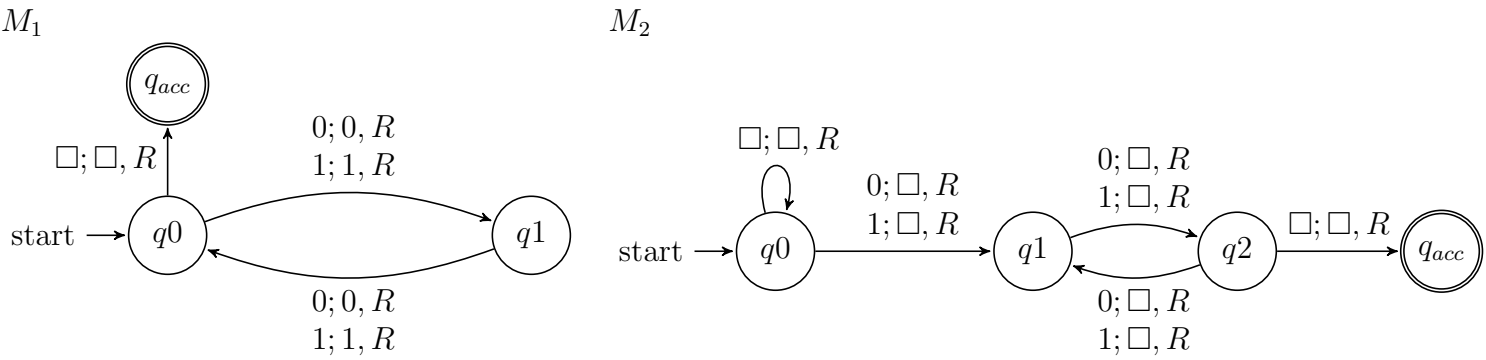


Week8 monday

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



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.

# Week8 wednesday

## 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  $\Sigma^*$ ,

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

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

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

*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.

## Week8 friday

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

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

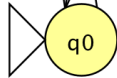
## Halting problem

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

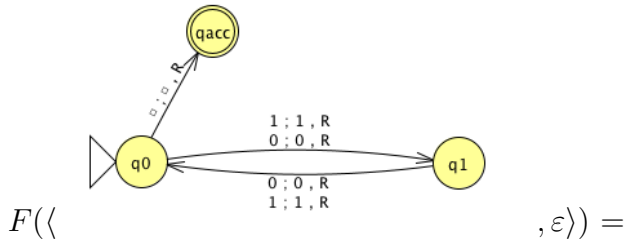
Define  $F : \Sigma^* \rightarrow \Sigma^*$  by

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

□ ; □ , R  
1 ; 1 , R  
0 ; 0 , R



where  $const_{out} = \langle \text{reject state}, \varepsilon \rangle$  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.



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  $F(x) \in HALT_{TM}$ .