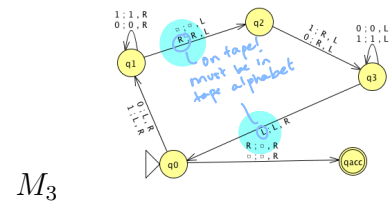
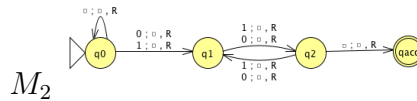
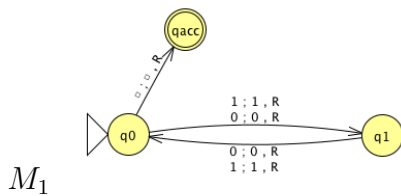


# Monday: $A_{TM}$ is recognizable but undecidable

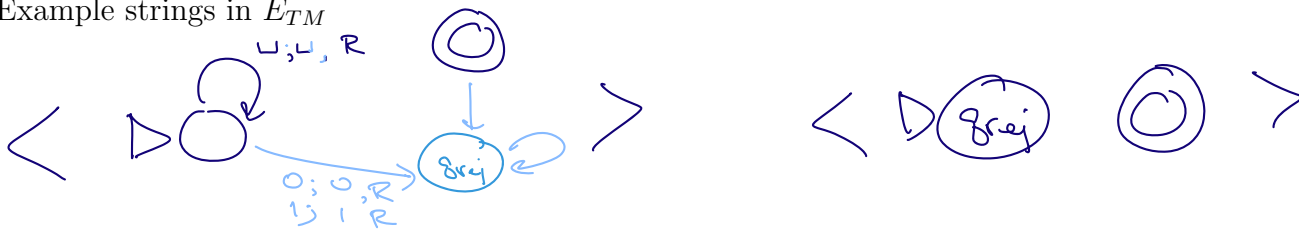
<b>Acceptance problem</b>	
for Turing machines $A_{TM}$	$\{\langle M, w \rangle \mid M \text{ is a Turing machine that accepts input string } w\}$
<b>Language emptiness testing</b>	
for Turing machines $E_{TM}$	$\{\langle M \rangle \mid M \text{ is a Turing machine and } L(M) = \emptyset\}$
<b>Language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



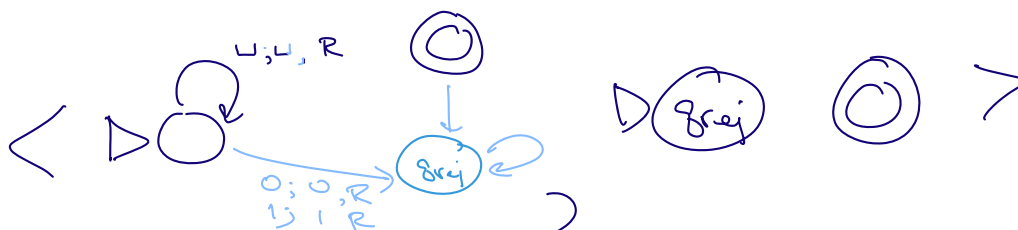
(For these TMs,  $\Sigma = \{0,1\}$   $\Gamma = \{0,1,\sqcup, L, R\}$ )  
 Example strings in  $A_{TM}$

$\langle M_1, 01 \rangle$

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  $R_{ATM} =$  " On input  $x$

1. If  $x \neq \langle M, w \rangle$   $M$  Turing machine,  $w$  a string  
reject
- 2 Otherwise  $x = \langle M, w \rangle$   $M$  Turing machine,  $w$  a string  
Simulate  $M$  on  $w$
- 3 If  $M$  accepts  $w$ , accept  $x$ .
4. If  $M$  rejects  $w$ , reject  $x$  "

Proof of correctness:

Need two subset inclusions to prove  $L(R_{ATM}) = A_{TM}$ .

