Theory of Computation Notes: Mapping Reductions

Arjun Chandrasekhar

Previously we have seen informal notions of reduciblility. We showed instances in which a language A was 'reducible' to language B: a machine that decided language B could be used to construct a machine that decided language A.

By extension, if a solution to B would yield a solution to A, but we know that A has no solution, we conclude that there cannot be a solution to B.

Here we will formalize that notion, with the concept of mapping reduciblility.

1 Computable Functions

So far in this course we have worked with machines that take an input string w and either accept or reject w. But we can also construct machines such that take an input, and produce an output string. We will make this notion precise here.

Definition 1.1. Let $f : \Sigma^* \to \Sigma^*$ be a function from strings to strings. We say f is a **computable function** if there is a Turing machine M such that for all input strings w, M will always halt, and at the end of the computation M will leave $f(w)$ (and nothing else) on the tape.

Remark. As always, you can think of a computable as a java function or program that takes an input and produces an output. We are simply expanding our definition of computability to include all java functions, rather than just boolean functions.

Example 1.1. The function $f(n) = 2n$ is a computable function. We start with an integer n on the tape. We can program a TM to perform binary multiplication (or multiplication in whatever base we are using). At the end of the computation we leave $2n$ on the tape.

2 Mapping Reducibility

Definition 2.1. Let $A, B \subseteq \Sigma^*$ be formal languages. Suppose there is a function $f : \Sigma^* \to \Sigma^*$ such that $w \in A \Leftrightarrow f(w) \in B$. We say A is **mapping reducible** to B. We denote this as $A \leq_M B$. We call the function f a **reduction** from A to B.

Remark. We sometimes use the following two phrases to characterize a mapping reduction:

1. "YES maps to YES"
2. "NO maps to NO"

Figure 1 illustrates this concept.

Example 2.1. Consider the following two langauges:

$$A = \{n \in \mathbb{N} | n \text{ is even}\}$$
$$B = \{n \in \mathbb{N} | n \text{ is odd}\}$$

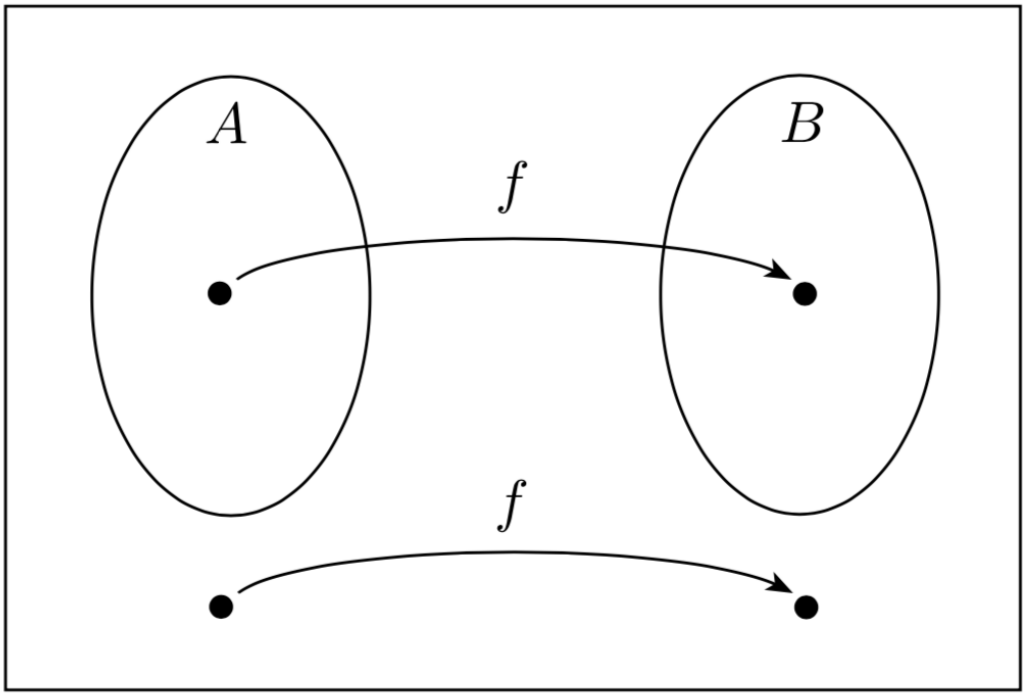

Figure 1: **Illustration of a mapping reduction.** Yes maps to Yes - if we start with an element in A, we end up with an element in B. Similarly, No maps to No.

