Reducibility, Time complexity

204213 Theory of Computation

Jittat Fakcharoenphol

Kasetsart University

January 27, 2009

Outline

Undecidable languages from undecidability of A_{TM}

- If given that language S is decidable, one can show that A_{TM} is also decidable, we can conclude that S is also undecidable.

Undecidable languages from undecidability of A_{TM}

- If given that language S is decidable, one can show that A_{TM} is also decidable, we can conclude that S is also undecidable. (why?)

Undecidable languages from undecidability of A_{TM}

- If given that language S is decidable, one can show that A_{TM} is also decidable, we can conclude that S is also undecidable. (why?)
- This general technique is called **reduction**.

Reduction: informally

- "Reduce problem A **to** problem B": show how to solve A using a TM for B.

Reduction: informally

- "Reduce problem A **to** problem B": show how to solve A using a TM for B.
- If we can do that, and:
  - If B is decidable, A is also decidable. (why?)

Reduction: informally

- "Reduce problem A **to** problem B": show how to solve A using a TM for B.
- If we can do that, and:
  - If B is decidable, A is also decidable. (why?)
  - **If A is undecidable, B is also undecidable.**

Halting problem

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

Theorem 1

$HALT_{TM}$ is undecidable.

Proof idea: Halting problem

- Since our goal is to show that $HALT_{TM}$ is undecidable, we should show that we can solve some undecidable language by a TM that uses a TM for $HALT_{TM}$ as a subroutine.

Proof idea: Halting problem

- Since our goal is to show that $HALT_{TM}$ is undecidable, we should show that we can solve some undecidable language by a TM that uses a TM for $HALT_{TM}$ as a subroutine.
- Note that if we can determine if a TM M halts on w, we can combine it with a recognizer for A_{TM} to get a decider.

Proof: Halting problem

Proof.

- We'll prove by reducing A_{TM} to $HALT_{TM}$.
- Assume that $HALT_{TM}$ is decidable; thus, there exists a TM R that decides $HALT_{TM}$.

Proof: Halting problem

Proof.

- We'll prove by reducing A_{TM} to $HALT_{TM}$.
- Assume that $HALT_{TM}$ is decidable; thus, there exists a TM R that decides $HALT_{TM}$.
- We can construct a TM S that decides A_{TM}:
  $S =$ "On input $\langle M, w\rangle$,
  1. Run R on $\langle M, w\rangle$; if R rejects, REJECT.
  2. If R accepts, simulate M on w until it halts.
  3. If M accepts, ACCEPT; otherwise, REJECT."

Proof: Halting problem

Proof.

- We'll prove by reducing A_{TM} to $HALT_{TM}$.
- Assume that $HALT_{TM}$ is decidable; thus, there exists a TM R that decides $HALT_{TM}$.
- We can construct a TM S that decides A_{TM}:
  $S =$ "On input $\langle M, w\rangle$,
  1. Run R on $\langle M, w\rangle$; if R rejects, REJECT.
  2. If R accepts, simulate M on w until it halts.
  3. If M accepts, ACCEPT; otherwise, REJECT."
- Since A_{TM} is undecidable, we can conclude that $HALT_{TM}$ is also undecidable.

$\square$

Emptiness

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

Theorem 2

E_{TM} is undecidable.

Proof idea: emptiness

We'll have to solve either A_{TM} or $HALT_{TM}$ by solving E_{TM}.

Proof idea: emptiness

We'll have to solve either A_{TM} or $HALT_{TM}$ by solving E_{TM}.
It may help to think about "how" to distinguish between accepting a string and not accepting that string by "some TM" that distinguishes between accepting nothing and accepting something.

Proof: Emptiness problem (1)

Proof.

- We'll prove by reducing A_{TM} to E_{TM}.
- Assume that E_{TM} is decidable; thus, there exists a TM R that decides E_{TM}.

Proof: Emptiness problem (1)

Proof.

- We'll prove by reducing A_{TM} to E_{TM}.
- Assume that E_{TM} is decidable; thus, there exists a TM R that decides E_{TM}.
- We'll construct another machine M_1 in such a way that:
  - If M accepts w, $L(M_1) \neq \emptyset$, and
  - If M does not accept w, $L(M_1) = \emptyset$.

Proof: Emptiness problem (1)

Proof.

- We'll prove by reducing A_{TM} to E_{TM}.
- Assume that E_{TM} is decidable; thus, there exists a TM R that decides E_{TM}.
- We'll construct another machine M_1 in such a way that:
  - If M accepts w, $L(M_1) \neq \emptyset$, and
  - If M does not accept w, $L(M_1) = \emptyset$.
- $M_1 =$ "On input x,
  1. If $x \neq w$, REJECT.
  2. If $x = w$, run M on input w and ACCEPT if M does."

Proof: Emptiness problem (1)

Proof.

