Week 8 at a glance

Textbook reading: Chapter 4, Section 5.3

Before Monday, "An undecidable language", Sipser pages 207-209.

Before Wednesday, Definition 5.20 and figure 5.21 (page 236) of mapping reduction.

Before Friday, Example 5.24 (page 236).

For Week 9 Monday: Example 5.26 (page 237).

We will be learning and practicing to:

- Clearly and unambiguously communicate computational ideas using appropriate formalism. Translate across levels of abstraction.
  - Give examples of sets that are regular, context-free, decidable, or recognizable (and prove that they are).
    - **Define and explain the definitions of the computational problem A_{TM}**
    - **Define and explain the definitions of the computational problem $HALT_{TM}$**
- Know, select and apply appropriate computing knowledge and problem-solving techniques. Reason about computation and systems.
  - Use diagonalization to prove that there are 'hard' languages relative to certain models of computation.
    - **Trace the argument that proves A_{TM} is undecidable and explain why it works.**
  - 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**
    - **Build and analyze mapping reductions between computational problems**
    - **Deduce the decidability or undecidability of a computational problem given mapping reductions between it and other computational problems, or explain when this is not possible.**
  - Classify the computational complexity of a set of strings by determining whether it is regular, context-free, decidable, or recognizable.
    - **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 machines with appropriate properties.**

TODO:

Review Quiz 7 on PrairieLearn (http://us.prairielearn.com), due 2/26/2025

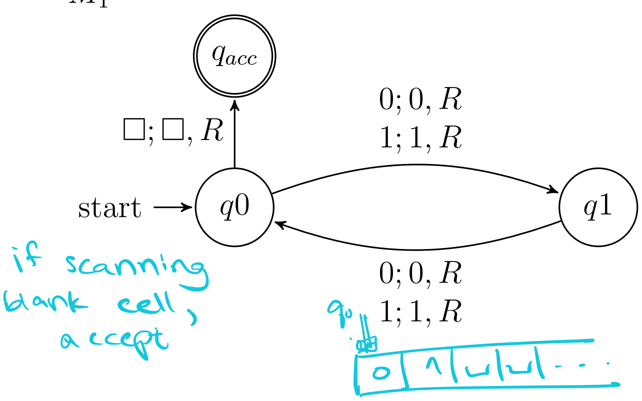
Homework 5 submitted via Gradescope (https://www.gradescope.com/), due 2/27/2025

Review Quiz 8 on PrairieLearn (http://us.prairielearn.com), due 3/5/2025

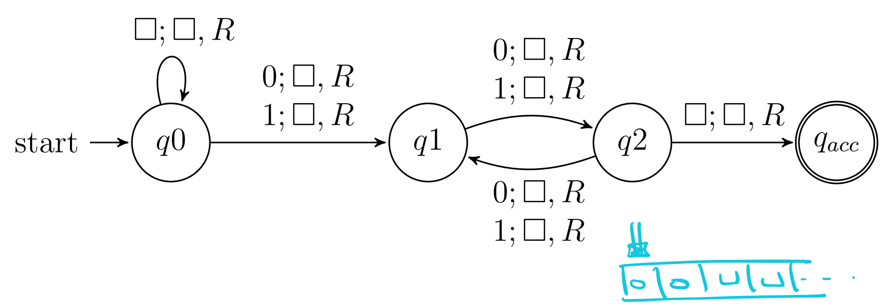
Monday: A_{TM} is recognizable but undecidable

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

M_1



M_2



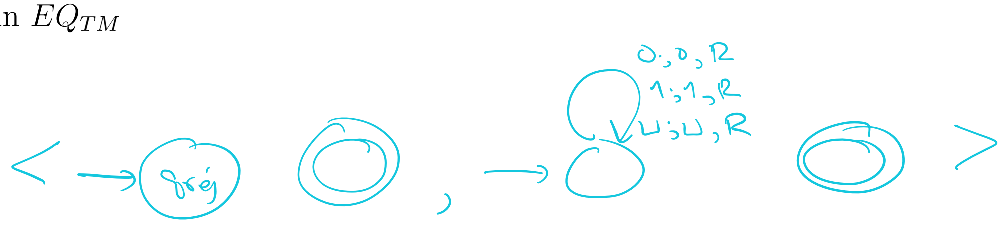
Example strings in A_{TM}

$\langle M_1, 01\rangle$ because $01 \in L(M_1)$

$\langle M_2, 00\rangle$ because $00 \in L(M_2)$

Example strings in E_{TM}

Example strings in EQ_{TM}



$\langle M_1, M_1\rangle$

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

high level description

0. Type check whether $x = \langle M, w\rangle$ where M is TM and w string. If not, reject.
1. Run M on w
2. If M accepts w, accept.
3. If M rejects w, reject. "

