

## Day23

Recall definition:  $A$  is **mapping reducible** to  $B$  means there is a computable function  $f : \Sigma^* \rightarrow \Sigma^*$  such that *for all* strings  $x$  in  $\Sigma^*$ ,

$$x \in A \quad \text{if and only if} \quad f(x) \in B.$$

Notation: when  $A$  is mapping reducible to  $B$ , we write  $A \leq_m B$ .

**Theorem** (Sipser 5.23): If  $A \leq_m B$  and  $A$  is undecidable, then  $B$  is undecidable.

*Last time* we proved that  $A_{TM} \leq_m HALT_{TM}$  where

$$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a Turing machine, } w \text{ is a string, and } M \text{ halts on } w \}$$

and since  $A_{TM}$  is undecidable,  $HALT_{TM}$  is also undecidable. The function witnessing the mapping reduction mapped strings in  $A_{TM}$  to strings in  $HALT_{TM}$  and strings not in  $A_{TM}$  to strings not in  $HALT_{TM}$  by changing encoded Turing machines to ones that had identical computations except looped instead of rejecting.

True or False:  $\overline{A_{TM}} \leq_m \overline{HALT_{TM}}$

True or False:  $HALT_{TM} \leq_m A_{TM}$ .

**Proof:** Need computable function  $F : \Sigma^* \rightarrow \Sigma^*$  such that  $x \in HALT_{TM}$  iff  $F(x) \in A_{TM}$ . Define

$F =$  “ On input  $x$ ,

1. Type-check whether  $x = \langle M, w \rangle$  for some TM  $M$  and string  $w$ . If so, move to step 2; if not, output  $\langle \quad \quad \quad \rangle$
2. Construct the following machine  $M'_x$ :

3. Output  $\langle M'_x, w \rangle$ .”

Verifying correctness: (1) Is function well-defined and computable? (2) Does it have the translation property  $x \in HALT_{TM}$  iff its image is in  $A_{TM}$ ?

Input string	Output string
$\langle M, w \rangle$ where $M$ halts on $w$	
$\langle M, w \rangle$ where $M$ does not halt on $w$	
$x$ not encoding any pair of TM and string	

**Theorem** (Sipser 5.28): If  $A \leq_m B$  and  $B$  is recognizable, then  $A$  is recognizable.

**Proof:**

**Corollary:** If  $A \leq_m B$  and  $A$  is unrecognizable, then  $B$  is unrecognizable.

*Strategy:*

- (i) To prove that a recognizable language  $R$  is undecidable, prove that  $A_{TM} \leq_m R$ .
- (ii) To prove that a co-recognizable language  $U$  is undecidable, prove that  $\overline{A_{TM}} \leq_m U$ , i.e. that  $A_{TM} \leq_m \overline{U}$ .

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

Can we find algorithms to recognize

$E_{TM}$  ?

$\overline{E_{TM}}$  ?

**Claim:**  $A_{TM} \leq_m \overline{E_{TM}}$ . And hence also  $\overline{A_{TM}} \leq_m E_{TM}$

**Proof:** Need computable function  $F : \Sigma^* \rightarrow \Sigma^*$  such that  $x \in A_{TM}$  iff  $F(x) \notin E_{TM}$ . Define

$F =$  “ On input  $x$ ,

1. Type-check whether  $x = \langle M, w \rangle$  for some TM  $M$  and string  $w$ . If so, move to step 2; if not, output  $\langle \quad \quad \quad \rangle$
2. Construct the following machine  $M'_x$ :

3. Output  $\langle M'_x \rangle$ .”

Verifying correctness: (1) Is function well-defined and computable? (2) Does it have the translation property  $x \in A_{TM}$  iff its image is **not** in  $E_{TM}$  ?

