

# Chapter 6

## Reducibility

We will further look at more unsolvable problems, and particularly, we will introduce the primary technique of showing a problem is not solvable. It is called *reducibility*.

Reducibility always involves two problems, say  $A$  and  $B$ . If  $A$  *reduces* to  $B$ , then a solution for  $B$  can be used to solve  $A$ . For example, to clean up your driveway, you only need to have a shovel. Hence, the problem of cleaning the driveway reduces to the problem of finding a shovel. So, if you can solve the latter problem, i.e., if you have found a shovel, then the original problem, i.e., cleaning the driveway, is also solvable.

The reducibility says nothing about the actual solving of problems  $A$  and  $B$ . It only deals with the solvability of problem  $A$ , given a solution of problem  $B$ .

## Back to the business

A reduction is a way of converting one problem to another so that a solution to the latter can be used to solve the former. The key is that *if we know that the former problem is unsolvable, so must be the latter.*

The unsolvable problem we will use most of the times is  $A_{\text{T.M}}$ , which is the set of  $\langle M, w \rangle$  such that  $M$  accepts  $w$ . We have proved that it is not decidable. Hence, the problem to decide, for any given  $M$  and  $w$ , if  $M$  accepts  $w$  is not solvable.

Let  $HALT_{\text{T.M}}$  be the problem that, for any given  $M$  and  $w$ , if  $M$  halts on  $w$ . We will prove that  $HALT_{\text{T.M}}$  is unsolvable, by reducing  $A_{\text{T.M}}$  to  $HALT_{\text{T.M}}$ .

We will show that, assuming that  $HALT_{TM}$  has a solution, then we can also provide a solution to the  $A_{TM}$ , which is a contradiction, since we know that the latter is unsolvable. The key is to reduce  $A_{TM}$  to  $HALT_{TM}$ .

Assume that  $R$  is the TM that decides  $HALT_{TM}$ , i.e., for any given  $M$  and  $w$ ,  $R$  will tell us if  $M$  halts on  $w$ . We will develop another TM  $S$ , that decides if  $M$  accepts  $w$ : For any given  $M$  and  $w$ , we run  $R$  first, if  $R$  says  $M$  does not halt on  $w$ ,  $S$  rejects  $\langle M, w \rangle$ , since then there is no way  $M$  will accept  $s$ . On the other hand, if  $R$  says that  $M$  does halt on  $w$ , we then simply run  $M$  on  $w$ , and accepts  $\langle M, w \rangle$ , if  $M$  accepts  $w$ ; rejects otherwise.

Therefore, any solution to the  $HALT_{TM}$  problem indeed provides a solution for the  $A_{TM}$ . Hence, the former does reduce to the latter.

**Theorem**  $HALT_{TM}$  is undecidable.

**Proof:** Assume that  $R$  decides the  $HALT_{TM}$ . We construct the following  $S$  that decides  $A_{TM}$ .

$S =$  “On input  $\langle M, w \rangle$ ,

1. Run  $R$  on input  $\langle M, w \rangle$ .
2. If  $R$  rejects, reject.
3. If  $R$  accepts, run  $M$  on  $w$  until it halts.
4. If  $M$  accepts, accept; otherwise, reject.”

But, as  $A_{TM}$  is unsolvable, we reach a contradiction. Therefore,  $HALT_{TM}$  is not solvable.

## Another problem

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

**Theorem**  $E_{\text{TM}}$  is undecidable.

Again, we are trying to reduce  $A_{\text{TM}}$  to  $E_{\text{TM}}$ . Let  $R$  be the TM that decides  $E_{\text{TM}}$ , we are trying to construct another TM,  $S$ , that decides  $A_{\text{TM}}$ , with the help of  $R$ .

One thing is easy, for any given  $M$  and  $w$ , if  $R$  accepts  $M$ , then  $M$  definitely will not accept  $w$ . (?)

But, if  $R$  rejects  $M$ , the only thing we know is that  $L(M)$  is not empty, but does it contain  $w$ ? We have no clue.

What we will do is to construct another TM,  $M_1$ , first such that the only string it *might* accept is  $w$ . Then, run  $R$  on  $M_1$ . If  $R$  accepts  $M_1$ , it definitely says that  $M$  doesn't accept  $w$ . On the other hand, if  $R$  rejects  $M_1$ , then  $M$  accepts something, which must be  $w$ .

**Proof:** Let  $R$  decide  $E_{\text{TM}}$ , and construct  $M_1$  as follows:  $M_1 =$  "On input  $x$

1. If  $x \neq w$ , reject.
2. otherwise, run  $M$  on  $w$  and accept if  $M$  does.

Thus, when  $R$  rejects  $M_1$ ,  $L(M_1) = \{w\}$ , i.e., when  $M$  accepts  $w$ ; and  $R$  accepts  $M_1$  when  $L(M_1) = \emptyset$ , which happens when  $M$  does not accept  $w$ .