- ① WTS for each  $x \in \Sigma^*$  if  $x \in A_{TM}$  then  $R_{ATM}$  accepts  $x$ .  
Consider arbitrary  $x \in \Sigma^*$  and assume  $x \in A_{TM}$ . By definition of  $A_{TM}$   $x = \langle M, w \rangle$  for some TM  $M$  and string  $w$  with  $w \in L(M)$ . Tracing  $R_{ATM}$  on  $x$ , step 1 type check passes and in step 2 simulate  $M$  on  $w$  so since  $M$  accepts  $w$ , in step 3  $R_{ATM}$  accepts  $x$ . does not
- ② WTS for each  $x \in \Sigma^*$  if  $x \notin A_{TM}$  then  $R_{ATM}$  reject  $x$ .  
Consider arbitrary  $x \in \Sigma^*$  and assume  $x \notin A_{TM}$ .  
Case 2a:  $x \neq \langle M, w \rangle$  for any  $M$  TM,  $w$  string. Then  $R_{ATM}$  reject  $x$  in step 1.  
Case 2b:  $x = \langle M, w \rangle$ ,  $M$  rejects  $w$ . Then  $R_{ATM}$  rejects  $x$  in step 4.  
Case 2c:  $x = \langle M, w \rangle$ ,  $M$  loops on  $w$ . Then  $R_{ATM}$  loops on  $x$  in step 2.

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  $\langle M, w \rangle$  accepts
  - If  $w \notin L(M)$ , then the computation of  $M_{ATM}$  on  $\langle M, w \rangle$  rejects
- Handwritten notes:*  $\langle M, w \rangle \in A_{TM}$  (above first bullet),  $\langle M, w \rangle \notin A_{TM}$  (above second bullet),  $M$  rejects  $w$  (next to  $w \notin L(M)$ ),  $M$  loops on  $w$  (next to  $w \notin L(M)$ ).

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."

*Handwritten notes:* "string" (above  $\langle M \rangle$ ), "Turing machine" (above  $M$ ), "string representing Turing machine" (above the input to  $M_{ATM}$  in step 1).

Is  $D$  a Turing machine? Yes.

Is  $D$  a decider? Yes.

*Handwritten notes:* step 1: finite time b/c  $M_{ATM}$  is a decider; step 2: Boolean so it also takes finite time.

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

Case ①  $D$  halts and accepts  $\langle D \rangle$   
 Then  $\langle D, \langle D \rangle \rangle \in A_{TM}$  i.e.  $M_{ATM}$  accepts  $\langle D, \langle D \rangle \rangle$   
 Tracing  $D$ : step 1 Run  $M_{ATM}$  on  $\langle D, \langle D \rangle \rangle$  step 2. Reject  $\langle D \rangle$  !  $\rightarrow \leftarrow$

Case ②  $D$  halts and rejects  $\langle D \rangle$

Then  $\langle D, \langle D \rangle \rangle \notin A_{TM}$  i.e.  $M_{ATM}$  rejects  $\langle D, \langle D \rangle \rangle$   
 Tracing  $D$ : step 1 Run  $M_{ATM}$  on  $\langle D, \langle D \rangle \rangle$  step 2: Accept  $\langle D \rangle$ !  $\rightarrow$

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.

$A_{TM}$  is recognizable and undecidable.

$\overline{A_{TM}}$  is co-recognizable b/c  $\overline{\overline{A_{TM}}} = A_{TM}$

once we have Theorem 4.22, putting these together gives that  $\overline{A_{TM}}$  is not recognizable.

But:  $\Sigma^*$ ,  $\emptyset$ ,  $A_{PFA}$  recognizable and co-recognizable.

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

**Proof, first direction:** Suppose language  $L$  is Turing-decidable. WTS that both it and its complement are Turing-recognizable.

Let  $L$  be decidable. By definition, there is a decider, call it  $M_L$  that recognizes  $L$ . Then  $M_L$  witnesses that  $L$  is recognizable. By closure of the class of decidable languages under complementation,  $\overline{L}$  is also decidable, hence also recognizable.

**Proof, second direction:** Suppose language  $L$  is Turing-recognizable, and so is its complement. WTS that  $L$  is Turing-decidable.

