Chapter 6
Reducibility

As the first undecidability result, in last chapter, we went through all the way to show that *Halting* problem is unsolvable.

In other words, there is no way for us to construct a TM, H, that will be able to tell, for any TM, M, and a string, w, M will accept w, or not.

Technically, let $A_{\text{TM}} = \{\langle M, w\rangle : M$ is a TM, and w is a string over $\Sigma(M)\}$, A_{TM} is not decidable. On the other hand, we proved that both A_{FA} and A_{NPDA} are decidable.

Recall that LBA is a special TM in the sense that the amount of space that we can use there is a linear function of $|w|$, the length of w.

Question: How about A_{LBA}?

Answer: It is decidable (Cf. Page 27)

We have also discussed several other problems. For example, we have seen that we can construct a TM which will be able to tell, given any FA, or NPDA ($\equiv$CFG), whether it accepts any string w.

In other words, both E_{FA} and E_{NPDA} are decidable, where, e.g., $E_{\mathsf{FA}} = \{\langle M\rangle : M$ is an FA, $L(M) = \emptyset\}$.

But, E_{LBA} is not decidable (Cf. Page 28)

We also have seen that we can construct a TM which will be able to tell, given any pair of FAs, M_1, M_2, whether they accept the same language, i.e., $L(M_1) = L(M_2)$.

In other words, EQ_{FA} is decidable. But EQ_{NPDA} is not.

Hence, EQ_{TM} is certainly not, either. Otherwise, EQ_{NPDA} would be.

Pain reliever

The question is how to prove the unsolvability of E_{LBA} and EQ_{TM}, and other problems?

It would be a pain to repeat the lengthy process that we went through with A_{TM}.

Fortunately, we have a much more efficient technique, based on logic reasoning, called *reducibility*, to show the undecidability of these, and other, problems.

Reducibility always involves two problems, say A and B. *If A reduces to $B, A \leq_m B$, then a solution for B can be used to solve A.* ☺

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

On the positive side...

If a problem A can be reduced to a problem B, and B is solvable, its solution does lead to a solution to A.

For example, given the statement “If I have five bucks, I will buy you lunch.”, the problem of “getting your lunch” (A) is reduced to “my having five bucks” (B). Thus, if I could somehow find five bucks, then your lunch is on me. ☺

As another example, by the grading criteria of this course, if you get at least 92.5, you will get ‘A’. Thus, the problem of getting ‘A’ is reduced to that of getting 92.5.

We went through lots of applications of *this positive direction* in our business in the *Decidability* chapter, from Page 3 to 12, to find quite a few problems solvable.

An example

On Page 3 of the *Decidability* chapter, we showed that A_{DFA} is decidable, where

$A_{\text{DFA}} = \{\langle B, w\rangle : B$ is a DFA that accepts $w.\}$

This is to say that, there exists a TM, M, for any DFA, B, and a string w, M decides if $w \in L(B)$.

Using this result, we can show that A_{NFA} is decidable:

It is shown, on Page 63 of the *FA* notes, we can mechanically convert any NFA to a DFA, thus, given an NFA, N, and a string w, to decide whether $w \in L(N)$, we could build up the following TM, by the Church-Turing Thesis,

1. Convert N to a DFA D, $L(N) = L(D)$.

2. If D accepts $\langle B, w\rangle$, accept $\langle N, w\rangle$; otherwise, reject $\langle N, w\rangle$.

Thus, A_{NFA} is reduced to A_{DFA}.

On the other side...

Given “If I have five bucks, I will buy you lunch”, what would you conclude if you know for sure that I won’t buy you lunch? ☺

Leaves are on your yard.... To clean it up, you only need to have a rake or a blower. Hence, the problem of cleaning the yard reduces to the problem of finding a tool.

If you can solve the latter problem, i.e., if you have found a rake or a blower, then the original problem, i.e., cleaning the yard can also be solved.

What if you get a message, from God, that you won’t be able to clean your yard. Can you find a rake or a blower? ☺

When a problem A is known to be unsolvable, and it is reduced to $B, A \leq_m B$, then, B cannot solved, either.

Question: I feel dizzy, why is *this* the case?

Back to the business

If B can be solved, by definition, its solution can be used to find a solution to solve this problem A, which is known to be *unsolvable.* A contradiction.