Then, we construct  $S$  that accepts  $A_{\text{TM}}$ .

$S =$  "On input  $\langle M, w \rangle$ .

1. Construct  $M_1$ .
2. Run  $R$  on input  $\langle M_1 \rangle$ .
3. If  $R$  accepts, reject; accept otherwise."

## Yet another problem

Let  $Regular_{TM}$  be  $\{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular.}\}$

**Theorem**  $Regular_{TM}$  is undecidable.

Assume that  $R$  decides  $Regular_{TM}$ , we construct an  $S$  that decides  $A_{TM}$ .

The key idea is to construct another  $M_2$  such that, for any given  $M$  and  $w$ ,  $L(M_2)$  is a regular language iff  $M$  accepts  $w$ . More specifically, we design  $M_2$  so that  $L(M_2) = \{0^n 1^n \mid n \geq 0\}$  if  $M$  does not accept  $w$  and  $L(M_2) = \Sigma^*$  if  $M$  accepts  $w$ .

Hence,  $A_{TM}$  reduces to  $Regular_{TM}$ .

**Proof:** Let  $R$  decides  $Regular_{TM}$ , and construct  $S$  that decides  $A_{TM}$ .

$S =$  "On input  $\langle M, w \rangle$  :

1. Construct the following  $M_2$

$M_2 =$  "On input  $x$ :

1.1. If  $x$  has the format  $0^n 1^n$ , accept.

1.2. Otherwise, run  $M$  on  $w$  and accept  $x$  if  $M$  accepts  $w$ .

2. Run  $R$  on input  $\langle M_2 \rangle$ .

3. If  $R$  accepts, accept; otherwise, reject."

**Note:** If  $M$  doesn't accept  $w$ , then the only strings that  $M_2$  will accept are just those in the form of  $0^n 1^n$ ; i.e.,  $L(M_2) = \{0^n 1^n | n \geq 0\}$ ; which is not regular. But, if  $M$  does accept  $w$ ,  $M_2$  accepts everything, i.e.,  $\Sigma^*$ , which is regular.

Therefore,  $S$  accepts  $\langle M, w \rangle$  iff  $R$  accepts  $M_2$  iff  $L(M_2)$  is regular iff  $M$  accepts  $w$ .

## Not always $A_{\text{TM}}$

So far, we always reduce the problem we are interested in to  $A_{\text{TM}}$ , which is certainly not necessary. Let  $EQ_{\text{TM}}$  be  $\{\langle M_1, M_2 \rangle \mid L(M_1) = L(M_2)\}$ , we reduces  $E_{\text{TM}}$  to it.

**Theorem**  $EQ_{\text{TM}}$  is undecidable.

**Proof:** Let  $R$  decides  $EQ_{\text{TM}}$ , we construct the following  $S$  to decide  $E_{\text{TM}}$ .

$S =$  “ On input  $\langle M \rangle$  :

1. Run  $R$  on inputs  $\langle M, M_1 \rangle$ , where  $M_1$  rejects everything, i.e.,  $L(M_1) = \emptyset$ .

2. If  $R$  accepts, accepts; otherwise rejects. ”

Hence  $S$  accepts  $M$  iff  $R$  accepts  $\langle M, M_1 \rangle$  iff  $L(M) = L(M_1) = \emptyset$ , i.e.,  $S$  decides  $E_{\text{TM}}$ . But, as we showed before, the latter is unsolvable, so must be the former.

# Computation history

The computation history method is an important technique for proving that  $A_{\text{TM}}$  reduces to other languages. It is often used to test if something exists, when showing a problem is unsolvable.

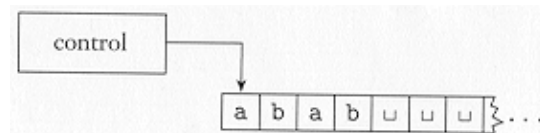
**Definition:** Let  $M$  be a Turing machine and  $w$  an input string. An *accepting computation history* for  $M$  on  $w$  is a sequence of configurations,  $C_1, \dots, C_l$ , where  $C_1$  is the start configuration of  $M$  on  $w$ ,  $C_l$  is an accepting configuration of  $M$ , and each  $C_i$  legally follows from  $C_{i-1}$  according to the rules of  $M$ .

Similarly, we can define *rejecting computation history*.

# Linear bounded automata

A *linear bounded automaton* is a restricted type of Turing machine, in which the tape head is not allowed to move off the portion of the tape containing the input. If it tries to, it will stay where it is.

A *LBA* is a restricted TM with a limited, but unrestricted, memory. Using a tape alphabet larger than the input alphabet allows the available space increased up to a constant factor. Hence, for a given input of length  $n$ , the amount of memory available is linear in  $n$ .