Input string	Output string
$\langle M, w \rangle$ where $w \in L(M)$	
$\langle M, w \rangle$ where $w \notin L(M)$	
$x$ not encoding any pair of TM and string	

# Day24

Recall:  $A$  is **mapping reducible to**  $B$ , written  $A \leq_m B$ , means there is a computable function  $f : \Sigma^* \rightarrow \Sigma^*$  such that *for all* strings  $x$  in  $\Sigma^*$ ,

$$x \in A \quad \text{if and only if} \quad f(x) \in B.$$

So far:

- $A_{TM}$  is recognizable, undecidable, and not-co-recognizable.
- $\overline{A_{TM}}$  is unrecognizable, undecidable, and co-recognizable.
- $HALT_{TM}$  is recognizable, undecidable, and not-co-recognizable.
- $\overline{HALT_{TM}}$  is unrecognizable, undecidable, and co-recognizable.
- $E_{TM}$  is unrecognizable, undecidable, and co-recognizable.
- $\overline{E_{TM}}$  is recognizable, undecidable, and not-co-recognizable.

$$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are both Turing machines and } L(M_1) = L(M_2) \}$$

Can we find algorithms to recognize

$EQ_{TM}$  ?

$\overline{EQ_{TM}}$  ?

*Goal:* Show that  $EQ_{TM}$  is not recognizable and that  $\overline{EQ_{TM}}$  is not recognizable.

Using Corollary to **Theorem 5.28**: If  $A \leq_m B$  and  $A$  is unrecognizable, then  $B$  is unrecognizable, it's enough to prove that

$$\begin{array}{ll} \overline{HALT_{TM}} \leq_m EQ_{TM} & \text{aka } HALT_{TM} \leq_m \overline{EQ_{TM}} \\ \overline{HALT_{TM}} \leq_m \overline{EQ_{TM}} & \text{aka } HALT_{TM} \leq_m EQ_{TM} \end{array}$$

Need computable function  $F_1 : \Sigma^* \rightarrow \Sigma^*$  such that  $x \in HALT_{TM}$  iff  $F_1(x) \notin EQ_{TM}$ .

*Strategy:*

Map strings  $\langle M, w \rangle$  to strings  $\langle M'_x,$   $\rangle$ . This image string is not in  $EQ_{TM}$  when  $L(M'_x) \neq \emptyset$ .

We will build  $M'_x$  so that  $L(M'_x) = \Sigma^*$  when  $M$  halts on  $w$  and  $L(M'_x) = \emptyset$  when  $M$  loops on  $w$ .

Thus: when  $\langle M, w \rangle \in HALT_{TM}$  it gets mapped to a string not in  $EQ_{TM}$  and when  $\langle M, w \rangle \notin HALT_{TM}$  it gets mapped to a string that is in  $EQ_{TM}$ .

Define

$F_1 =$  “ On input  $x$ ,

1. Type-check whether  $x = \langle M, w \rangle$  for some TM  $M$  and string  $w$ . If so, move to step 2; if not, output  $\langle$
2. Construct the following machine  $M'_x$ :

3. Output  $\langle M'_x,$   $\rangle$ ”

Verifying correctness: (1) Is function well-defined and computable? (2) Does it have the translation property  $x \in HALT_{TM}$  iff its image is **not** in  $EQ_{TM}$  ?

Input string	Output string
$\langle M, w \rangle$ where $M$ halts on $w$	
$\langle M, w \rangle$ where $M$ loops on $w$	
$x$ not encoding any pair of TM and string	

Conclude:  $HALT_{TM} \leq_m \overline{EQ_{TM}}$

Need computable function  $F_2 : \Sigma^* \rightarrow \Sigma^*$  such that  $x \in HALT_{TM}$  iff  $F_2(x) \in EQ_{TM}$ .

*Strategy:*

Map strings  $\langle M, w \rangle$  to strings  $\langle M'_x, \text{start} \rightarrow \text{q}_0 \rangle$ . This image string is in  $EQ_{TM}$  when  $L(M'_x) = \Sigma^*$ .

We will build  $M'_x$  so that  $L(M'_x) = \Sigma^*$  when  $M$  halts on  $w$  and  $L(M'_x) = \emptyset$  when  $M$  loops on  $w$ .

Thus: when  $\langle M, w \rangle \in HALT_{TM}$  it gets mapped to a string in  $EQ_{TM}$  and when  $\langle M, w \rangle \notin HALT_{TM}$  it gets mapped to a string that is not in  $EQ_{TM}$ .

