INT201 Decision, Computation and Language

Lecture 12 –Reducibility

Dr Yushi Li

Xi'an Jiaotong-Liverpool University
西交利物浦大學

Reducibility

Definition

Reduction is a way of converting one problem to another problem, so that the solution to the second problem can be used to solve the first problem.

If A **reduces** to B, then any solution of B solves A (Reduction always involves two problems, A and B).

- If A is reducible to B, then A cannot be harder than B.
- If A is reducible to B and B is decidable, then A is also decidable.
- If A is reducible to B and A is undecidable, then B is also undecidable.

Reducibility

A common strategy for proving that a language L is undecidable is by reduction method, proceeding as follows:

Typical approach to show L is undecidable via reduction from A to L:

- Find a problem A known to be undecidable
- Suppose L is decidable.
- Let R be a TM that decides L.
- Using R as subroutine to construct another TM S that decides A.
- But A is not decidable.
- Conclusion: L is not decidable.

Computation histories

Definition

An **accepting computation history** for a TM M on a string w is a sequence of configurations

$$C_1, C_2, \ldots, C_k$$

for some $k \geq 1$ such that the following properties hold:

1. C_1 is the start configuration of M on w.
2. Each C_j yields C_{j+1}.
3. C_k is an accepting configuration.

A **rejecting computation history** for M on w is the same except last configuration C_k is a rejecting configuration of M.

Computation histories

Accepting and rejecting computation histories are finite.

If M does not halt on w,

- then no accepting or rejecting computation history exists.

Useful for both

- deterministic TMs (one history)
- nondeterministic TMs (many histories).

"$\langle M, w \rangle \notin A_{TM}$" is equivalent to

- "there is no accepting computation history $C_1, C_2, \ldots, C_k$ for M on w"
- "all histories $C_1, C_2, \ldots, C_k$ are non-accepting ones for M on w".

Computable functions

Suppose we have 2 languages A and B, where

- A is defined over alphabet Σ_1, so $\mathrm{A} \subseteq \Sigma_1^*$
- B is defined over alphabet Σ_2, so $\mathrm{B} \subseteq \Sigma_2^*$

Informally speaking, A is reducible to B if we can use a "black box" for B to build an algorithm for A.

Computable functions

Definition

A function

$$f : \Sigma_1^* \rightarrow \Sigma_2^*$$

is a **computable function** if some TM M, on every input $w \in \Sigma_1^*$ halts with just $f(w)$ $\in \Sigma_2^*$ on its tape. (there exists a TM can compute this function)

One useful class of computable functions transforms one TM into another.

Example

All the usual integer computations are computable:

Addition, multiplication, sorting, etc.

Mapping Reducibility

Definition

Suppose that A and B are two languages

- A is defined over alphabet Σ_1^*, so $\mathrm{A} \subseteq \Sigma_1^*$
- B is defined over alphabet Σ_2^*, so $\mathrm{B} \subseteq \Sigma_2^*$