The insight is that *if we know that the former problem, A, is* unsolvable, *so must be the latter problem B.*

The unsolvable problem we will use most of the times is A_{TM}, which is the collection of $\langle M, w\rangle$ such that M accepts w. (Cf. Page 13, *Decidability* Chapter notes) This problem is shown to be undecidable, or unsolvable.(Cf. Page 29, *Decidability* Chapter notes)

We will now show several problems, P, unsolvable, by reducing A_{TM} to P.

The logic is simply this: if P is solvable, so is A_{TM}. Since the latter is not solvable, P cannot solvable, either.

From “$\rightarrow$” to “$\leq_m$”

We can simplify “If I have five bucks, I will buy you lunch.” to “Five bucks $\rightarrow$ Lunch”. Then, the problem of “I will buy you lunch” is reduced to that of “I have five bucks”.

- “Five bucks $\rightarrow$ Lunch’ $\equiv$ “Lunch $\leq_m$ Five bucks”
- “92.5 $\rightarrow$ A” $\equiv$ “A $\leq_m$ 92.5”
- “A_{DFA} is decidable $\rightarrow$ A_{NFA} is decidable $\equiv$ “Decidability of A_{NFA} $\leq_m$ Decidability of A_{DFA}”

Let $HALT_{\text{TM}}$ be the problem that, for any given M and w, if M *halts* on w.

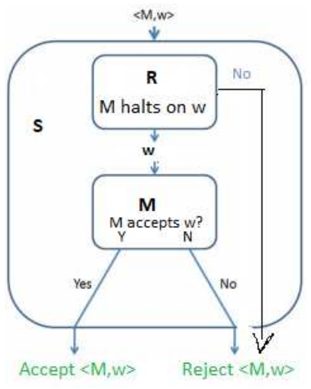
We will prove that $HALT_{\text{TM}}$ is unsolvable, by reducing A_{TM} to $HALT_{\text{TM}}$, i.e., if $HALT_{\text{TM}}$ is solvable, so is A_{TM}.

I want to see...

By the reduction idea, if $HALT_{\text{TM}}$ has a solution, then we can also solve the A_{TM} problem, which is a contradiction, since we know that A_{TM} is unsolvable.

The key is thus how to reduce A_{TM} to $HALT_{\text{TM}}$.

If we have a machine R that decides $HALT_{\text{TM}}$, we can use it to build up another one, S, that will decide the A_{TM} problem as follows:



What is going on?

Let R be the TM that decides $HALT_{\mathsf{TM}}$, i.e., for any given M and w, R will tell us if M *halts* on w. We use it to construct a TM, S, that decides the A_{TM} language, i.e., for any TM M and a string w, S would decide 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 $HALT_{\mathsf{TM}}$ problem indeed provides a solution for the A_{TM}.

Hence, A_{TM} reduces to $HALT_{\mathsf{TM}}$.

Question: Ready to turn *it* to a TM?

$HALT_{\mathsf{TM}}$ is undecidable

Assume that R decides $HALT_{\mathsf{TM}}$. 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 $\langle M, w\rangle$, reject $\langle M, w\rangle$.
3. If R accepts $\langle M, w\rangle$, then we know that M does halt on w. We thus run M on w until it halts.
4. If M accepts w, accept $\langle M, w\rangle$; otherwise, reject."

But, as A_{TM} is unsolvable, we reach a contradiction. Therefore, there could not exist such a TM R, and $HALT_{\mathsf{TM}}$ is not solvable. ☹

Church-Turing Thesis might be correct... ☺

Another problem

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

We saw earlier that both $E_{\textsf{FA}}$ and $E_{\textsf{CFL}}$ are decidable (Cf. Pages 4 and 9 in the *Decidability* notes).

Theorem $E_{\textsf{TM}}$ is undecidable.

Again, we are trying to reduce $A_{\textsf{TM}}$ to $E_{\textsf{TM}}$.

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

Given a TM M, R either accepts M, when $L(M)=\emptyset$; or rejects M, when $L(M)\neq\emptyset$.

One thing is easy, for any given M and w, if R accepts M, then $L(M)=\emptyset$, namely, M accepts nothing. In particular, M definitely will not accept w. ☹

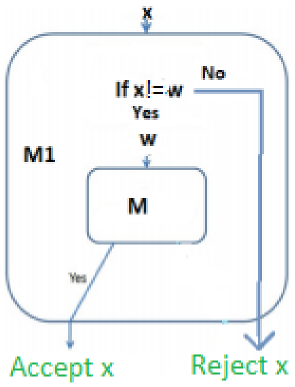
The other side...