Define

$F_2 =$  “ On input  $x$ ,

1. Type-check whether  $x = \langle M, w \rangle$  for some TM  $M$  and string  $w$ . If so, move to step 2; if not, output  $\langle \text{start} \rightarrow \text{q}_0 \rangle$
2. Construct the following machine  $M'_x$ :

3. Output  $\langle M'_x, \text{start} \rightarrow \text{q}_0 \rangle$  ”

Verifying correctness: (1) Is function well-defined and computable? (2) Does it have the translation property  $x \in HALT_{TM}$  iff its image is in  $EQ_{TM}$  ?

Input string	Output string
$\langle M, w \rangle$ where $M$ halts on $w$	
$\langle M, w \rangle$ where $M$ loops on $w$	
$x$ not encoding any pair of TM and string	

Conclude:  $HALT_{TM} \leq_m EQ_{TM}$

# Day25

Two models of computation are called **equally expressive** when every language recognizable with the first model is recognizable with the second, and vice versa.

True / False: NFAs and PDAs are equally expressive.

True / False: Regular expressions and CFGs are equally expressive.

**Church-Turing Thesis** (Sipser p. 183): The informal notion of algorithm is formalized completely and correctly by the formal definition of a Turing machine. In other words: all reasonably expressive models of computation are equally expressive with the standard Turing machine.

*Some examples of models that are **equally expressive** with deterministic Turing machines:*

**May-stay machines** The May-stay machine model is the same as the usual Turing machine model, except that on each transition, the tape head may move L, move R, or Stay.

Formally:  $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$  where

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$

**Claim:** Turing machines and May-stay machines are equally expressive. *To prove ...*

To translate a standard TM to a may-stay machine: never use the direction *S*!

To translate one of the may-stay machines to standard TM: any time TM would Stay, move right then left.

**Multitape Turing machine** A multitape Turing machine with  $k$  tapes can be formally represented as  $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$  where  $Q$  is the finite set of states,  $\Sigma$  is the input alphabet with  $\sqcup \notin \Sigma$ ,  $\Gamma$  is the tape alphabet with  $\Sigma \subsetneq \Gamma$ ,  $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$  (where  $k$  is the number of states)

If  $M$  is a standard TM, it is a 1-tape machine.

To translate a  $k$ -tape machine to a standard TM: Use a new symbol to separate the contents of each tape and keep track of location of head with special version of each tape symbol. Sipser Theorem 3.13



FIGURE 3.14  
Representing three tapes with one

**Enumerators** Enumerators give a different model of computation where a language is **produced, one string at a time**, rather than recognized by accepting (or not) individual strings.

Each enumerator machine has finite state control, unlimited work tape, and a printer. The computation proceeds according to transition function; at any point machine may “send” a string to the printer.

$$E = (Q, \Sigma, \Gamma, \delta, q_0, q_{print})$$

$Q$  is the finite set of states,  $\Sigma$  is the output alphabet,  $\Gamma$  is the tape alphabet ( $\Sigma \subsetneq \Gamma, \sqcup \in \Gamma \setminus \Sigma$ ),

$$\delta : Q \times \Gamma \times \Gamma \rightarrow Q \times \Gamma \times \Gamma \times \{L, R\} \times \{L, R\}$$

where in state  $q$ , when the working tape is scanning character  $x$  and the printer tape is scanning character  $y$ ,  $\delta((q, x, y)) = (q', x', y', d_w, d_p)$  means transition to control state  $q'$ , write  $x'$  on the working tape, write  $y'$  on the printer tape, move in direction  $d_w$  on the working tape, and move in direction  $d_p$  on the printer tape. The computation starts in  $q_0$  and each time the computation enters  $q_{print}$  the string from the leftmost edge of the printer tape to the first blank cell is considered to be printed.

The language **enumerated** by  $E$ ,  $L(E)$ , is  $\{w \in \Sigma^* \mid E \text{ eventually, at finite time, prints } w\}$ .

**Theorem 3.21** A language is Turing-recognizable iff some enumerator enumerates it.

**Proof, part 1:** Assume  $L$  is enumerated by some enumerator,  $E$ , so  $L = L(E)$ . We'll use  $E$  in a subroutine within a high-level description of a new Turing machine that we will build to recognize  $L$ .

