Theory of Computation
Reducibility

Dimitris Diochnos
School of Computer Science
University of Oklahoma

Outline

Table of Contents

Introduction

Oftentimes we reduce the solution of one problem to the solution of another problem.

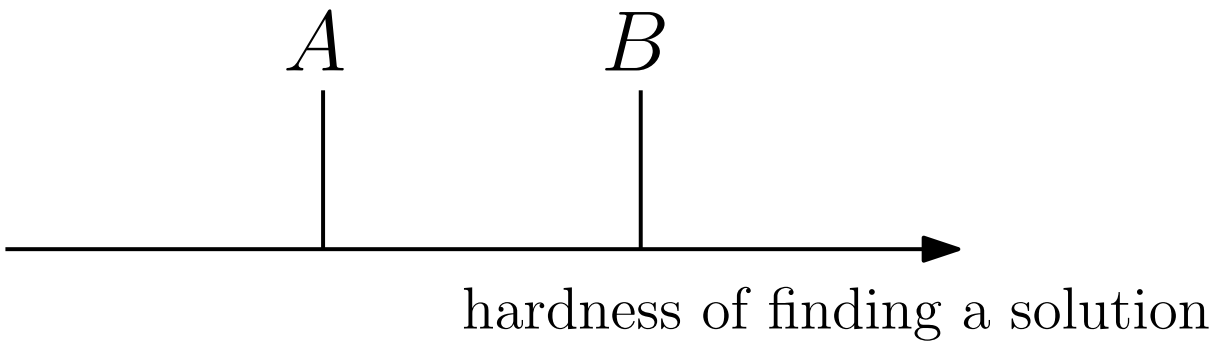
For example,

measure the area of a rectangle $\xrightarrow{\text{reduces to}}$

measure the length and height of a rectangle

Introduction

"Problem A is reducible to Problem B" implies:



- A solution to B provides a solution to A in this case. So A can not be harder than B.
- Conversely: If problem A is "hard" and A is reducible to B, then B is also "hard" (or "harder"...)

Table of Contents

Undecidable Problems from Language Theory

Theorem 1

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input string } w\}$$

is undecidable.

Undecidable Problems from Language Theory

Theorem 1

$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input string } w\}$$

is undecidable.

Proof.

Towards contradiction, assume HALT_{TM} is decidable.
Then, there is a TM R that decides HALT_{TM}.
We now reduce A_{TM} to HALT_{TM}:
Let S be a TM such that: "on input $\langle M, w\rangle$, where M is a TM and w is a string, does the following:

1. Run R on input $\langle M, w\rangle$
   If "no" (i.e., $\langle M, w\rangle$ rejected) $\Rightarrow$ M loops forever on $w \Rightarrow$ **reject**.
2. If "yes" from (1), then run M on w and return whatever M returns."
   (Impossible to have such an S because now we decide A_{TM}) □

Undecidable Problems from Language Theory

Theorem 2

$$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } M \text{ and } L(M) = \emptyset\}$$

is undecidable.

Undecidable Problems from Language Theory

Theorem 2

$$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } M \text{ and } L(M) = \emptyset\}$$

is undecidable.

Proof (to be cont.)

Towards contradiction, assume E_{TM} is decidable by some TM R.
We want to reduce A_{TM} to E_{TM}.
(Technically we will reduce A_{TM} to $\overline{\mathrm{E}_{TM}}$ since M accepts if $L(M) \neq \emptyset$.)
For the critical input $\langle M, w \rangle$ of A_{TM}, construct another TM M_1:
$M_1 =$ "on input x:

1. If $x \neq w$, **reject**.
2. If $x = w$, run M on input w and accept if M accepts." (cont'd $\rightarrow$)

Undecidable Problems from Language Theory

Proof (cont.)

So, $L(M_1)$ is either $\emptyset$ or $\{w\}$.
Check with R which case is true.
Construct another TM S such that:
$S =$ "on input $\langle M, w \rangle$, an encoding of a TM M and a string w:

1. Use the description of M and w to construct the TM M_1.
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, **reject**,
   if R rejects, **accept**."

In other words, S is now a decider for A_{TM}.
This is impossible, so we reach a contradiction. □

Undecidable Problems from Language Theory

Theorem 3

$$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

is undecidable.

Undecidable Problems from Language Theory

Theorem 3

$$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

is undecidable.

Proof.

Towards contradiction, assume EQ_{TM} is decidable by some TM R.
Reduce E_{TM} to EQ_{TM}. The idea is

- to create a TM M_1 that rejects all inputs, and
- use R to decide $\langle M, M_1\rangle$ for some arbitrary M, thus solving E_{TM} (which is impossible).