- We'll prove by reducing A_{TM} to E_{TM}.
- Assume that E_{TM} is decidable; thus, there exists a TM R that decides E_{TM}.
- We'll construct another machine M_1 in such a way that:
  - If M accepts w, $L(M_1) \neq \emptyset$, and
  - If M does not accept w, $L(M_1) = \emptyset$.
- $M_1 =$ "On input x,
  1. If $x \neq w$, REJECT.
  2. If $x = w$, run M on input w and ACCEPT if M does."
- Can you fill the rest of the proof?

□

Regular languages

Let $REGULAR_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$

Theorem 3

$REGULAR_{TM}$ is undecidable.

Proof idea: $REGULAR_{TM}$

Again, given M, we'll build another TM M_2 such that if M accept w, M_2 will accept a regular language, and M_2 will accept non-regular language otherwise.

Proof: $REGULAR_{TM}$ is undecidable

Proof.

- Assume that $REGULAR_{TM}$ is decidable; thus, there exists a TM R that decides $REGULAR_{TM}$.

Proof: $REGULAR_{TM}$ is undecidable

Proof.

- Assume that $REGULAR_{TM}$ is decidable; thus, there exists a TM R that decides $REGULAR_{TM}$.
- We'll build a TM S that decides A_{TM} as follows:
  $S =$ "On input $\langle M, w \rangle$,
  1. Construct the following TM M_2:
     $M_2 =$ "On input x:
     1. If x has the form $0^n 1^n$, ACCEPT.
     2. If not, run M on w, and ACCEPT iff M accepts w"

Proof: $REGULAR_{TM}$ is undecidable

Proof.

- Assume that $REGULAR_{TM}$ is decidable; thus, there exists a TM R that decides $REGULAR_{TM}$.
- We'll build a TM S that decides A_{TM} as follows:
  $S =$ "On input $\langle M, w \rangle$,
  1. Construct the following TM M_2:
     $M_2 =$ "On input x:
     1. If x has the form $0^n 1^n$, ACCEPT.
     2. If not, run M on w, and ACCEPT iff M accepts w"
  2. Run R on input $\langle M_2 \rangle$.
  3. If R accepts, ACCEPT; if R rejects, REJECT.

$\square$

Notes

- It is not hard to turn the previous proof to show that determining if a TM recognizes a CFL or a decidable language is undecidable.

Equivalence

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

Theorem 4

EQ_{TM} is undecidable.

Proof idea: EQ_{TM}

Note that the reduction can be from any undecidable languages.
For this one, it'll be easier to do the reduction from E_{TM} to EQ_{TM}.

Proof: EQ_{TM}

Proof.

- Assume that EQ_{TM} is decidable; thus, there exists a TM R that decides EQ_{TM}.

Proof: EQ_{TM}

Proof.

- Assume that EQ_{TM} is decidable; thus, there exists a TM R that decides EQ_{TM}.
- We'll build a TM S that decides E_{TM} as follows:
  $S =$ "On input $\langle M \rangle$,
  1. Run R on input $\langle M, M' \rangle$, where M' is a TM that rejects every string.
  2. If R accepts, ACCEPT; if R rejects, REJECT.

$\square$

The Post Correspondence Problem (PCP)

Formalizing reducibility

Informally, we can reduce problem A to problem B if we can use TM for B to solve A.

Formalizing reducibility

Informally, we can reduce problem A to problem B if we can use TM for B to solve A.
"Use": can have many meaning.

Formalizing reducibility

Informally, we can reduce problem A to problem B if we can use TM for B to solve A.
"Use": can have many meaning.
We'll formalize it (in one way) using the notion of **mapping reducibility**.

Formalizing reducibility

Informally, we can <u>reduce</u> problem A to problem B if we can use TM for B to solve A.
"Use": can have many meaning.
We'll formalize it (in one way) using the notion of **mapping reducibility**. In essence, this means that there is a "computable" function that takes an instance of A to an instance of B.

Formalizing reducibility

Informally, we can <u>reduce</u> problem A to problem B if we can use TM for B to solve A.
"Use": can have many meaning.
We'll formalize it (in one way) using the notion of **mapping reducibility**. In essence, this means that there is a "computable" function that takes an instance of A to an instance of B.
That function is called a **reduction**.

Computable functions

Definition

A function $f : \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some TM M on every input w halts with only $f(w)$ remaining on the tape.

Mapping reducibility

Definition

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

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

Mapping reducibility

Definition

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

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

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

$$A \leq_m B.$$

Theorem 5

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

Theorem 5

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

Can you prove it?

Corollary 6

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

Let's go back to see our previous reductions

Let's go back to see our previous reductions

Note that except the proof that shows that E_{TM} is undecidable every reductions are mapping reduction.

Let's go back to see our previous reductions

Note that except the proof that shows that E_{TM} is undecidable every reductions are mapping reduction.
It turns out that no mapping reduction from A_{TM} to E_{TM} exists.

Time complexity

It's good that a problem is decidable, but sometimes that's not good enough.

Time complexity

It's good that a problem is decidable, but sometimes that's not good enough.
Think about an algorithm that runs for 200 years.

Time complexity

It's good that a problem is decidable, but sometimes that's not good enough.