**Goal:** build Turing machine  $M_E$  with  $L(M_E) = L(E)$ .

Define  $M_E$  as follows:  $M_E =$  “On input  $w$ ,

1. Run  $E$ . For each string  $x$  printed by  $E$ .
2. Check if  $x = w$ . If so, accept (and halt); otherwise, continue.”

**Proof, part 2:** Assume  $L$  is Turing-recognizable and there is a Turing machine  $M$  with  $L = L(M)$ . We'll use  $M$  in a subroutine within a high-level description of an enumerator that we will build to enumerate  $L$ .

**Goal:** build enumerator  $E_M$  with  $L(E_M) = L(M)$ .

**Idea:** check each string in turn to see if it is in  $L$ .

*How?* Run computation of  $M$  on each string. *But:* need to be careful about computations that don't halt.

*Recall* String order for  $\Sigma = \{0, 1\}$ :  $s_1 = \varepsilon$ ,  $s_2 = 0$ ,  $s_3 = 1$ ,  $s_4 = 00$ ,  $s_5 = 01$ ,  $s_6 = 10$ ,  $s_7 = 11$ ,  $s_8 = 000$ , ...

Define  $E_M$  as follows:  $E_M =$  “*ignore any input*. Repeat the following for  $i = 1, 2, 3, \dots$

1. Run the computations of  $M$  on  $s_1, s_2, \dots, s_i$  for (at most)  $i$  steps each
2. For each of these  $i$  computations that accept during the (at most)  $i$  steps, print out the accepted string.”



## Nondeterministic Turing machine

At any point in the computation, the nondeterministic machine may proceed according to several possibilities:  $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$  where

$$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

The computation of a nondeterministic Turing machine is a tree with branching when the next step of the computation has multiple possibilities. A nondeterministic Turing machine accepts a string exactly when some branch of the computation tree enters the accept state.

Given a nondeterministic machine, we can use a 3-tape Turing machine to simulate it by doing a breadth-first search of computation tree: one tape is “read-only” input tape, one tape simulates the tape of the nondeterministic computation, and one tape tracks nondeterministic branching. Sipser page 178

## Summary

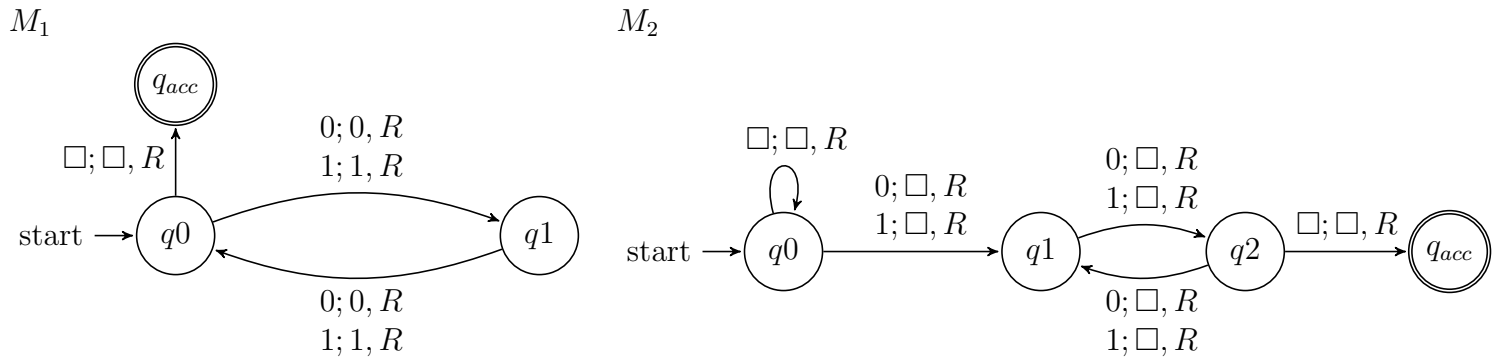
Two models of computation are called **equally expressive** when every language recognizable with the first model is recognizable with the second, and vice versa.

To prove the existence of a Turing machine that decides / recognizes some language, it’s enough to construct an example using any of the equally expressive models.