$S =$ "on input $\langle M\rangle$, where M is a TM:

1. Run R on input $\langle M, M_1\rangle$, where M_1 is a TM that rejects all inputs.
2. If R accepts, **accept**. If R rejects, **reject**."

$\Rightarrow$ So S decides $\mathrm{E}_{TM} \Rightarrow$ contradiction. □

Table of Contents

The Post Correspondence Problem

We have dominoes that look like:

$$\left[\frac{a}{ab}\right]$$

A collection of dominoes looks like:

$$\left\{\left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right]\right\}$$

The task is to make a list of these dominoes (repetitions permitted) so that we get the same string at the top and the bottom.
Such a list is called a **match**. For example,

$$\left[\frac{a}{ab}\right], \left[\frac{b}{ca}\right], \left[\frac{ca}{a}\right], \left[\frac{a}{ab}\right], \left[\frac{abc}{c}\right]$$

The Post Correspondence Problem

Theorem 4

$PCP = \{\langle P \rangle \mid P \text{ is an instance of the Post correspondence problem with a match}\}$

is undecidable.

Proof idea.

Reduction from A_{TM} to PCP via accepting computation histories. □

Rice's Theorem

Problem 5 (Rice's Theorem)

Let P be a property about the language recognized by a Turing machine such that:

(i) *P is non-trivial. Some, but not all Turing machines recognize a language that has this property,*

(ii) *whenever $L(M_1) = L(M_2)$ for two Turing machines M_1 and M_2, then both $L(M_1)$ and $L(M_2)$ have the property P (or none of them has the property P),*

then the language
$L_P = \{\langle M \rangle \mid M \text{ is a TM such that } L(M) \text{ recognized by } M \text{ has property } P\}$ *is undecidable.*

Rice's Theorem

Proof idea (to be cont.)

Towards contradiction, suppose L_P is decidable by some TM R.

- By condition (i), since P is a non-trivial property for some language, there are TM $M_P, M_{\neg P}$ such that:
  $$\begin{cases} L(M_P) & \text{has the property } P \\ L(M_{\neg P}) & \text{does not have the property } P \end{cases}$$
  We assume that we have access to such TMs.
- We will now reduce A_{TM} to L_P.
  Eventually our argument depends on whether or not the empty language $L_\emptyset = \emptyset$ has this property P or not.
  We construct a TM S such that:
  $$L(S) = \begin{cases} L_\emptyset, & \text{if } M \text{ does not accept } w \\ \begin{cases} L(M_P), & \text{if } M \text{ accepts } w \text{ and } L_\emptyset \text{ doesn't have property } P \\ L(M_{\neg P}), & \text{if } M \text{ accepts } w \text{ and } L_\emptyset \text{ does have the property } P \end{cases} \end{cases}$$

Rice's Theorem

Proof idea (cont.)

- We feed the description of S, $\langle S \rangle$, into R
  $\Rightarrow$ R tells us if $L(S)$ has the property
  $\Rightarrow$ Based on the relationship of $L(S)$ with $L_\emptyset$ we can tell if M accepts w
  $\Rightarrow$ **Contradiction**
- Case where $L_\emptyset = \emptyset$ does not have property P
  $S =$ "on input x:
  1. For the moment ignore x and simulate M on input w, until M accepts so that we can go to step 2.
     (So, if M rejects, loop and try again, ...forever ...)
  2. (Reached this point because M accepted w)
     Simulate M_P on x, if $L_\emptyset$ does not have the property P.
     Otherwise,
     Simulate $M_{\neg P}$ on x ($L_\emptyset$ has the property P this time).
     Accept x, if the simulation by M_P (or $M_{\neg P}$) accepted."

Rice's Theorem

Proof idea (cont.)

In other words:

M accepts $w \Rightarrow L(S) = L(M_P)$ (or $L(S) = L(M_{\neg P})$)

M does not accept $w \Rightarrow L(S) = L_\emptyset$

- Feed $\langle S \rangle$ into R

$$R(\langle S \rangle) = \begin{cases} \text{accept} & \text{if } L_\emptyset \text{ has property } P, \text{ then } L(S) = L_\emptyset \\ & \Rightarrow M \text{ did not accept } w \\ & \Rightarrow \text{For the Problem } A_{TM} \text{ answer } \textbf{reject} \\ & \quad \text{on input } \langle M, w \rangle \\ \text{reject} & \text{if } L_\emptyset \text{ does not have property } P, \text{ then } L(S) = L(M_P) \\ & \Rightarrow M \text{ accepted } w \\ & \Rightarrow \text{For the Problem } A_{TM} \text{ answer } \textbf{accept} \\ & \quad \text{on input } \langle M, w \rangle \end{cases}$$

Table of Contents

Mapping Reducibility

Definition 6

A function $f : \Sigma^* \to \Sigma^*$ is a **computable function** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Example 7

All usual arithmetic operations on integers are computable functions.
For example, addition: Input $\langle m, n \rangle$, output $\langle m + n \rangle$.