Suppose  $L$  is arbitrary language that is recognizable and corecognizable. Let  $M$  and  $M_{comp}$  be TMs recognizing  $L$  and  $\overline{L}$ , respectively. Define the new TM

$D =$  "On input  $x$

1. For  $n = 1, 2, 3, \dots$
2. Run  $M$  on  $x$  for at most  $n$  steps. If it halts and accepts, accept; if it halts and rejects, reject.
3. Run  $M_{comp}$  on  $x$  for at most  $n$  steps. If it halts and accepts, reject; if it halts and rejects, accept.
4. increment  $n$  and go to next loop iteration."

Notation: The complement of a set  $X$  is denoted with a superscript  $c$ ,  $X^c$ , or an overline,  $\overline{X}$ .

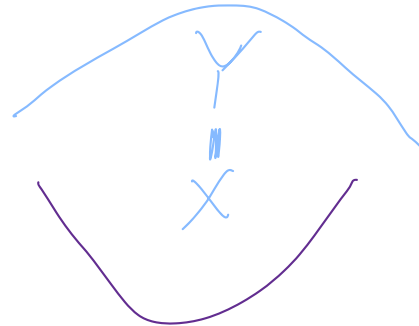
## Wednesday: Computable functions and reduction

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

$x \in A$  if and only if  $f(x) \in B$ .

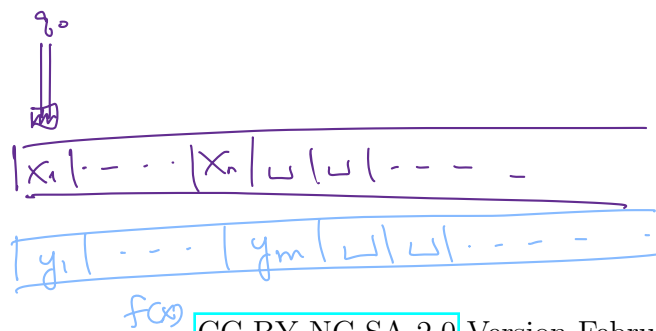
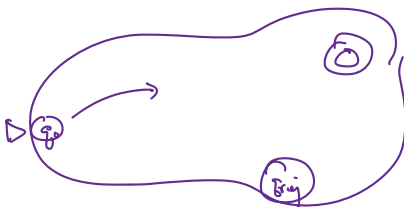
Notation: when  $A$  is mapping reducible to  $B$ , we write  $A \leq_m B$ .

*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  $B$ .

To do

① what is a computable function? ✓

② How do mapping reductions help? ↩



for

Use Turing machines, interpreted a little differently

## Computable functions

a Turing machine computing  $f(x)$

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)

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

← shift in binary multiplies by 2

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.”

Test cases

$$f_1(0) = 00$$

$$f_1(10) = 100$$

$$f_1(\epsilon) = 0$$

Implementation-level description

“On input  $w$

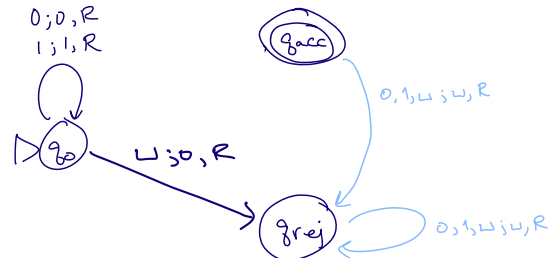
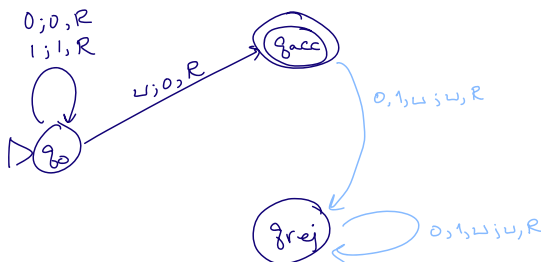
1. Sweep read-write head to the right until find first blank cell.
2. Write 0.
3. Halt.”

3 states

$\Sigma$

$\Gamma$

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:



OR

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

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

" On input  $x$   
1. Output  $xx$  " .

Extra practice: state diagram.

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.