But: some of the **performance** properties of these models are not equivalent.

# Day20

<b>Acceptance problem</b>	
for Turing machines	$A_{TM} \quad \{\langle M, w \rangle \mid M \text{ is a Turing machine that accepts input string } w\}$
<b>Language emptiness testing</b>	
for Turing machines	$E_{TM} \quad \{\langle M \rangle \mid M \text{ is a Turing machine and } L(M) = \emptyset\}$
<b>Language equality testing</b>	
for Turing machines	$EQ_{TM} \quad \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are Turing machines and } L(M_1) = L(M_2)\}$



Example strings in  $A_{TM}$

Example strings in  $E_{TM}$

Example strings in  $EQ_{TM}$

**Theorem:**  $A_{TM}$  is Turing-recognizable.

**Strategy:** To prove this theorem, we need to define a Turing machine  $R_{ATM}$  such that  $L(R_{ATM}) = A_{TM}$ .

Define  $R_{ATM} =$  “

Proof of correctness:

We will show that  $A_{TM}$  is undecidable. *First, let's explore what that means.*

To prove that a computational problem is **decidable**, we find/ build a Turing machine that recognizes the language encoding the computational problem, and that is a decider.

How do we prove a specific problem is **not decidable**?

How would we even find such a computational problem?

*Counting arguments for the existence of an undecidable language:*

- The set of all Turing machines is countably infinite.
- Each recognizable language has at least one Turing machine that recognizes it (by definition), so there can be no more Turing-recognizable languages than there are Turing machines.
- Since there are infinitely many Turing-recognizable languages (think of the singleton sets), there are countably infinitely many Turing-recognizable languages.
- Such the set of Turing-decidable languages is an infinite subset of the set of Turing-recognizable languages, the set of Turing-decidable languages is also countably infinite.

Since there are uncountably many languages (because  $\mathcal{P}(\Sigma^*)$  is uncountable), there are uncountably many unrecognizable languages and there are uncountably many undecidable languages.

Thus, there's at least one undecidable language!

**What's a specific example of a language that is unrecognizable or undecidable?**

To prove that a language is undecidable, we need to prove that there is no Turing machine that decides it.

**Key idea:** proof by contradiction relying on self-referential disagreement.

**Theorem:**  $A_{TM}$  is not Turing-decidable.

**Proof:** Suppose **towards a contradiction** that there is a Turing machine that decides  $A_{TM}$ . We call this presumed machine  $M_{ATM}$ .

By assumption, for every Turing machine  $M$  and every string  $w$

- If  $w \in L(M)$ , then the computation of  $M_{ATM}$  on  $\langle M, w \rangle$  \_\_\_\_\_
- If  $w \notin L(M)$ , then the computation of  $M_{ATM}$  on  $\langle M, w \rangle$  \_\_\_\_\_

Define a **new** Turing machine using the high-level description:

$D =$  “ On input  $\langle M \rangle$ , where  $M$  is a Turing machine:

1. Run  $M_{ATM}$  on  $\langle M, \langle M \rangle \rangle$ .
2. If  $M_{ATM}$  accepts, reject; if  $M_{ATM}$  rejects, accept.”

Is  $D$  a Turing machine?

Is  $D$  a decider?

What is the result of the computation of  $D$  on  $\langle D \rangle$ ?

### Summarizing:

- $A_{TM}$  is recognizable.
- $A_{TM}$  is not decidable.

Recall definition: A language  $L$  over an alphabet  $\Sigma$  is called **co-recognizable** if its complement, defined as  $\Sigma^* \setminus L = \{x \in \Sigma^* \mid x \notin L\}$ , is Turing-recognizable.

and Recall Theorem (Sipser Theorem 4.22): A language is Turing-decidable if and only if both it and its complement are Turing-recognizable.

- $A_{TM}$  is recognizable.
- $A_{TM}$  is not decidable.
- $\overline{A_{TM}}$  is not recognizable.
- $\overline{A_{TM}}$  is not decidable.

# Day21

## Mapping reduction

Motivation: Proving that  $A_{TM}$  is undecidable was hard. How can we leverage that work? Can we relate the decidability / undecidability of one problem to another?