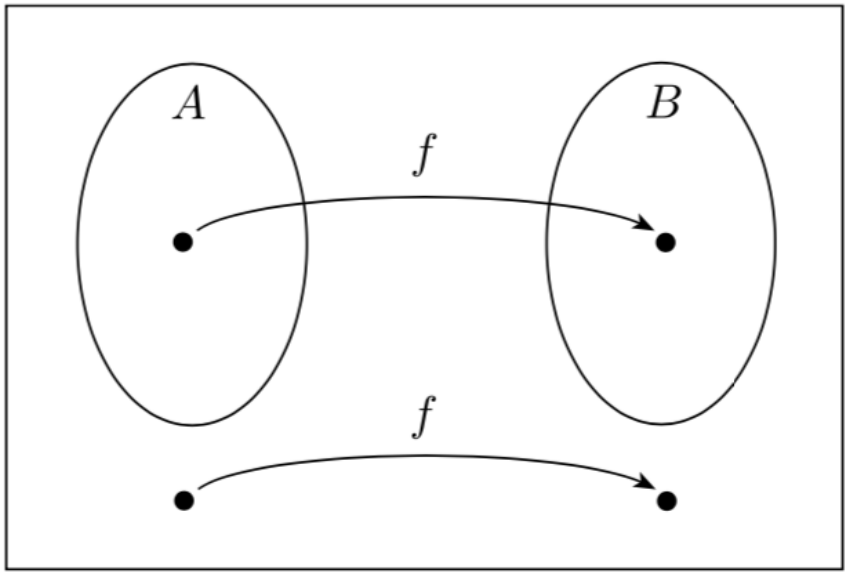
Mapping Reducibility

Definition 8

A language A is mapping reducible to language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every w,

$$w \in A \Leftrightarrow f(w) \in B$$

The function f is called the **reduction** of A to B.



Function f reducing A to B

As usual, computational problems are represented as languages.

Mapping Reducibility

Theorem 9

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

Proof.

Let M be a decider for B. Let f be the reduction from A to B.
Construct a decider N for A as follows.
$N =$ "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

$$w \in A \Rightarrow f(w) \in B \quad \Rightarrow M \text{ accepts } f(w) \text{ whenever } w \in A$$
$$\Rightarrow N \text{ works as expected.}$$

Mapping Reducibility

Corollary 10

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

Mapping Reducibility

Example 11

In Theorem 1 we reduced A_{TM} to $HALT_{TM}$.
We need a function

$$f : \langle M, w \rangle \in A_{TM} \longrightarrow \langle M', w' \rangle \in \text{HALT}_{TM}$$

where $\langle M, w \rangle \in A_{TM}$ if and only if $\langle M', w' \rangle \in \text{HALT}_{TM}$.
$F =$ "On input $\langle M, w \rangle$:

1. Construct the TM M':
   $M' =$ "On input x:
   1. Run M on x.
   2. If M accepts, accept.
   3. If M rejects, enter a loop."
2. Output $\langle M', w \rangle$."

Mapping Reducibility

Remark 1 (malformed inputs)

When we describe a TM that computes a reduction from A to B, improperly formed inputs are assumed to map to strings outside of B.

Example 12

In Theorem 4 we reduced E_{TM} to EQ_{TM}. There the reduction f maps $\langle M \rangle$ to $\langle M, M_1 \rangle$, where M_1 is the machine that rejects all inputs.

Example 13

In Theorem 2 we reduced A_{TM} to $\overline{E_{TM}}$, since M accepts w **iff** $L(M_1)$ is **not** empty. So f is a mapping reduction from A_{TM} to $\overline{E_{TM}}$.
In fact, no reduction from A_{TM} to E_{TM} exists.

Mapping Reducibility

Theorem 14

If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable.

Proof.