Then A is **mapping reducible** to B, written

$$\mathrm{A} \leq_{\mathrm{m}} \mathrm{B}$$

if there is a computable function

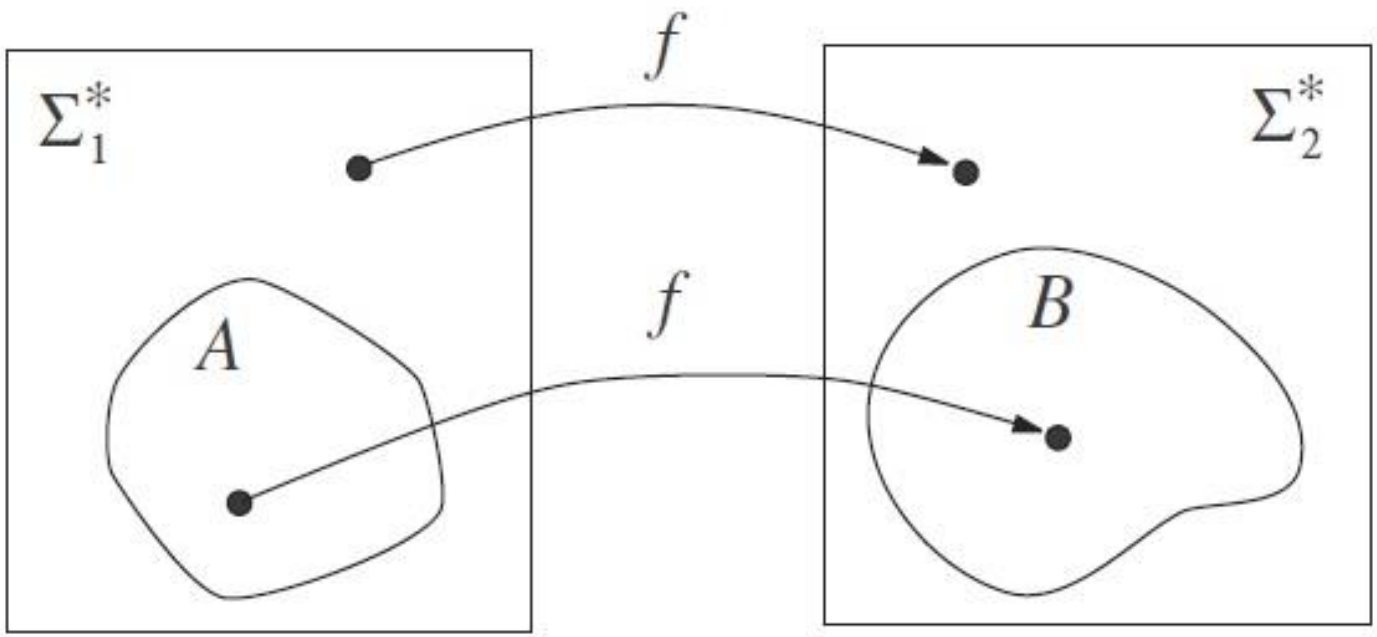
$$\mathrm{f} : \Sigma_1^* \rightarrow \Sigma_2^*$$

such that, for every $\mathrm{w} \in \Sigma_1^*$

$$\mathrm{w} \in \mathrm{A} \Leftrightarrow \mathrm{f}(\mathrm{w}) \in \mathrm{B}$$

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

Mapping Reduction



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

YES instance for problem $B \Longleftrightarrow$ YES instance for problem B

Theorem

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

Corollary

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

Theorem Proof

Theorem Proof

Theorem

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

Corollary

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

Theorem Proof

Theorem Proof

Algorithm and Information

Algorithm is independent of computation model

All reasonable variants of TM models are equivalent to TM:

- k-tape TM
- nondeterministic TM
- enumerator
- random-access TM: head can jump to any cell in one step.

Similarly, all “reasonable” programming languages are equivalent.

The notion of an algorithm is independent of the computation model.

Algorithm

What is an algorithm

Informally

- a recipe
- a procedure
- a computer program

Historically

- algorithms have long history in mathematics
- but not precisely defined until 20th century
- informal notions rarely questioned, but insufficient to show a problem has no algorithm.

Information

Consider the information content of the following two binary sequences.

A = 01010101010101010101010101010101010101

B = 11100101101000111010100001110100110101111

Which contains more information?

Sequence A contains little information because it is merely a repetition of the pattern 01 twenty times. In contrast, sequence B appears to contain more information, because it seems to have no concise description.

Information

We define the quantity of information contained in an object to be the size of that object's smallest representation or description (a precise and unambiguous characterization of the object so that we may recreate it from the description alone.).

Why do we consider only the shortest description when determining an object's quantity of information?

The size of the shortest description determines the amount of information.

Minimal length description

Many types of description language can be used to define information. Selecting which language to use affects the characteristics of the definition.

In this class, our description languages is based on algorithms.

One way to use algorithms to describe strings is to construct a Turing machine that prints out the string when it is started on a blank tape and then represent that Turing machine itself as a string.

Drawback to this approach:
A Turing machine cannot represent a table of information concisely with its transition function. To represent a string of n bits, you might use n states and n rows in the transition function table. That would result in a description that is excessively long for our purpose.

Minimal length description

We describe a binary string x with a Turing machine M and a binary input w to M. The length of the description is the combined length of representing M and w.

- Writing this description with our usual notation for encoding several objects into a single binary string $\langle M, w \rangle$.
- To produce a concise result, we define the string $\langle M, w \rangle$ to be $\langle M \rangle w$

However, we might run into trouble if directly concatenating w onto the end of $\langle M \rangle$.

The point at which $\langle M \rangle$ ends and w begins is not discernible from the description itself.

Minimal length description

How can we solve this problem?