We once discussed context sensitive language, which corresponds to LBA.

## *LBA* is powerful

Despite its constrained memory, *LBA* is quite powerful. For example, the deciders for  $A_{\text{DFA}}$ ,  $A_{\text{CFG}}$ ,  $E_{\text{DFA}}$ , and  $E_{\text{CFG}}$  are all *LBA*s. Every *CFL* can also be decided by an *LBA*: for a word  $w$ , it only checks all the derivations with their length being exactly  $2|w| - 1$ .

We will now check the decidability of several *LBA* related problems.

We have the following simple result.

**Lemma:** Let  $M$  be an *LBA* with  $q$  states, and  $g$  symbols in its tape alphabet. There are exactly  $qng^n$  configurations of  $M$  for a tape of length  $n$ .

**Proof:** Consider a configuration  $up_i v$ . The state can be any of  $q$  states, the R/W head points to any of the  $n$  symbol, each of which can be any of the  $g$  symbols.

## $A_{\text{LBA}}$ is decidable

Let  $A_{\text{LBA}}$  be  $\{\langle M, w \rangle \mid M \text{ is an LBA that accepts } w.\}$  We have the following

**Theorem:**  $A_{\text{LBA}}$  is decidable.

If  $M$  goes into an infinite loop, it will get locked up in that loop. On the other hand, the maximum length of an accepting computation history for  $M$  on  $w$ , by the lemma, is  $q|w|g^{|w|}$ . Hence, the following decider  $L$ .

$L =$  “On input  $\langle M, w \rangle$ ,

1. Simulate  $M$  on  $w$  for  $q|w|g^{|w|}$  many steps or until it halts.
2. If  $M$  has halted, accept if  $M$  accepts, and reject otherwise. If  $M$  has not halted, reject.

The key here is the finiteness of the configuration, which does not hold for a general TM.

# $E_{LBA}$ is undecidable

Let  $E_{LBA}$  be  $\{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset.\}$  We have the following

**Theorem:**  $E_{LBA}$  is not decidable.

The proof is by reduction from  $A_{TM}$ . For a Turing machine  $M$  and  $w$ , we will construct an LBA  $B$  that accepts all the accepting computation history of  $M$  on  $w$ , then testing if  $L(B)$  is empty, using the assumed decider for  $E_{LBA}$ . If  $L(B)$  is empty,  $M$  does not accept  $w$ , otherwise, it does. This leads to the contradiction we need.

Below shows a possible input to  $B$ .

The diagram shows a sequence of configurations  $C_1, C_2, C_3, \dots, C_l$  separated by markers  $\#$ . Each configuration  $C_i$  is enclosed in a bracket underneath. The sequence is:  $\#$   $\underbrace{\hspace{1cm}}$   $\#$   $\underbrace{\hspace{1cm}}$   $\#$   $\underbrace{\hspace{1cm}}$   $\#$   $\dots$   $\#$   $\underbrace{\hspace{1cm}}$   $\#$ . The brackets are labeled  $C_1, C_2, C_3, \dots, C_l$  respectively.

## How does $B$ work?

When it receives an input  $x$ ,  $B$  is supposed to accept  $x$  if  $x$  is an accepting computation for  $M$  on  $w$ .

For a given  $\langle M \rangle$  and  $w$ ,  $B$  will designate  $q_0w$  as the initial configuration, where  $q_0$  is the starting state of  $M$ , and it will accept a configuration  $uqv$  if  $q$  is the accepting state of  $M$ . For any given configuration  $C_i$ , it will generate a next configuration  $C_{i+1}$  with the rules as given in  $M$ . For more construction and operation details, check out the book.

The language of  $B$  is the collection of all the accepting computation of  $M$  on  $w$ .

## Now what?

Clearly,  $L(B) = \emptyset$  iff  $M$  accepts  $w$ .

Now let  $R$  decide  $E_{LBA}$ . Then the following  $S$  decides  $A_{TM}$ .

$S =$  "On input  $\langle M, w \rangle$ ,

1. Construct the  $LBA$   $B$ , as described
2. Run  $R$  on  $\langle B \rangle$ .
3. If  $R$  reject, accept; reject otherwise.

## A simple problem?

We will show that some rather natural problem is also undecidable. Particularly, we will show that the so-called *Post Correspondence Problem (PCP)* is not decidable.