Same as in Theorem 9, except that M and N are recognizers instead of deciders. □

Mapping Reducibility

Corollary 15

If $A \leq_m B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Typical application of Corollary 15:

- A mapping reduction $f : A \leq_m B$ is also a mapping reduction $f : \overline{A} \leq_m \overline{B}$.
- So, since we know that $\overline{A_{TM}}$ is not Turing-recognizable, we can show that **a problem** P is **not Turing-recognizable** by designing a mapping reduction $f : A_{TM} \leq_m \overline{P}$. (This will also imply $\overline{A_{TM}} \leq_m P$.)

Mapping Reducibility

Theorem 16

$EQ_{TM} = \{\langle M_1, M_2\rangle \mid M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\}$ is neither Turing-recognizable nor co-Turing-recognizable.

Proof (to be cont.)

Part A: "EQ_{TM} is not Turing-recognizable"

A_{TM} reduces to $\overline{EQ_{TM}}$.
$F =$ "On input $\langle M, w\rangle$, where M is a TM and w is a string:

1. Construct the following two machines, M_1 and M_2:
   $M_1 =$ "On any input, **reject**."
   $M_2 =$ "On any input:
   Run M on w. If M accepts, **accept**."
2. Output $\langle M_1, M_2\rangle$."

Mapping Reducibility

Proof (to be cont.)

Here, M_1 accepts nothing.
$\{\Sigma^*\} = L(M_2) \neq L(M_1) = \emptyset$, if M accepts w.
$L(M_2) = L(M_1) = \emptyset$, if M does not accept w.

So, M_1 and M_2 are **equivalent** in the second case
$\Rightarrow F$ reduces A_{TM} to $\overline{EQ_{TM}}$, as desired.

Mapping Reducibility

Proof (to be cont.)

Part B: "$\overline{EQ_{TM}}$ is not Turing-recognizable"

A_{TM} reduces to EQ_{TM}.
$G =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Construct the following two machines, M_1 and M_2:
   $M_1 =$ "On any input, **accept**."
   $M_2 =$ "On any input:
   Run M on w. If M accepts, **accept**."
2. Output $\langle M_1, M_2 \rangle$."

Mapping Reducibility

Proof (cont.)

Here, M_1 accepts everything.
This time:
M accepts $w \Rightarrow L(M_2) = L(M_1)$
M does not accept $w \Rightarrow L(M_2) \neq L(M_1)$

So, M_1 and M_2 are **equivalent** in the first case (where M accepts w)
$\Rightarrow$ G reduces A_{TM} to EQ_{TM}, as desired. □

Table of Contents

Turing Reducibility

Definition 17

An **oracle** for a language B is an external device that is capable of reporting whether any string w is a member of B.

An **oracle Turing machine** is a modified Turing machine that has the additional capability of querying an oracle. We write M^B to describe an oracle Turing machine that has an oracle for language B.

Turing Reducibility

Example 18

An oracle Turing machine with an oracle for A_{TM} can decide more languages than an ordinary Turing machine can.
Apart from deciding A_{TM} (obviously), we can also decide, eg., E_{TM}, the emptiness testing problem for TMs by using $T^{A_{TM}}$ below.

$T^{A_{TM}}$ = "On input $\langle M \rangle$, where M is a TM:

1. Construct the following TM N:
   N = "On any input x:
   1. Run M in parallel on all strings in Σ^*.
   2. If M accepts any of these strings, accept x."
2. Query the oracle for A_{TM} to determine whether $\langle N, 0 \rangle \in A_{TM}$.
3. If the oracle answers NO, **accept**; otherwise, **reject**."

Turing Reducibility

Example 18 (cont.)

- So, if $L(M) \neq \emptyset \Rightarrow L(N) = \{w \in \Sigma^*\} \Rightarrow 0 \in L(N) \Rightarrow$ Oracle "*Yes*" $\Rightarrow$ reject.
- Conversely, if $L(M) = \emptyset \Rightarrow L(N) = \emptyset \Rightarrow 0 \notin L(N) \Rightarrow$ Oracle "*No*" $\Rightarrow$ accept.

In other words, $T^{A_{TM}}$ decides E_{TM}.

We say that E_{TM} is **decidable relative to** A_{TM}.

Turing Reducibility

Definition 19

Language A is **Turing reducible** to language B, written $A \leq_T B$, if A is decidable relative to B.

Theorem 20

If $A \leq_T B$ and B is decidable, then A is decidable.

Proof.

B is decidable $\Rightarrow \exists$ TM M that decides $B \Rightarrow$ Use M instead of an oracle for B in order to solve B and subsequently A. $\square$