$

Checking Regularity

Proposition

The language REGULAR $= \{M \mid L(M)$ *is regular*$\}$ *is undecidable.*

Proof.

We give a reduction f from A_{TM} to REGULAR. Let $f(\langle M, w\rangle) = N$, where N is a TM that works as follows:

```
On input x
    If x is of the form 0^n1^n then accept x
    else run M on w and accept x only if M does
```

If $w \in L(M)$ then $L(N) = \Sigma^*$. If $w \notin L(M)$ then $L(N) = \{0^n1^n \mid n \geq 0\}$.

Checking Regularity

Proposition

The language REGULAR $= \{M \mid L(M)$ *is regular*$\}$ *is undecidable.*

Proof.

We give a reduction f from A_{TM} to REGULAR. Let $f(\langle M, w\rangle) = N$, where N is a TM that works as follows:

```
On input x
    If x is of the form 0^n1^n then accept x
    else run M on w and accept x only if M does
```

If $w \in L(M)$ then $L(N) = \Sigma^*$. If $w \notin L(M)$ then $L(N) = \{0^n 1^n \mid n \geq 0\}$. Thus, $\langle N\rangle \in$ REGULAR if and only if $\langle M, w\rangle \in A_{\mathrm{TM}}$ □

Checking Equality

Proposition

$EQ_{\mathrm{TM}} = \{\langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$ *is not r.e.*

Checking Equality

Proposition

$EQ_{\mathrm{TM}} = \{\langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$ *is not r.e.*

Proof.

We will give a reduction f from E_{TM} to $\mathrm{EQ}_{\mathrm{TM}}$.

Checking Equality

Proposition

$EQ_{\mathrm{TM}} = \{\langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$ is not r.e.

Proof.

We will give a reduction f from E_{TM} to $\mathrm{EQ}_{\mathrm{TM}}$. Let M_1 be the Turing machine that on any input, halts and rejects

Checking Equality

Proposition

$EQ_{\text{TM}} = \{\langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$ is not r.e.

Proof.

We will give a reduction f from E_{TM} to EQ_{TM}. Let M_1 be the Turing machine that on any input, halts and rejects i.e., $L(M_1) = \emptyset$. Take $f(M) = \langle M, M_1\rangle$.

Checking Equality

Proposition

$EQ_{\mathrm{TM}} = \{\langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$ is not r.e.

Proof.

We will give a reduction f from E_{TM} to $\mathrm{EQ}_{\mathrm{TM}}$. Let M_1 be the Turing machine that on any input, halts and rejects i.e., $L(M_1) = \emptyset$. Take $f(M) = \langle M, M_1\rangle$.
Observe $M \in E_{\mathrm{TM}}$ iff $L(M) = \emptyset$ iff $L(M) = L(M_1)$ iff $\langle M, M_1\rangle \in \mathrm{EQ}_{\mathrm{TM}}$. □