Proof of correctness: WTS $L(R_{ATM}) = A_{TM}$

Take arbitrary x

Case 1 $x \neq \langle M, w\rangle$ for any TM M string w

By def of A_{TM}, $x \notin A_{TM}$ so WTS R_{ATM} doesn't accept x. Tracing R_{ATM} on x, in type check x is rejected (by case assumption) so R_{ATM} doesn't accept x. ✓

Case 2 $x = \langle M, w\rangle$ for some TM M string w

Case 2a $w \in L(M)$

By def of A_{TM}, $x \in A_{TM}$ so WTS R_{ATM} accepts x. Tracing R_{ATM} on x, (by case assumption) x passes type check and in step 1 R_{ATM} runs M on w as a subroutine. By case assumption the subroutine halts and accepts, so in step 2 R_{ATM} accepts x ✓.

Case 2b $w \notin L(M)$

Case 2bi M rejects w

By def of A_{TM}, $x \notin A_{TM}$ so WTS R_{ATM} doesn't accept x. Tracing R_{ATM} on x, by case assumption x passes type check and in step 1 R_{ATM} runs M on w as a subroutine. By case assumption the subroutine halts and rejects so in step 3 R_{ATM} rejects x ✓

Case 2bii M loops on w

By def of A_{TM}, $x \notin A_{TM}$ so WTS R_{ATM} doesn't accept x. Tracing R_{ATM} on x, by case assumption x passes the type check and in step 1 R_{ATM} runs M on w as a subroutine. By case assumption the subroutine doesn't halt so R_{ATM} doesn't halt on x so doesn't accept x. ✓

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.

Define a **new** Turing machine using the high-level description: (work towards contradiction)

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

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

Is D a Turing machine? high level description

Is D a decider? step 1 : run M_{ATM} , which takes finitely many steps , because M_{ATM} is a decider
step 2 : conditional

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

Case 1 $\langle D, \langle D \rangle \rangle \in A_{TM}$

Because M_{ATM} decides A_{TM}, M_{ATM} accept $\langle D, \langle D \rangle \rangle$

Trace D on $\langle D \rangle$

Step 1: Run M_{ATM} on $\langle D, \langle D \rangle \rangle$
by case assumption, accept.

Step 2: D rejects $\langle D \rangle$.

so by definition of A_{TM}

$\langle D, \langle D \rangle \rangle \notin A_{TM}$

$\rightarrow\!\!\!\!\times$

Case 2 $\langle D, \langle D \rangle \rangle \notin A_{TM}$

Because M_{ATM} decides A_{TM}, M_{ATM} rejects $\langle D, \langle D \rangle \rangle$.

Trace D on $\langle D \rangle$

Step 1: Run M_{ATM} on $\langle D, \langle D \rangle \rangle$
by case assumption, reject.

Step 2: D accepts $\langle D \rangle$.

so by definition of A_{TM}

$\langle D, \langle D \rangle \rangle \in A_{TM}$

$\rightarrow\!\!\!\!\times$

The existence of M_{ATM} lead to a contradiction so there can be no decider that decides A_{TM}.