We will show that $A \leq_M B$. Consider the function $f(n) = n + 1$. Clearly this function is computable. Furthermore, we note that for any integer n, if n is even then $n + 1$ is odd and vice-versa. Thus $n \in A \Leftrightarrow f(w) \in B$. Thus, $A \leq_M B$, and f is a reduction from A to B.

Theorem 2.1. $\mathrm{A_{TM}} \leq_M \mathrm{HALT}$

Proof. Recall that

$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$
$$\mathrm{HALT} = \{\langle M', w'\rangle | M' \text{ halts on } w'\}$$

We need to define a function $f : \langle M, w\rangle \to \langle M', w'\rangle$ such that

$$\langle M, w\rangle \in \mathrm{A_{TM}} \Leftrightarrow f(\langle M, w\rangle) \in \mathrm{HALT}$$

We will define $f : \langle M, w\rangle \to \langle M', w'\rangle$ as follows:

1. M' is a TM defined as follows:
   (a) M' takes input s
   (b) M' simulates M on s
   (c) If M accepts then M' accepts
   If M rejects or loops then M' goes into a loop
2. $w' = w$

Note that M' is the same as M, but instead of rejecting it loops.

1. "YES maps to YES": If M accepts w then so will M'
2. "NO maps to NO"If M does not accept w then M' will loop. If M loops on w then so will M'. If M halts and rejects w then M' will still go into a loop after M rejects

Thus

$$\langle M, w\rangle \in \mathrm{A_{TM}} \Leftrightarrow M \text{ accepts } w \Leftrightarrow M' \text{ halts on } w' \Leftrightarrow \langle M', w'\rangle = f(\langle M, w\rangle) \in \mathrm{HALT}$$

□

Remark. This example illustrates the difference between Turing reductions and mapping reductions. We saw earlier that $\mathrm{A_{TM}} \leq_T \mathrm{HALT}$. In that example, we used a decider M_H for HALT to check if M would loop on w. If so, we automatically rejected $\langle M, w\rangle$; otherwise we ran M on w to see what happened. In that example, we ran M_H on $\langle M, w\rangle$, and then did further processing. If we had wanted to, we could have run M_H as many times as we wanted.

A mapping reduction is more restrictive. To show that $\mathrm{A_{TM}} \leq_M \mathrm{HALT}$, we still get to assume we have a decider M_H for HALT. However, we only get to run M_H once, and it has the be the very last step. Once we run M_H, we have to give whatever output M_H gives.

Theorem 2.2. $\mathrm{A_{TM}} \leq_M \mathrm{EQ_{TM}}$. Recall that

$$\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$$
$$\mathrm{EQ_{TM}} = \{\langle M_1, M_2\rangle | L(M_1) = L(M_2)\}$$

Proof. We need a function $f : \langle M, w\rangle \to \langle M_1, M_2\rangle$ such that

$$\langle M, w\rangle \in \mathrm{A_{TM}} \Leftrightarrow \langle M_1, M_2\rangle \in \mathrm{EQ_{TM}}$$

We define $f : \langle M, w\rangle \to \langle M_1, M_2\rangle$ as follows:

1. M_1 is a machine that accepts everything
2. M_2 is a TM that does the following:
   (a) M_2 takes s as input
   (b) Ignore s and run M on w

1. "YES maps to YES": If M accepts w then both M_1 and M_2 accept everything, so they are equal
2. "NO maps to NO": If M doesn't accept w then M_1 accepts everything but M_2 accepts nothing, so they are not equal

Thus

$$\langle M, w\rangle \in \mathrm{A_{TM}} \Leftrightarrow w \in L(M) \Leftrightarrow L(M_1) = L(M_2) \Leftrightarrow \langle M_1, M_2\rangle = f(\langle M, w\rangle) \in \mathrm{EQ_{TM}}$$

□

Theorem 2.3. $\mathrm{A_{TM}} \leq_M \overline{\mathrm{EQ_{TM}}}$

Proof. We need a function $f : \langle M, w\rangle \to \langle M_1, M_2\rangle$ such that

$$\langle M, w\rangle \in \mathrm{A_{TM}} \Leftrightarrow \langle M_1, M_2\rangle \in \overline{\mathrm{EQ_{TM}}}$$

We define $f : \langle M, w\rangle \to \langle M_1, M_2\rangle$ as follows:

1. M_1 is a machine that rejects everything
2. M_2 is a TM that does the following:
   (a) M_2 takes s as input
   (b) Ignore s and run M on w