If R rejects M, the only thing we know is that $L(M)$ is not empty, so, it accepts *something.*

Question: Does it accept w?

We have no clue. ☹

We construct yet another TM, M_1, with both M and w as its parameters, such that the only string it *might* accept is w,, i.e., $L(M_1) \subseteq \{w\}$, depending on whether M accepts w.



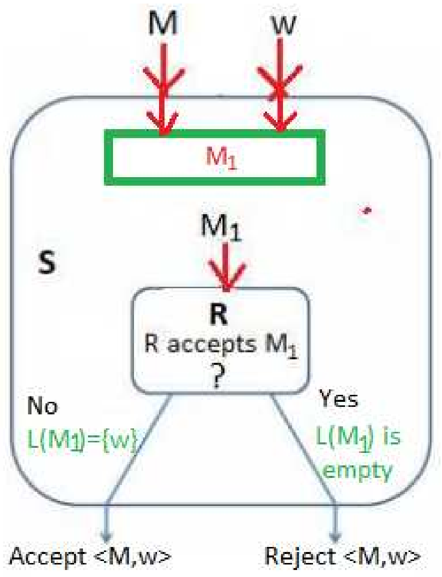
If $x! = w$, reject x; otherwise, M_1 accepts x ($= w$) if M accepts w. As a result, $L(M_1) = \{w\}$ if M accepts w; otherwise, $L(M_1) = \emptyset$.

I want to see...

Technically, we construct M_1 as follows:

$M_1 =$ "On input x
1. If $x! = w$, reject;
2. otherwise, run M on $x\,(= w)$ and accept w if M does.

We then run R on M_1, instead.



With M, w, if $L(M_1)! = \emptyset$,, i.e., $L(M_1) = \{w\}$, then M accepts w; otherwise, $L(M_1) = \emptyset$, when M rejects w. Thus, S decides $\langle M, w\rangle$. ☹

What would you write?...

Let R be a decider of E_{TM}.

$S =$ "On input $\langle M, w\rangle$.
1. Construct M_1, with M and w as parameters
2. Run R on input $\langle M_1\rangle$.
3. If R accepts $\langle M_1\rangle$, i.e., $L(M_1) = \emptyset$, implying $w \notin L(M)$; reject $\langle M, w\rangle$;
4. otherwise, $L(M_1) = \{w\}$, i.e., M accepts w; thus accept $\langle M, w\rangle$."

If R accepts M_1, $L(M_1) = \emptyset$, thus, M doesn't accept w. On the other hand, if R rejects M_1, then M accepts something, which must be w, since this is the only string that could be accepted. ☺

We can thus solve A_{TM}, which is a contradiction. As a result, E_{TM} cannot be solved, either.

Yet another problem

Let $Regular_{\text{TM}}$ be $\{\langle M \rangle : M$ is a TM and $L(M)$ is regular$\}$, where M is an FA, i.e., all the TMs that accept a regular language.

Theorem $Regular_{\text{TM}}$ is undecidable.

Although an FA is a TM, there is no way we can tell if a given TM is a FA or not. ☹

Question: How could we prove it?

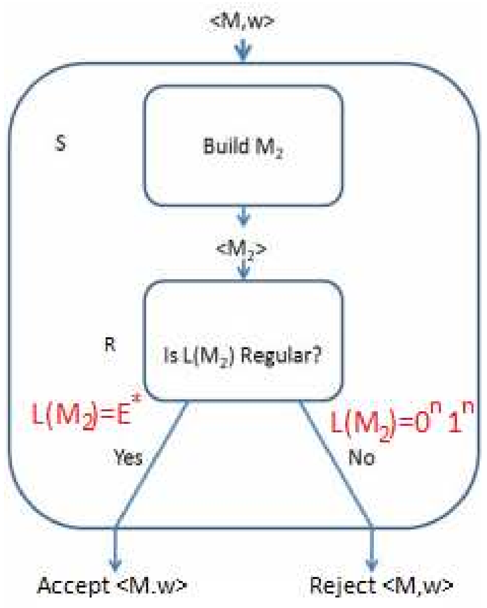
Answer: We reduce A_{TM} to $Regular_{\text{TM}}$: Assume that R decides $Regular_{\text{TM}}$, we construct a TM S that decides A_{TM}, making use of R. ☺

The trick is to design an M_2 so that $L(M_2) = \{0^n1^n | n \geq 0\}$ if M does not accept w and $L(M_2) = \Sigma^*$ if M does.