In other words, A_{TM} is undecidable $\boxtimes$

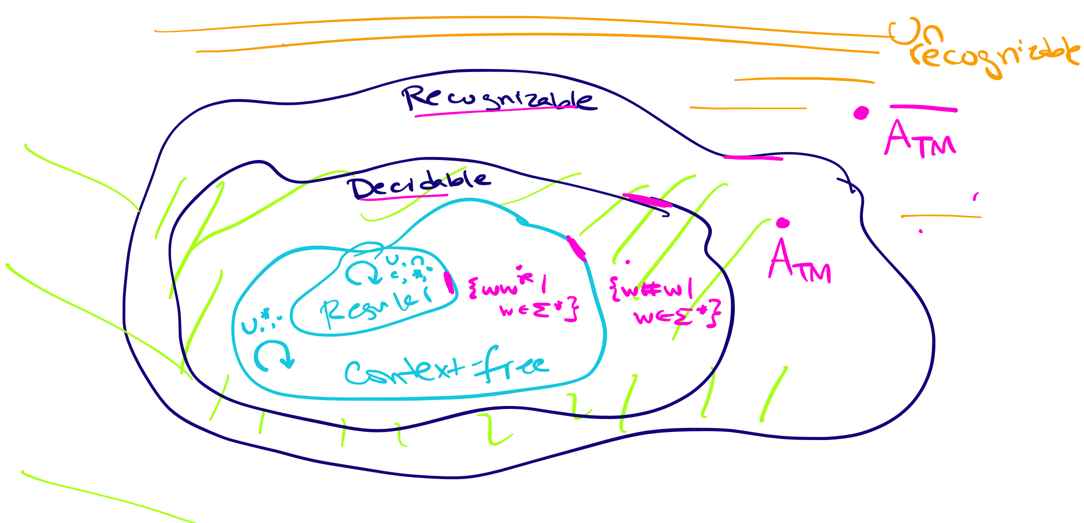
Summarizing:

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

$$A_{TM} = \left\{ \langle M, w \rangle \;\middle|\; \begin{array}{l} M \text{ is TM}, w \text{ is string} \\ M \text{ accepts } w \end{array} \right\}$$

Recall definition: A language L over an alphabet Σ 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.

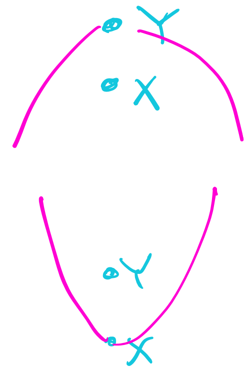
Wednesday: Computable functions and mapping reduction

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, decidable
...then X must be easy too.
decidable

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



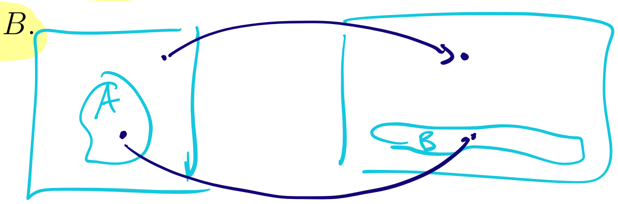
"Problem X is no harder than problem Y" means "Can answer questions about membership in X by converting them to questions about membership in Y".

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

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

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

and we say f witnesses this mapping reduction



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 $f : \Sigma^* \to \Sigma^*$ 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)

$$f_1 : \Sigma^* \to \Sigma^* \qquad f_1(x) = x0$$

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."

" On input w
1. Append 0 to w
2. Output result "

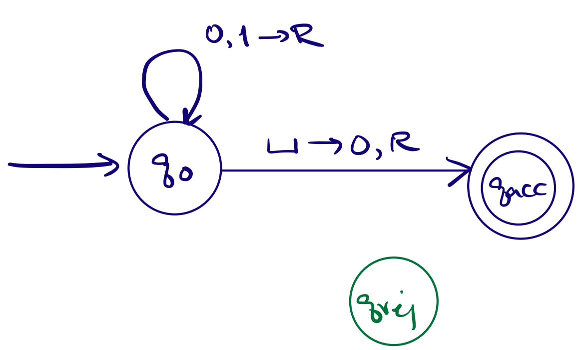
" On input w
1 Output w0 "

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 $(\{q0, qacc, qrej\}, \{0,1\}, \{0,1,\sqcup\}, \delta, q0, qacc, qrej)$ where δ is specified by the state diagram:



(still using convention that all "missing" arrows go to qrej)

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

$$f_2 : \Sigma^* \to \Sigma^* \qquad f_2(x) = xx$$

"On input x

1. Output 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.

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

where $q_{trap} \notin Q$ and

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

$F =$ "On input x

1. If $x \neq \langle M \rangle$ for any TM M, then output ε.
2. Else, $x = \langle M \rangle$ for some TM, build

   $M_{new} =$ "On input w,
   1. Simulate M on w
   2. If M accepts, accept.
   3. If M rejects, go to 4.
   4. Go to 4."
3. Output $\langle M_{new} \rangle$"

Note $L(M_{new}) = L(M)$

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

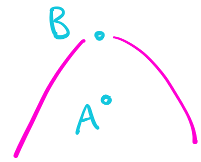
$$x \in A \qquad \text{if and only if} \qquad f(x) \in B.$$

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

Making intutition precise ...