1. "YES maps to YES": If M accepts w then M_1 accepts nothing while M_2 accepts everything, so they are unequal
2. "NO maps to NO": If M doesn't accept w then M_1 and M_2 both accept nothing, so they are not unequal

Thus

$$\langle M, w\rangle \in \mathrm{A_{TM}} \Leftrightarrow w \in L(M) \Leftrightarrow L(M_1) \neq L(M_2) \Leftrightarrow \langle M_1, M_2\rangle = f(\langle M, w\rangle) \in \overline{\mathrm{EQ_{TM}}}$$

□

3 Mapping Reductions and Recognizability

We have seen that Turing reductions are a powerful tool to prove that certain languages are undecidable. We will show that mapping reductions can accomplish this too, while also allowing us to show that some problems are not even recognizable.

Theorem 3.1. The following two statements are true:

- If $A \leq_M B$ and B is decidable then A is decidable.
- If $A \leq_M B$ and B is recognizable then A is recognizable.

Proof.

- If B is decidable then some machine M_B decides B. If $A \leq_M B$ then there is a computable function f such that $w \in A \Leftrightarrow f(w) \in B$. We construct a machine M_A that decides A as follows:
  1. M_A takes w as input
  2. Compute $f(w)$
  3. Run M_B on $f(w)$
  4. If M_B accepts $f(w)$, accept
     If M_B rejects $f(w)$, reject

Note that because f is computable and M_B is a decider, M_A is guaranteed to halt. Furthermore, by the definition of mapping reducibility, we have that

$$M_A \text{ accepts } w \Leftrightarrow M_B \text{ accepts} f(w) \Leftrightarrow f(w) \in B \Leftrightarrow w \in A$$

- If B is recognizable then some machine M_B recognizes B. If $A \leq_M B$ then there is a computable function f such that $w \in A \Leftrightarrow f(w) \in B$. We construct a machine M_A that recognizes A as follows:

  1. M_A takes w as input
  2. Compute $f(w)$
  3. Run M_B on $f(w)$
  4. If M_B accepts $f(w)$, accept
     If M_B does not accept $f(w)$, do not accept

  By the definition of mapping reducibility, we have that

  $$M_A \text{ accepts } w \Leftrightarrow M_B \text{ accepts} f(w) \Leftrightarrow f(w) \in B \Leftrightarrow w \in A$$

  Note that if $w \notin A$ then $f(w) \notin B$ so M_A might loop in this scenario when it passes control to M_B, but that's ok.

□

Theorem 3.2. The following two statements are true:

- If $A \leq_M B$ and A is undecidable then B is undecidable
- If $A \leq_M B$ and A is unrecognizble then B is unrecognizable

Proof.

- AFSOC B is decidable. Then by theorem 3.1 A is also decidable. But we know that A is undecidable, so this is a contradiction. We conclude that B must be undecidable.
- AFSOC B is recognizable. Then by theorem 3.1 A is also recognizable. But we know that A is unrecognizable, so this is a contradiction. We conclude that B must be unrecognizable.

□

Theorem 3.3. Suppose $A \leq_M B$. Then $\overline{A} \leq_M \overline{B}$

Proof. If $A \leq_M B$ then there is a computable function f such that

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

This also means that

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

This is just another way of saying that

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

Thus f also is a reduction from $\overline{A}$ to $\overline{B}$. □

Theorem 3.4. $\mathrm{EQ_{TM}}$ is neither recognizable nor co-recognizable.

Proof. To show that $\mathrm{EQ_{TM}}$ is not recognizable, we first note that $\overline{\mathrm{A_{TM}}} \leq_M \mathrm{EQ_{TM}}$ - this is a consequence of combining theorem 2.3 and theorem 3.3. We then note that $\overline{\mathrm{A_{TM}}}$ is not Turing-recognizable. Thus, theorem 3.2 tells us that $\mathrm{EQ_{TM}}$ is not recognizable.

To show that $\mathrm{EQ_{TM}}$ is not co-recognizable, we will show that $\overline{\mathrm{EQ_{TM}}}$ is not recognizable. To do this, we first note that $\overline{\mathrm{A_{TM}}} \leq_M \overline{\mathrm{EQ_{TM}}}$ - this is a consequence of combining theorem 2.2 and theorem 3.3. We then note that $\overline{\mathrm{A_{TM}}}$ is not Turing-recognizable. Thus, theorem 3.2 tells us that $\overline{\mathrm{EQ_{TM}}}$ is not recognizable. By definition, this means $\mathrm{EQ_{TM}}$ is not co-recognizable. □