We avoid this problem by ensuring that we can locate the separation between $\langle M \rangle$ and w in $\langle M \rangle w$.

Minimal length description

Definition

Let x be a binary string. The **minimal description** of x, written $d(x)$, is the shortest string $\langle M, w \rangle$ where TM M on input w halts with x on its tape. The **descriptive complexity** of x, written $K(x)$, is

$$K(x) = |d(x)|$$

- In other words, $K(x)$ is the length of the minimal description of x.
- The definition of $K(x)$ is intended to capture our intuition for the amount of information in the string x.

Minimal length description

Theorem

$\exists c \, \forall x \;\; [K(x) \leq |x| + c]$

This theorem says that the descriptive complexity of a string is at most a fixed constant more than its length. The constant is a universal one, not dependent on the string.

$\exists$: existential quantifier, meaning “there exists”

$\forall$: universal quantifier, meaning “for all”

Minimal length description

Theorem

$\exists c\ \forall x\ \ [K(xx) \leq K(x) + c]$

The information contained by the string xx is not significantly more than the information contained by x.

Proof

Proof

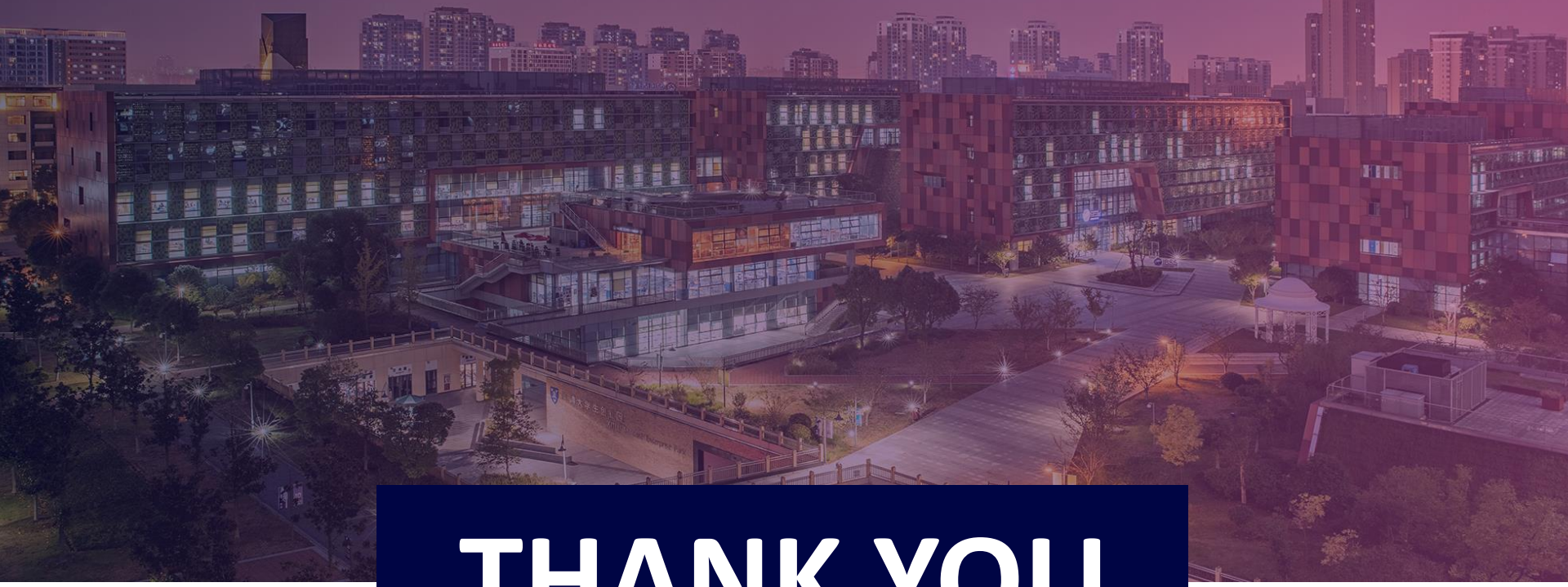
Minimal length description

Theorem

$\exists c\ \forall x, y\ \ [K(xy) \leq 2K(x) + K(y) + c]$

The cost of combining two descriptions leads to a bound that is greater than the sum of the individual complexities.

Proof (tutorial)



THANK YOU

Xi'an Jiaotong-Liverpool University
西交利物浦大学

XJTLU | SCHOOL OF FILM AND TV ARTS