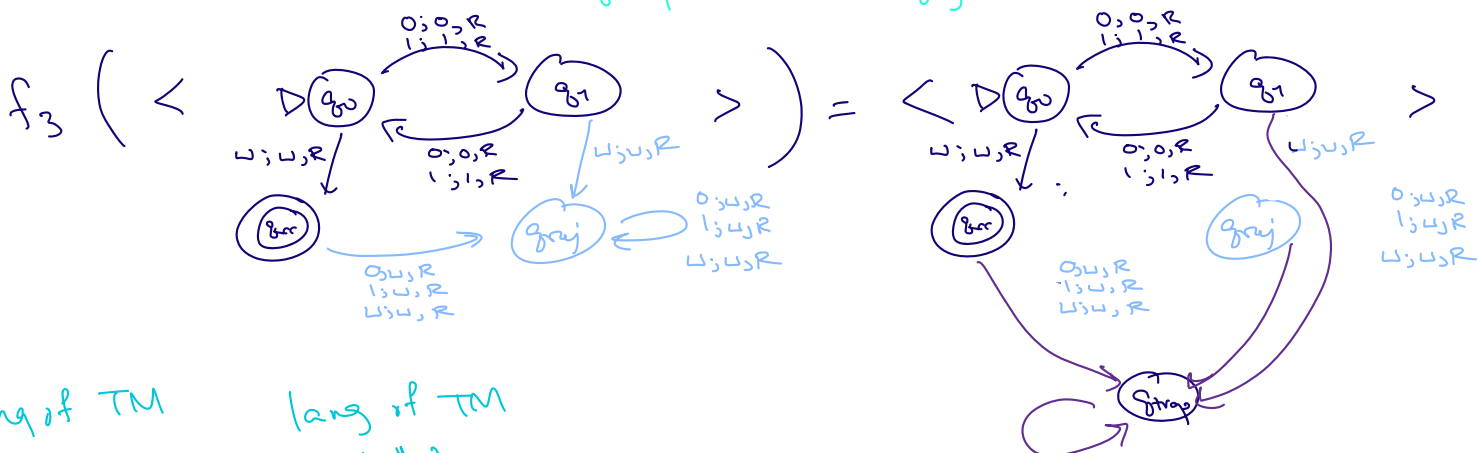
$$f_3 : \Sigma^* \rightarrow \Sigma^* \quad f_3(x) = \begin{cases} \varepsilon & \text{if } x \text{ is not the code of a TM} \\ \langle \langle Q \cup \{q_{trap}\}, \Sigma, \Gamma, \delta', q_0, q_{acc}, q_{rej} \rangle \rangle & \text{if } x = \langle \langle Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej} \rangle \rangle \end{cases}$$

where  $q_{trap} \notin Q$  and

*Handwritten notes:*   
 -  $\varepsilon$ : default output   
 -  $\langle \langle Q \cup \{q_{trap}\}, \Sigma, \Gamma, \delta', q_0, q_{acc}, q_{rej} \rangle \rangle$ : one more state   
 -  $\langle \langle Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej} \rangle \rangle$ : string representing a TM   
 -  $\langle \langle Q \cup \{q_{trap}\}, \Sigma, \Gamma, \delta', q_0, q_{acc}, q_{rej} \rangle \rangle$ : piecewise definition   
 -  $\delta'$ : string

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

*Handwritten notes:*   
 -  $(q_{trap}, -, R)$ : redirect to  $q_{trap}$  instead of  $q_{rej}$    
 -  $r \neq q_{rej}$ : arrow doesn't point to reject state don't get changed.



lang of TM  
coded by  
input = lang of TM  
coded by  
output of  $f$

$f_3$  is computable!

" On input  $x$

1. If  $x \neq \langle M \rangle$  for any TM  $M$ , output  $\epsilon$ .

2. If  $x = \langle M \rangle$  output  $\langle \dots \rangle$   
by adopting set of states and  
transition function of  $M$ .