Notice that $\{0^n1^n | n \geq 0\}$ is not regular, while Σ^* is. Then, *for any given M and w, $L(M_2)$ is a regular language if and only if M accepts w.*

I Want to see...

Let M_2 be defined as discussed, the following chart shows the construction of the machine S that decides A_{TM}.



Again, since A_{TM} is not decidable, there is no way to construct S, nor is R.

Hence, $Regular_{\mathsf{TM}}$ is not decidable, either.

Question: What does M_2 look like?

What would you write?...

Let R decide $Regular_{\text{TM}}$, construct S that decides A_{TM} as follows:

```
S = "On input ⟨M, w⟩ :
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 ⟨M_2⟩.
3. If R accepts ⟨M_2⟩, accept ⟨M, w⟩; otherwise,
    reject ⟨M, w⟩."
```

If M rejects w, then M_2 will accept only those strings in the form of 0^n1^n; i.e., $L(M_2) = \{0^n1^n | n \geq 0\}$; which is not regular (Cf. Page 115 of the *FA* notes). But, if M does accept w, M_2 accepts everything, i.e., Σ^*, which is regular (Cf. Pages 85 and 86 of the *FA* notes).

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

Not always A_{TM}

So far, we always reduce A_{TM} to the problem that we are interested in. This is certainly not necessary. ☺

Question: Can we tell if two animals, whoops, machines, speak the same language?

We saw that EQ_{DFA} is decidable (Cf. Page 6 of the decidability chapter), and it is stated on Page 10 of the decidability chapter that EQ_{CFL} is not decidable.

For a detailed discussion of the undecidability of the latter problem of EQ_{CFL}, check out the piece on the course page.

We saw, on Page 10, E_{TM} is not decidable.

Let EQ_{TM} be $\{\langle M_1, M_2\rangle : L(M_1) = L(M_2)\}$, we now reduce E_{TM} to EQ_{TM} to show that EQ_{TM} is not decidable, either. ☹

The reduction process

Assume R decide EQ_{TM}, we construct the following S to decide E_{TM}.

Let M_1 be a TM such that no instruction goes to the accept state, thus nothing could be accepted by M_1. Clearly $L(M_1) = \emptyset$.

$S =$ " On input $\langle M \rangle$:
1. Run R on inputs $\langle M, M_1 \rangle$, where M_1 does not accept anything, i.e., $L(M_1) = \emptyset$.

2. If R accepts $\langle M, M_1 \rangle$, accepts $\langle M \rangle$; otherwise rejects $\langle M \rangle$. "

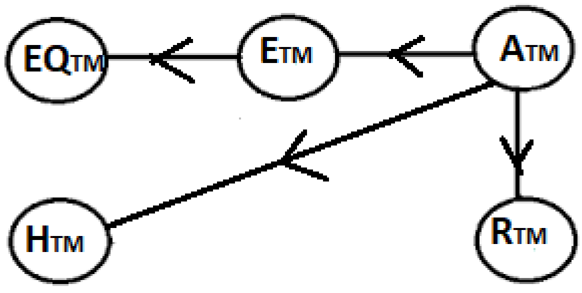
Thus, S decides E_{TM}. ☺

So what?

Hence S accepts M if and only if R accepts $\langle M, M_1\rangle$ if and only if $L(M) = L(M_1) = \emptyset$, i.e., S decides E_{TM}, which contradicts the undecidability of the emptiness problem of a Turing machine. (Cf. Page 10)

Theorem EQ_{TM} is undecidable.

So, the whole structure looks like a tree, rooted at the A_{TM} problem, where $Q \to P$ means P reduces to Q.



This structure certainly can grow to include more problems that are not decidable, starting with the very beginning of A_{TM}, the mother of all the undecidable problems, *MoUD (?)*.

Computation history

The computation history method is another important technique of showing that A_{TM} reduces to other languages.

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, $C_1, \ldots, C_l$, where C_1 is the start configuration of M on w, C_l is an accepting configuration of M, and each C_i is derived from C_{i-1} according to the rules of M.