Given a collection of dominos, each of which contains two strings, one on each side, an instance of the *PCP* problem is a collection,  $P$ , of dominos:

$$\left\{ \left[ \begin{array}{c} t_1 \\ b_1 \end{array} \right], \left[ \begin{array}{c} t_2 \\ b_2 \end{array} \right], \dots, \left[ \begin{array}{c} t_k \\ b_k \end{array} \right] \right\},$$

and a *match* is a sequence,  $i_1, i_2, \dots, i_l$ , of such dominos in  $P$ , such that

$$t_{i_1} t_{i_2} \cdots t_{i_l} = b_{i_1} b_{i_2} \cdots b_{i_l}.$$

## An example

Let our collection look like the following:

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

Below shows a match:

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

## Guess what?

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

**Theorem:**  $PCP$  is undecidable.

The proof is actually quite simple, which basically shows that for any given Turing machine,  $M$ , and string,  $w$ , we can construct an instance  $P$  of  $PCP$  such that  $P$  has a match iff  $M$  accepts  $w$ .

That is to say, we simply reduce  $A_{TM}$  to  $PCP$ .

For the details of this proof, check out the book.

## What is really going on?

We have been showing how to apply the reducibility technique to prove the undecidability of several problems.

Now, we are trying to formalize this handy notion of reduction so that we can understand it better.

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

## An example

If we represent any number  $n$  with  $n + 1$  '1's, i.e., 0 is '1', 1 is '11', etc. Then, we have the following turing machine:

On input  $\overbrace{1 \cdots 1}^{m+1} b \overbrace{1 \cdots 1}^{n+1}$ ,

1. Scan all the symbols. If the second segment contains only one '1', change it to 'b'
2. Otherwise, go over the first segment of '1's.
3. Fill the blank 'b' with 1.
4. Go over to the end of the second segment of '1's, and change the last two '1's with 'b'.

Thus, addition is computable.

## Formalize reduction

**Definition:** Let  $A$  and  $B$  be languages. By saying that  $A$  is *mapping reducible* to  $B$ , ( $A \leq_m B$ ), we mean there is a computable function,  $f : \Sigma^* \mapsto \Sigma^*$ , such that for every  $w \in \Sigma^*$ ,

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

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

## What does it mean?

A mapping reduction of  $A$  to  $B$  provides a method to convert questions about membership testing in  $A$  to membership testing in  $B$ . Hence, to test  $w \in A$ , we can use the reduction to compute  $f(w)$  and test if  $f(w) \in B$ . By definition, we can provide a definitive answer for the former question based on the result obtained from the latter question.

More particularly, if one problem is mapping reducible to a second, previously solved problem, we can obtain a solution to the original one.

On the other hand, if we can map reducible a known unsolvable problem to a problem, then, the latter problem must be unsolvable, as well.

**Theorem:** If  $A \leq_m B$  and  $B$  is decidable, so is  $A$ .

**Proof:** Let  $M$  decide  $B$ , and let  $f$  be the reduction of  $A$  to  $B$ . We construct the following decider,  $N$ , for  $A$ .

$N =$  “On input  $w$

1. Compute  $f(w)$ .
2. Run  $M$  on  $f(w)$ .
3. Accept, if  $M$  accepts; reject otherwise.”

**Corollary:** If  $A \leq_m B$  and  $A$  is undecidable, so is  $B$ .

## An example

We once informally showed that the  $HALT_{TM}$  is undecidable by reducing  $A_{TM}$  to it. Now, we look into it from this new perspective of mapping reduction.

Particularly, we have to present a computable function,  $f$ , which, for any input  $\langle M, w \rangle$ , will provide an output  $\langle M', w \rangle$  such that

$$\langle M, w \rangle \in A_{TM} \text{ iff } \langle M', w \rangle \in HALT_{TM}.$$

Below shows  $F$ , a Turing machine that computes  $f$  :

$F =$  " On input  $\langle M, w \rangle$

1. Construct the following machine  $M'$

$M' =$  "On input  $x$

1.1. Run  $M$  on  $x$ .

1.2. If  $M$  accepts, accept.

1.3. If  $M$  rejects, enter a loop."

2. Output  $\langle M', w \rangle$ ."

## So what?

Now let's show that with this function,  $A_{\text{TM}}$  mapping reduces to  $HALT_{\text{TM}}$ , i.e.,

$$\langle M, w \rangle \in A_{\text{TM}} \text{ iff } \langle M', w' \rangle \in HALT_{\text{TM}}.$$

Assume  $\langle M, w \rangle \in A_{\text{TM}}$ , i.e.,  $M$  accepts  $w$ , then by the construction,  $M'$  also accepts  $w$ , thus halts with  $w$ , i.e.,  $\langle M', w \rangle \in HALT_{\text{TM}}$ .

On the other hand, if  $\langle M', w \rangle \in HALT_{\text{TM}}$ , then  $M$  has to accept  $w$ ; otherwise, by the construction of  $M'$ , either  $M$  rejects  $w$  or get into an infinite loop,  $M'$  would get into an infinite loop in both cases, thus,  $\langle M', w \rangle \notin HALT_{\text{TM}}$ .

This shows that  $A_{\text{TM}} \leq_m HALT_{\text{TM}}$ .

Finally, by the corollary,  $HALT_{\text{TM}}$  is not solvable.