Think about an algorithm that runs for 200 years.

Therefore, we'll try to distinguish between easy problems and hard problems.

Time complexity

It's good that a problem is decidable, but sometimes that's not good enough.
Think about an algorithm that runs for 200 years.
Therefore, we'll try to distinguish between easy problems and hard problems. (Well, not very successfully though.)

Let's talk about "time"

Consider the following language

$$A = \{0^k 1^k \mid k \geq 0\}.$$

Can you describe a TM that decides A?

Let's talk about "time"

Consider the following language

$$A = \{0^k 1^k \mid k \geq 0\}.$$

Can you describe a TM that decides A?
How "fast" can it run?

M_1 for A

$M1 =$ "On input string w:

1. Scan across the tape and reject if 0 is found to the right of 1.
2. Rescan if both 0's and 1's remain on the tape.
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If neither 0 nor 1 remains, accept. Otherwise, reject."

Terminology

- worst-case analysis, average-case analysis
- running time, time complexity
- asymptotic notations: big-O, little-O

Running time for M_1

$M1 =$ "On input string w:

1. Scan across the tape and reject if 0 is found to the right of 1.
2. Rescan if both 0's and 1's remain on the tape.
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If neither 0 nor 1 remains, accept. Otherwise, reject."

Running time for M_1

$M1 =$ "On input string w:

1. Scan across the tape and reject if 0 is found to the right of 1.
2. Rescan if both 0's and 1's remain on the tape.
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If neither 0 nor 1 remains, accept. Otherwise, reject."

- Let n denote the length of the input.
- First stage takes $2n = O(n)$ time.

Running time for M_1

$M1 =$ "On input string w:

1. Scan across the tape and reject if 0 is found to the right of 1.
2. Rescan if both 0's and 1's remain on the tape.
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If neither 0 nor 1 remains, accept. Otherwise, reject."

- Let n denote the length of the input.
- First stage takes $2n = O(n)$ time.
- Each time the TM works on stages 2 and 3, it takes $O(n)$ time.

Running time for M_1

$M1 =$ "On input string w:

1. Scan across the tape and reject if 0 is found to the right of 1.
2. Rescan if both 0's and 1's remain on the tape.
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If neither 0 nor 1 remains, accept. Otherwise, reject."

- Let n denote the length of the input.
- First stage takes $2n = O(n)$ time.
- Each time the TM works on stages 2 and 3, it takes $O(n)$ time. Each time 2 symbols are crossed off.

Running time for M_1

$M1 =$ "On input string w:

1. Scan across the tape and reject if 0 is found to the right of 1.
2. Rescan if both 0's and 1's remain on the tape.
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If neither 0 nor 1 remains, accept. Otherwise, reject."

- Let n denote the length of the input.
- First stage takes $2n = O(n)$ time.
- Each time the TM works on stages 2 and 3, it takes $O(n)$ time. Each time 2 symbols are crossed off. Thus, this two stages is repeated at most $n/2$ times, with the total time of $(n/2)O(n) = O(n^2)$.

Running time for M_1

$M1$ = "On input string w:

1. Scan across the tape and reject if 0 is found to the right of 1.
2. Rescan if both 0's and 1's remain on the tape.
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If neither 0 nor 1 remains, accept. Otherwise, reject."

- Let n denote the length of the input.
- First stage takes $2n = O(n)$ time.
- Each time the TM works on stages 2 and 3, it takes $O(n)$ time. Each time 2 symbols are crossed off. Thus, this two stages is repeated at most $n/2$ times, with the total time of $(n/2)O(n) = O(n^2)$.
- Last stage, the TM scan the input. This takes $O(n)$ time.
- Thus, the total time M_1 on an input of length n is $O(n) + O(n^2) + O(n) = O(n^2)$

Can we do that faster?

Can we do that faster?

Yes.

M_2 for A

The idea is that, instead of crossing only one symbols, we cross of half of the symbols.

M_3 with two tape

We can even do better if we have 2-tape TM.

Time complexity class

Definition

Let $t : \mathcal{N} \rightarrow \mathcal{R}^+$ be a function. Define the **time complexity class**, $TIME(t(n))$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

Language A

On input of length n,

- M_1 runs in time $O(n^2)$.

Language A

On input of length n,

- M_1 runs in time $O(n^2)$.
- M_2 runs in time $O(n \log n)$. It can be shown that no single-tape TM runs faster than this.

Language A

On input of length n,

- M_1 runs in time $O(n^2)$.
- M_2 runs in time $O(n \log n)$. It can be shown that no single-tape TM runs faster than this.
- M_3, with 2 tapes, runs in $O(n)$.

Language A

On input of length n,

- M_1 runs in time $O(n^2)$.
- M_2 runs in time $O(n \log n)$. It can be shown that no single-tape TM runs faster than this.
- M_3, with 2 tapes, runs in $O(n)$.
    - The time complexity of A depends on the computational model.

Multitape TM's and single-tape TM's

Theorem 7

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape TM has a equivalent $O(t^2(n))$-time single-tape TM.

Proof