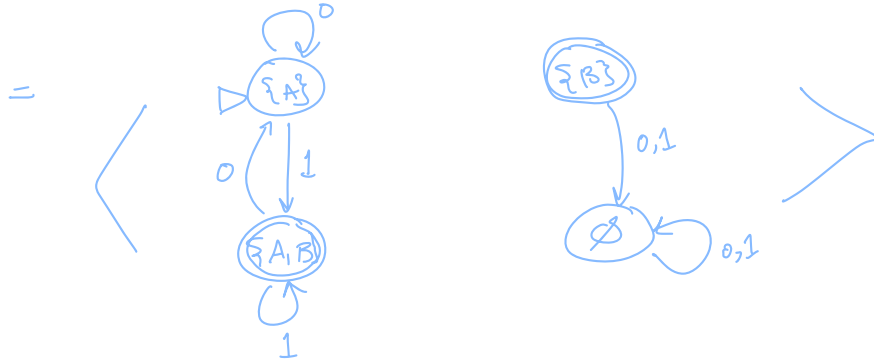
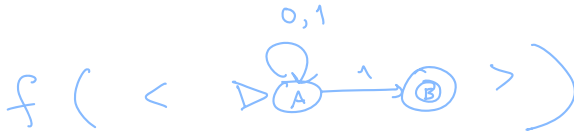
## NFAS

The function that maps strings that are not the codes of ~~CFGs~~ to the empty string and that maps strings that code ~~CFGs~~ to the code of a ~~PDA~~ that recognizes the language generated by the CFG.

NFA

DFA

recognized by the NFA.  
produced by the  
macro state construction



" On input  $x$

1. If  $x$  is not the code of an NFA, output  $\epsilon$ .
2. If  $x = \langle N \rangle$  for some NFA, use the macro state construction from ch 1 to produce DFA  $D$  with  $L(D) = L(N)$
3. Output  $\langle D \rangle$  "

The function that maps strings that are not the codes of CFGs to the empty string and that maps strings that code CFGs to the code of a PDA that recognizes the language generated by the CFG.

extra ex.

Other examples?

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  $B$ .

**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.



Pf of 5.22.

Given languages  $A, B$

with

$$A \leq_m B$$

and

$B$  decidable

Given  $M_B$  decides  $B$

Given TM  $F$  that computes function  $f: \Sigma^* \rightarrow \Sigma^*$  w/  $x \in A \iff f(x) \in B$ .  
WTs  $A$  is decidable.

Need TM decides  $A$ .

"On input  $x$

1. Use TM  $F$  to compute  $f(x)$ .
2. Run  $M_B$  on  $f(x)$ .
3. If  $M_B$  accepts  $f(x)$ , accept.
4. If  $M_B$  rejects  $f(x)$ , reject"

Claim this TM works . . . .



# Friday: The Halting problem

witness

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  $\Sigma^*$ ,

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

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:  $A_{TM} \leq_m A_{TM}$

Consider  $f: \Sigma^* \rightarrow \Sigma^*$

$$f(x) = x$$

computable? Yes, computed by

"On input  $x$   
1. output  $x$ "



translation? Yes, for each

string  $x$ ,

$$x \in A_{TM} \quad \text{iff} \quad x \in A_{TM}.$$

Generalize: identity function witnesses that  $X \leq_m X$  for all languages  $X$ .

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

Have decider  $M_D$  with  $L(M_D) = A_{DFA}$

decidable!

Consider function computed by

"On input  $x$

1. Run  $M_D$  on  $x$ .

2. If  $M_D$  accepts, output  $00$

3. If  $M_D$  rejects, output  $0$ "

always in one of these cases.

$$\in \{ww \mid w \in \{0,1\}^*\}$$

Translation?