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

Hyp Conc



Consider arbitrary languages A and B and assume ① $A \leq_m B$ and ② B decidable

By ① there is a witnessing computable function to the mapping reduction, so there's a TM F where, for each string x, $x \in A$ iff the output of F on input x is in B.

By ② there is a TM that is a decider and decides B, call it M_B.

We WTS that A is decidable

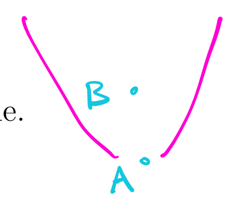
Define M_A = " On input x

1. Calculate $F(x)$ Takes finitely many steps
2. Run M_B on $F(x)$ Takes finitely many steps
3. If accepts, accept
4. If rejects, reject "

Claim ⊛ $\forall x \, (x \in A \to M_A \text{ accepts } x)$

Claim ⊛⊛ $\forall x \, (x \notin A \to M_A \text{ rejects } x)$

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



Proof by contradiction

Suppose there are sets A, B with A undecidable and B decidable and $A \leq_m B$.

By above Theorem, $A \leq_m B$ and B decidable guarantee that A is decidable so $\rightarrow\leftarrow$ with assumption that A is undecidable ⊠

Friday: The Halting problem

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

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

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.

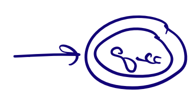
Example: $A_{TM} \leq_m A_{TM}$ $\qquad$ $A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } w \text{ is a string and } w \in L(M)\}$

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

Consider $f: \Sigma^* \to \Sigma^*$ defined by $f(x) = x$ (identity function)

- computable? Yes as witnessed by TM

  with state diagram



  or high level description

  "On input x

  1. Output x."

- translation property? For arbitrary x wts $x \in A_{TM}$ iff $f(x) \in A_{TM}$. Since $f(x) = x$, this is guaranteed by tautology $p \Leftrightarrow p$ (always true)

We didn't use any special properties of A_{TM} (recognizable, undecidable)

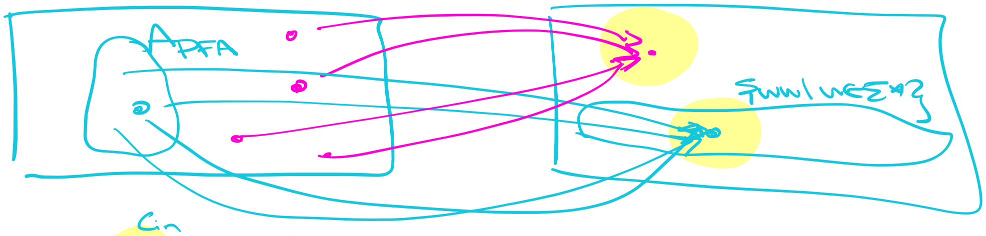
Corollary: For any language L, $L \leq_m L$, as witnessed by the identity function.

A_{DFA} decidable, recognizable

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

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

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



Define

$$f(x) = \begin{cases} \overset{C_{in}}{11} & \text{if } x \in A_{DFA} \\ \underset{C_{out}}{1} & \text{if } x \notin A_{DFA} \end{cases} \quad \text{for } x \in \Sigma^*$$

other valid choices could be
For C_{in} ε or 00 or 0101
For C_{out} 0 or 01 etc.

Have a decider M_1 for A_{DFA} which, for $x \in \Sigma^*$ accept x if $x \in A_{DFA}$ and reject x if $x \notin A_{DFA}$

Build the TM F = " On input x
1. Run M_1 on x
2. If M_1 accepts x, output 11
3. Otherwise (if M_1 rejects x), output 1 "

Corollary: For any language decidable language X and any set Y with at least one string (C_{in}) in Y and at least one string not in Y (C_{out}), $X \leq_m Y$, as witnessed by the function defined as

$$f(x) = \begin{cases} C_{in} & \text{if } x \in X \\ C_{out} & \text{if } x \notin X \end{cases}$$

$$X \leq_m Y$$

" decidable is as easy as it gets " (relative to mapping reduction)

Notice X decidable doesn't give any bounds on difficulty of Y.

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.

Fun fact: Also $HALT_{TM} \leq_m A_{TM}$

We will (first) prove that $A_{TM} \leq_m HALT_{TM}$

① Identity function

$f : \Sigma^* \to \Sigma^*$

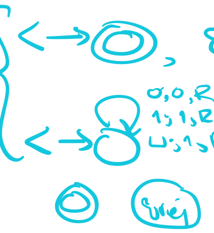
$f(x) = x$

computable ✓

what about translation property?