If problem  $X$  is **no harder than** problem  $Y$   
... and if  $Y$  is easy,  
... then  $X$  must be easy too.

If problem  $X$  is **no harder than** problem  $Y$   
... and if  $X$  is hard,  
... then  $Y$  must be hard too.

“Problem  $X$  is no harder than problem  $Y$ ” means “Can answer questions about membership in  $X$  by converting them to questions about membership in  $Y$ ”.

Definition: For any languages  $A$  and  $B$ ,  $A$  is **mapping reducible to**  $B$  means there is a computable function  $f : \Sigma^* \rightarrow \Sigma^*$  such that *for all* strings  $x$  in  $\Sigma^*$ ,

$$x \in A \quad \text{if and only if} \quad f(x) \in B.$$

Notation: when  $A$  is mapping reducible to  $B$ , we write  $A \leq_m B$ .

*Intuition:*  $A \leq_m B$  means  $A$  is no harder than  $B$ , i.e. that the level of difficulty of  $A$  is less than or equal the level of difficulty of  $B$ .

## TODO

1. What is a computable function?
2. How do mapping reductions help establish the computational difficulty of languages?

## Computable functions

Definition: A function  $f : \Sigma^* \rightarrow \Sigma^*$  is a **computable function** means there is some Turing machine such that, for each  $x$ , on input  $x$  the Turing machine halts with exactly  $f(x)$  followed by all blanks on the tape

*Examples of computable functions:*

The function that maps a string to a string which is one character longer and whose value, when interpreted as a fixed-width binary representation of a nonnegative integer is twice the value of the input string (when interpreted as a fixed-width binary representation of a non-negative integer)

$$f_1 : \Sigma^* \rightarrow \Sigma^* \quad f_1(x) = x0$$

To prove  $f_1$  is computable function, we define a Turing machine computing it.

*High-level description*

“On input  $w$

1. Append 0 to  $w$ .
2. Halt.”

*Implementation-level description*

“On input  $w$

1. Sweep read-write head to the right until find first blank cell.
2. Write 0.
3. Halt.”

*Formal definition* ( $\{q_0, q_{acc}, q_{rej}\}, \{0, 1\}, \{0, 1, \sqcup\}, \delta, q_0, q_{acc}, q_{rej}$ ) where  $\delta$  is specified by the state diagram:



The function that maps a string to the result of repeating the string twice.

$$f_2 : \Sigma^* \rightarrow \Sigma^* \quad f_2(x) = xx$$

The function that maps strings that are not the codes of NFAs to the empty string and that maps strings that code NFAs to the code of a DFA that recognizes the language recognized by the NFA produced by the macro-state construction from Chapter 1.

The function that maps strings that are not the codes of Turing machines to the empty string and that maps strings that code Turing machines to the code of the related Turing machine that acts like the Turing machine coded by the input, except that if this Turing machine coded by the input tries to reject, the new machine will go into a loop.

$$f_4 : \Sigma^* \rightarrow \Sigma^* \quad f_4(x) = \begin{cases} \varepsilon & \text{if } x \text{ is not the code of a TM} \\ \langle (Q \cup \{q_{trap}\}, \Sigma, \Gamma, \delta', q_0, q_{acc}, q_{rej}) \rangle & \text{if } x = \langle (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}) \rangle \end{cases}$$

where  $q_{trap} \notin Q$  and

$$\delta'((q, x)) = \begin{cases} (r, y, d) & \text{if } q \in Q, x \in \Gamma, \delta((q, x)) = (r, y, d), \text{ and } r \neq q_{rej} \\ (q_{trap}, \sqcup, R) & \text{otherwise} \end{cases}$$

Definition:  $A$  is **mapping reducible to**  $B$ ,  $A \leq_m B$  means there is a computable function  $f : \Sigma^* \rightarrow \Sigma^*$  such that *for all* strings  $x$  in  $\Sigma^*$ ,

$$x \in A \quad \text{if and only if} \quad f(x) \in B.$$

In this case, we say the function  $f$  **witnesses** that  $A$  is mapping reducible to  $B$ .