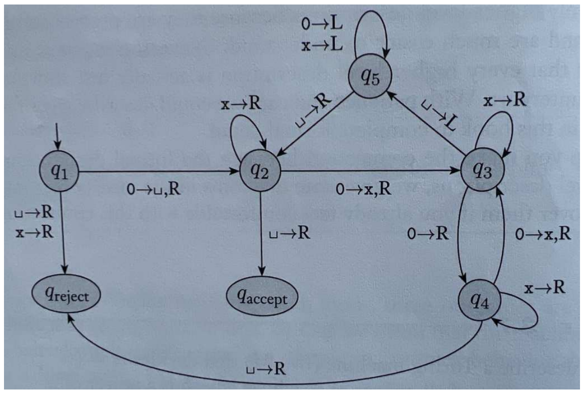
Similarly, we can define *rejecting computation history,* when C_l is a rejecting configuration.

Question: Did we talk about this stuff before?

Answer: Yes, we did on Page 14 of the *Computability* notes, when addressing the *accepting* configuration. ☺

An example

Given the following TM, as shown on Page 172 of the textbook, $L(M_2) = \{0^{2^n} : n \geq 0\}$.



Below is an accepting configuration,

$q_1$00	Bq_5x	Bxq_2
B$q_2$0	q_5Bx	accept
Bxq_3	Bq_2x	

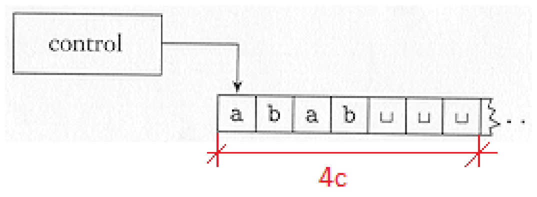
and a reject configuration.

$q_1$000	Bx0q_4
B$q_2$00	reject
Bx$q_3$0	

Linearly bounded automata

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

A *LBA* comes with a quantitatively limited, but qualitatively unrestricted, memory. Using a tape alphabet larger than the input alphabet allows the available space increased up to a *constant factor*. In other words, for a given input of length n, the amount of memory available is *linear in* n, i.e., cn, where c is a constant.



As we discussed on Pages 86, through 88, of the *NPDA* notes, *CSL*, the collection of context sensitive languages, corresponds to LBA.

LBA is powerful

Despite its constrained memory, *LBA* is quite powerful. For example, the deciders for A_{DFA}, A_{CFG}, E_{DFA}, and E_{CFG}, discussed in the *Decidability* chapter, are all *LBA*s.

Theorem: Every *CFL* can be decided by an LBA.

Given an CFG, Γ, we convert it to a Chomsky normal form grammar, Γ', first, as we discussed on Page 18 of the CFL notes.

Then, for a word w, it only needs to check all the derivations in Γ', with their length being exactly $2|w| - 1$, linear in $|w|$, setting a limit as how many cells we need to check.

We will accept w iff at least one of such derivations accepts w. Here the constant $c = 2$.

This again is the basis of a compiler of any programming language (Cf. Pages 11 and 12 of the *Decidability* notes).

Let's work on *LBA* a little

We have the following simple result.

Lemma: Let M $(= (Q, \Sigma, \Gamma, \delta, q_{\text{accept}}, q_{\text{reject}}))$ be an *LBA* with q states, i.e., $|Q| = q$, and g symbols in its tape alphabet, i.e., $|\Gamma| = g$. There are exactly qng^n configurations of M *for a tape of length* n, which exists because it is a LBA, which is in proportional to the input size.

Proof: Consider a configuration up_iv. The state p_i 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.

Each such combination can be a configuration, thus, by the multiplication rule, there are a total of qng^n configurations.

We have looked at several *DFA, NPDA*, and *TM* related problems, and will now check the decidability of several *LBA* related problems.

A_{LBA} is decidable

Let A_{LBA} be $\{\langle M, w\rangle | M$ is an *LBA* that accepts $w.\}$, we have the following result.

Theorem: A_{LBA} is decidable.

Let M be an *LBA,* the maximum length of an accepting computation history for M on w, by the lemma, is $cq|w|g^{|cw|}$. Since, if M goes into an infinite loop, it will get locked up in that loop. Hence, the following TM is able to decide $\langle M, w\rangle$.