When  $x \in A_{DFA}$ : function outputs  $00$  b/c  $M_D$  accepts  $x$

When  $x \notin A_{DFA}$ : function outputs  $0$  b/c  $M_D$  rejects  $x$

**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. \*

$A_{TM}$  is undecidable

$\overline{A_{TM}}$  is undecidable

Strategy: To prove  $B$  is undecidable, try to prove

$A_{TM} \leq_m B$

or

$\overline{A_{TM}} \leq_m B$

example in  $HALT_{TM}$  and not in  $A_{TM}$ :  $\langle \text{loop}, \epsilon \rangle$   
 example in  $A_{TM}$  and not in  $HALT_{TM}$ : IMPOSSIBLE

## Halting problem

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

GOAL PROVE  $A_{TM} \leq_m HALT_{TM}$  by finding a witness function

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

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

$\square; \square, R$   
 $1; 1, R$   
 $0; 0, R$



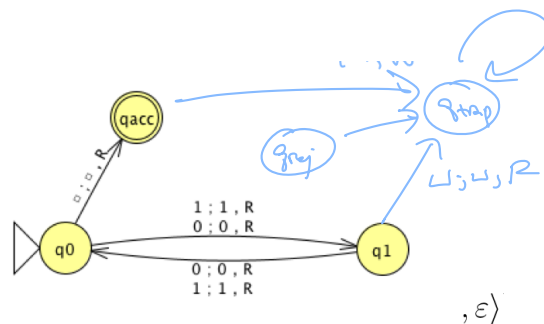
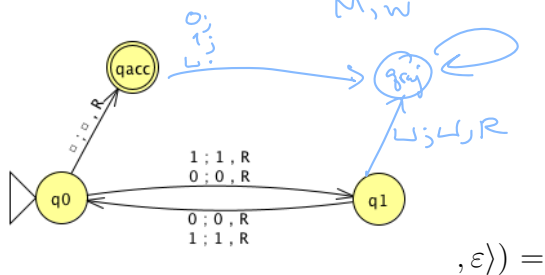
$const_{out} \notin HALT_{TM}$

where  $const_{out} = \langle \text{loop}, \epsilon \rangle$  and  $M'$  is a Turing machine that computes like  $M$  except, if the computation ever were to go to a reject state,  $M'$  loops instead.

Goal: ①  $x \in A_{TM}$  guarantees  $F(x) \in HALT_{TM}$

and  $x \notin A_{TM}$  guarantees  $F(x) \notin HALT_{TM}$

- ②  $x \neq \langle M, w \rangle$  for any  $M, w$   
 ③  $x = \langle M, w \rangle$   $M$  rejects  $w$   
 ④  $x = \langle M, w \rangle$   $M$  loops on  $w$ .



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

Claim (1):  $F$  is computable

using wednesday's work

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

Case (1) Assume  $x \in A_{TM}$ . WTS  $F(x) \in HALT_{TM}$ .

By assumption,  $x = \langle M, w \rangle$  and  $M$  accepts  $w$ .

By definition of  $F$ ,  $F(x) = \langle M', w \rangle$

where  $M'$  simulates  $M$  on all input except loops instead of rejecting.

Does  $M'$  halt on  $w$ ? i.e. is  $F(x) \in HALT_{TM}$ ?

Running  $M'$  on  $w$  looks like running  $M$  on  $w$  (except if  $M'$  were to reject  $w$ , which it doesn't by assumption that  $M$  accepts  $w$ ). So since  $M$  accepts  $w$ , so does  $M'$ , and hence  $M'$  halts on  $w$ .

## Week 8 at a glance

**Textbook reading:** Section 4.1, 4.2, 5.3

*For Monday:* An undecidable language, Sipser pages 207-209.

*For Wednesday:* Definition 5.20 and figure 5.21 (page 236)

*For Friday:* Example 5.24 (page 236)

*For Monday of Week 9:* Example 5.26 (page 237)

### Make sure you can:

- Classify the computational complexity of a set of strings by determining whether it is decidable or undecidable and recognizable or unrecognizable.
  - State, prove, and use theorems relating decidability, recognizability, and co-recognizability.
  - Prove that a language is decidable or recognizable by defining and analyzing a Turing machine with appropriate properties.
- Use diagonalization to prove that there are 'hard' languages relative to certain models of computation.
- Use mapping reduction to deduce the complexity of a language by comparing to the complexity of another.
  - Define computable functions, and use them to give mapping reductions between computational problems
  - Define and explain  $A_{TM}$  and  $HALT_{TM}$
  - Build and analyze mapping reductions between computational problems

### TODO:

Review quizzes based on class material each day.

Homework assignment 4 due this Thursday.

Test 2 next Friday.