*Making intuition precise ...*

**Theorem** (Sipser 5.22): If  $A \leq_m B$  and  $B$  is decidable, then  $A$  is decidable.

**Theorem** (Sipser 5.23): If  $A \leq_m B$  and  $A$  is undecidable, then  $B$  is undecidable.

## Day22

Recall definition:  $A$  is **mapping reducible to**  $B$  means there is a computable function  $f : \Sigma^* \rightarrow \Sigma^*$  such that *for all* strings  $x$  in  $\Sigma^*$ ,

$$x \in A \quad \text{if and only if} \quad f(x) \in B.$$

Notation: when  $A$  is mapping reducible to  $B$ , we write  $A \leq_m B$ .

*Intuition:*  $A \leq_m B$  means  $A$  is no harder than  $B$ , i.e. that the level of difficulty of  $A$  is less than or equal the level of difficulty of  $B$ .

*Example:*  $A_{TM} \leq_m A_{TM}$   $A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and } w \text{ is a string and } w \in L(M)\}$

*To prove, need a witnessing function  $f : \Sigma^* \rightarrow \Sigma^*$  that is (1) computable and (2) for each  $x \in \Sigma^*$ ,  $x \in A_{TM}$  iff  $f(x) \in A_{TM}$*

**Corollary:** For any language  $L$ ,  $L \leq_m L$ , as witnessed by

*Example:*  $A_{DFA} \leq_m \{ww \mid w \in \{0,1\}^*\}$

*To prove, need a witnessing function  $f : \Sigma^* \rightarrow \Sigma^*$  that is (1) computable and (2) for each  $x \in \Sigma^*$ ,  $x \in A_{DFA}$  iff  $f(x) \in \{ww \mid w \in \{0,1\}^*\}$*

**Corollary:** For any language decidable language  $X$  and any set  $Y$  with at least one string string in  $Y$  and at least one string not in  $Y$ ,  $X \leq_m Y$ , as witnessed by

Next: consider mapping reductions between potentially undecidable languages.

## Halting problem

$$HALT_{TM} = \{\langle M, w \rangle \mid M \text{ is a Turing machine, } w \text{ is a string, and } M \text{ halts on } w\}$$

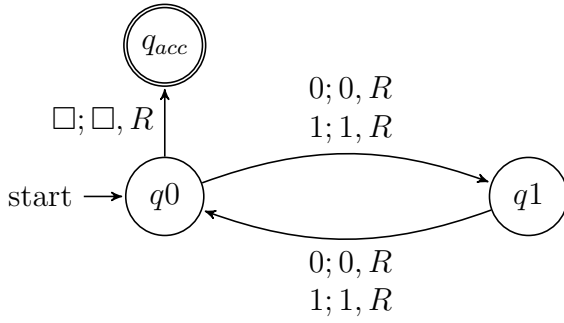
We know  $A_{TM}$  is undecidable. If we could prove that  $A_{TM} \leq_m HALT_{TM}$  then we could conclude that  $HALT_{TM}$  is undecidable too.

Could we adapt our approach from before by tweaking the identity map?

Define  $F : \Sigma^* \rightarrow \Sigma^*$  by

$$F(x) = \begin{cases} const_{out} & \text{if } x \neq \langle M, w \rangle \text{ for any Turing machine } M \text{ and string } w \text{ over the alphabet of } M \\ \langle M'_x, w \rangle & \text{if } x = \langle M, w \rangle \text{ for some Turing machine } M \text{ and string } w \text{ over the alphabet of } M. \end{cases}$$

where  $const_{out} = \langle \text{start} \rightarrow q0 \quad \text{ } \quad q_{acc}, \varepsilon \rangle$  and  $M'_x$  is a Turing machine that computes like  $M$  except, if the computation of  $M$  ever were to go to a reject state,  $M'_x$  loops instead.



$$F(\langle \quad \quad \quad , \varepsilon \rangle) =$$

To use this function to prove that  $A_{TM} \leq_m HALT_{TM}$ , we need two claims:

Claim (1):  $F$  is computable

Claim (2): for every  $x$ ,  $x \in A_{TM}$  iff  $F(x) \in HALT_{TM}$ .