$L =$ "On input $\langle M, w\rangle$,
1. Simulate M on w for $cq|w|g^{|cw|}$ many steps or until it halts, where c is a constant.
2. If M has halted, accept if M accepts, and reject otherwise. If M has not halted, reject, as it has got into a loop, because of Pigeonhole principle.

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 $\{\langle M\rangle : M$ is an *LBA* where $L(M) = \emptyset.\}$, we have the following result.

Theorem: E_{LBA} is not decidable.

We reduce A_{TM} to E_{LBA}.

For a TM 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, x, a possible tape content of B, the length of which is proportional to $|Q(M)|$, number of instructions in M,, and $|w|$, input size, thus it is an LBA.

$$\#\underbrace{\qquad\quad}_{C_1}\#\underbrace{\qquad\quad}_{C_2}\#\underbrace{\qquad\quad}_{C_3}\#\ \cdots\ \#\underbrace{\qquad\quad}_{C_l}\#$$

What does B look like?

Knowing $\langle M, w\rangle$, when it receives an input x, B is supposed to accept x if x is an accepting computation for M on w.

Let q_0 be the starting state of M, B works as follows:

For any input x,

```
1. if $x! = \#C_1\#C_2\# \cdots \#C_l\#$, reject $x$;
2. else if $C_1! = q_0w$, reject $x$;
3. else for $(i = 1; i \leq l - 1; i++)$
    3.1. if $C_i$ does not yield to $C_{i+1}$, reject $x$;
    3.2. if $C_l$ is an accepting configuration,
           accept $x$
    3.3. else reject $x$.
```

By the *Church-Turing thesis,* the above algorithm can be converted to a TM B, and the language of $B, L(B)$, is the collection of all the accepting computations of M on w.

Now what?

Clearly, $L(B) \neq \emptyset$ iff there is at least one accepting computation of M on w iff M accepts w.

Now let R decide E_{LBA}, the following TM S would decide A_{TM}.

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

1. Construct the *LBA* B, as described earlier.

2. Run R on $\langle B\rangle$.

3. If R rejects $\langle B\rangle$, accept (There exists at least one accepting computation, thus M accepts w.); reject otherwise (There does not exist any accepting computation, thus M does not accept w). ☹

Hence, E_{LBA} is not decidable. ☺

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:

$$\left\{\left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right]\right\},$$

and a *match* is a sequence, $i_1, i_2, \ldots, i_l$, of such dominos in P, such that $t_{i_1}t_{i_2}\cdots t_{i_l} = b_{i_1}b_{i_2}\cdots b_{i_l}$.

For example, given the following collection:

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

we can find a match as follows:

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

Guess what?

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

Theorem: *PCP* is undecidable.

The proof is actually quite conceptually simple but detailed with six parts. ☹

It 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, again, $A_{\mathsf{TM}} \leq_m$ *PCP*.

For the details of this proof, check out the proof of Theorem 5.15 in the book, as well as the piece in the resource section on the course page.

What is really going on?

To solve a problem A, we reduce it to an already solved problem $B, A \leq_m B$. We do this all the time, e.g., in software reuse.

On the other hand, to prove that a problem B is unsolvable, we reduce an already proved unsolvable problem A to B.

Throughout this chapter, we have been showing how to apply the reducibility technique to prove the undecidability of several problems, by reducing, e.g., A_{TM}, to such a problem.

Now, we formalize this handy notion of reduction so that we can understand it better (or worse ☹).

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.

How about an example?

We always use the decimal system, until we bump into a computer, which only knows two bits. So, we have to switch to the binary system, using just two digits, 0 and 1.

Question: Can we use just one digit?

We can 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 for $m+n$, $m \geq 1$:

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

1. Scan the first segment. If it contains at least two 1's, wipe out the first two.

2. Continue the scan of the first segment, until we see the blank symbol, 'B'. Fill it with 1.

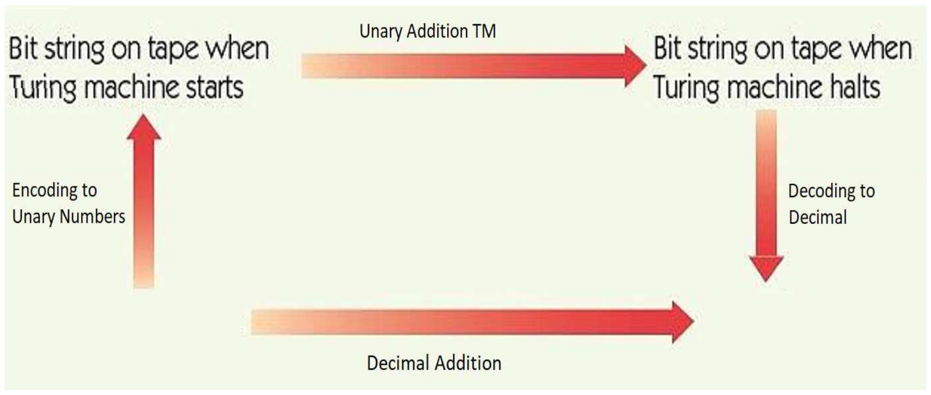
We now end up with $m+n+2-2+1$ $(= m+n+1)$ 1's, representing $m+n$.

How does a TM do it?

1) use a certain form to *encode* the decimal system to a unary one;

2) write a *Turing machine program* to add two numbers in unary system.

3). Once the computation is completed, the output will then be *decode* back to the decimal system.

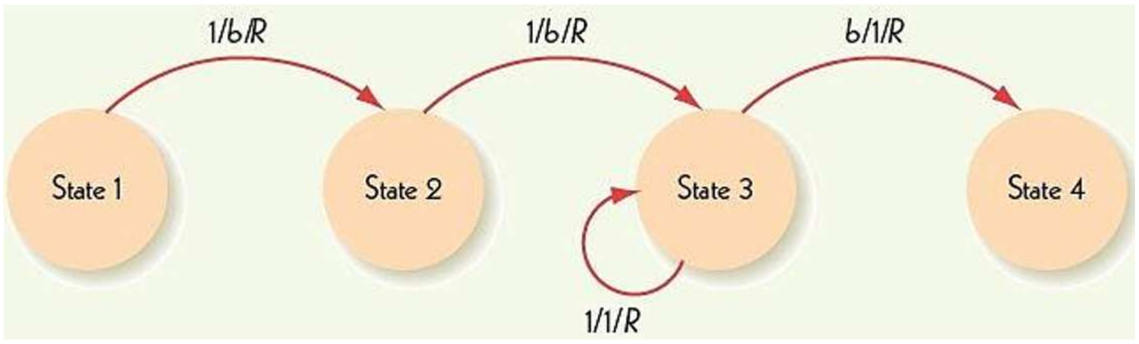

Bit string on tape when Turing machine starts
Unary Addition TM
Bit string on tape when Turing machine halts
Encoding to Unary Numbers
Decoding to Decimal
Decimal Addition

Thus, *technically,* the addition problem is computable. ☺

Question: What does this TM look like?

The unary addition TM

This unary addition TM looks like the following:



Technically, it consists of four instructions:

```
(1, 1, b, 2, R)
(2, 1, b, 3, R)
(3, 1, 1, 3, R)
(3, b, 1, 4, R)
```

Assignment: Play it out with 2+3, i.e.,

·	·	·	1	1	1	b	1	1	1	1	·	·	·

What is going on?

We wipe out the first two ‘1’ in m, then move over to flip ‘b’ back to 1. This would give $(m-2)+n+1=m+n-1$ ‘1’s on the tape.

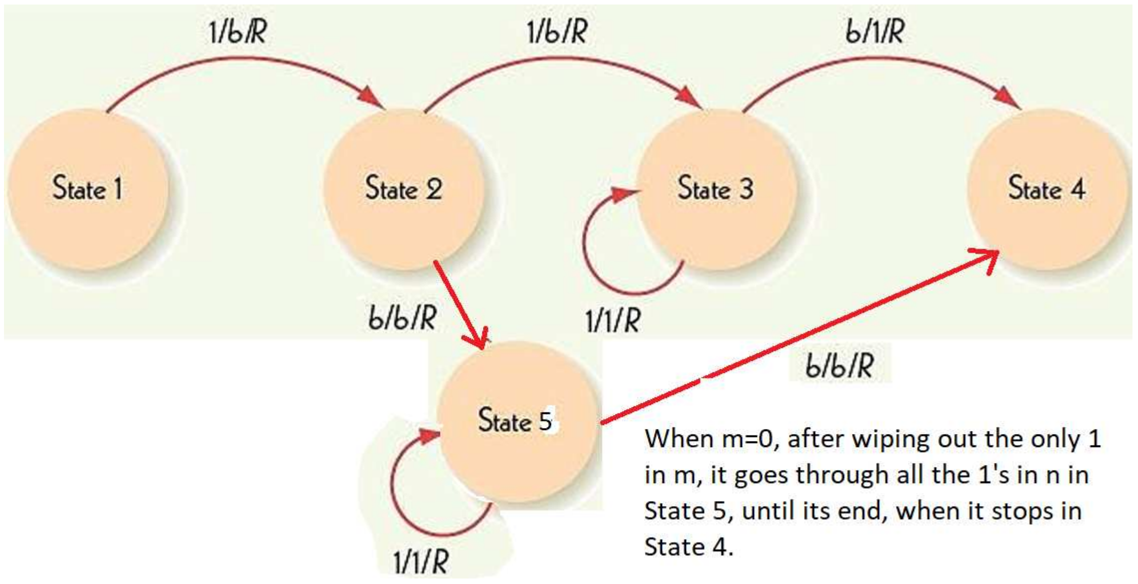
More specifically, in state 1, we wipe out the first 1, and go to State 2, where we wipe out the second 1, moving over to State 3, where we keep on moving to the right whenever we see a 1, until we see a blank, we flip it to 1, and stop with State 4.

Question: What happens if $m=0$, thus represented with one 1? ☹

Answer: We simply wipe out the first one 1, *if the next one is a blank,* move over to the right until we see a blank. We end up with $n+1$ 1’s, representing n. This even works when $n=0$. ☺

Thus, the unary addition function is unsurprisingly computable, by the above definition.

The revised TM



When m=0, after wiping out the only 1 in m, it goes through all the 1's in n in State 5, until its end, when it stops in State 4.

This TM consists of six instructions:

```
(1, 1, b, 2, R)
(2, 1, b, 3, R)
(2, b, b, 5, R)
(3, 1, 1, 3, R)
(3, b, 1, 4, R)
(5, 1, 1, 5, R)
(5, b, b, 4, R)
```

Assignment: Play it out with 0+3, i.e.,

·	·	·	b	b	1	b	1	1	1	1	·	·	·

Formalize reduction

We used to say, on Page 1, that, when a problem A reduces to another one, $B, A \leq_m B$, we mean a solution to B can be used to solve A.

Technically, we have the following definition.

Let A and B be languages. By saying that A is **m***apping reducible* to B, $(A \leq_{\mathbf{m}} B)$, we mean that, for a *computable function*, $f : \Sigma^* \mapsto \Sigma^*$, for every $w \in \Sigma^*$,

$$w \in A \quad \text{iff} \quad f(w) \in B.$$

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

In other words, to find out if $w \in A$, we just use the computable function f to figure out $f(w)$, then test if $f(w) \in B$.

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.

To test if $w \in A$, we can use the reduction to effectively compute $f(w)$ and test if $f(w) \in B$. By definition, we can provide a definitive answer for the former question on A based on the result obtained from the latter question on B : $w \in A$ iff $f(w) \in B$.

In particular, if one problem is mapping reducible to another, 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 $f(w)$. //That's why f has to be computable.

2. Run M on $f(w)$.

3. Accept w, if M accepts $f(w)$; reject w otherwise."

Since $A \leq_m B \equiv B \rightarrow A \equiv \neg A \rightarrow \neg B,$ we have

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

This sits underneath everything that we are going through in this chapter.

Back to the past

We once informally showed, on Pages 9 through 11, that the $HALT_{\mathsf{TM}}$ is undecidable by reducing A_{TM} to $HALT_{\mathsf{TM}}$. Now, we look into this result from this new perspective of mapping reduction.

First, 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

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

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 $x$, $M'$ accepts $w$.
 1.3. Otherwise, $M'$ enters a loop.
2. Output $\langle M', w\rangle$."
```

The final kick

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

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

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

On the other hand, if $\langle M, w\rangle \notin A_{\mathsf{TM}}$, i.e., either M rejects w *or gets into an infinite loop,* M' would get into an infinite loop *in both cases,* thus, won't halt with w. $\langle M', w\rangle \notin HALT_{\mathsf{TM}}$.

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

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

Guess we have to stop here. ☺