② Flags

$$f(x) = \begin{cases} \langle \cdot, \varepsilon \rangle & \text{if } x \in A_{TM} \\ \langle \cdot, \varepsilon \rangle & \text{if } x \notin A_{TM} \end{cases}$$



has translation property but is not computable

uh oh...

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

We need for arbitrary $x \in \Sigma^*$, $x \in A_{TM}$ iff $x \in HALT_{TM}$

Case ① $x \in A_{TM}$: since $x \in A_{TM}$ means $x = \langle M, w\rangle$ for some TM M and string w with M accepting w, know M halts on w so $x = \langle M, w\rangle \in HALT_{TM}$.

Case ② $x \notin A_{TM}$

Case (2a) $x \neq \langle M, w\rangle$ for any TM M or string w: so $x \notin HALT_{TM}$

Case (2b) $x = \langle M, w\rangle$ for some TM M and string w but M doesn't accept w

Case (2bi) M rejects w : $x \in HALT_{TM}$ ***

Case (2bii) M doesn't halt on w : $x \notin HALT_{TM}$

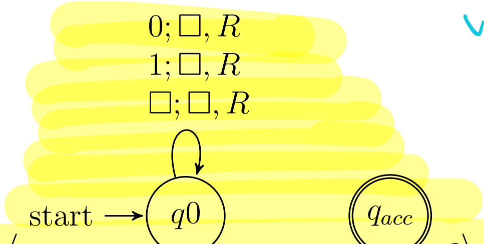
Define $F : \Sigma^* \to \Sigma^*$ by

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

"almost x"

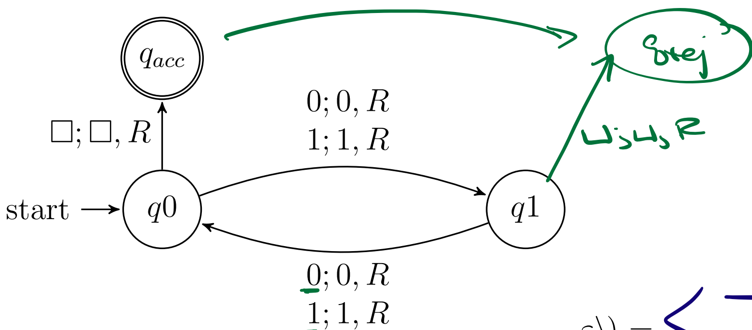
We intentionally choose $const_{out} \notin HALT_{TM}$

Build M' without running M

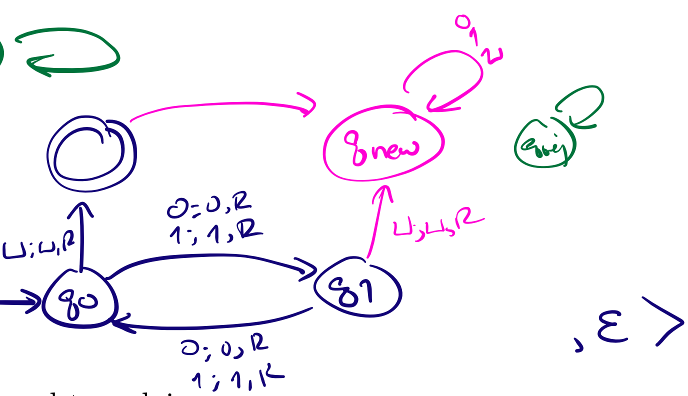


where $const_{out} = \langle$ [machine above] $, \varepsilon \rangle$ and M'_x is a Turing machine that computes like M except, if the computation of M ever were to go to a reject state, M'_x loops instead.

$F(\langle$



$, \varepsilon \rangle) = \langle$



$, \varepsilon \rangle$

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

Claim (1): F is computable

See page 9.

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

Proof: Let x be an arbitrary string.

Case ① $x \in A_{TM}$. Then $x = \langle M, w \rangle$ for some TM M and string w and M accepts w. By definition of F, $F(x) = \langle M'_x, w \rangle$

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

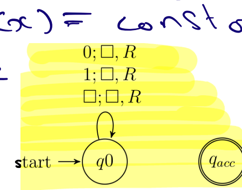
To check if $F(x) \in HALT_{TM}$, we trace the computation of M'_x on w. This computation (by definition of M'_x) starts by running M on w. Since (by case assumption) M accepts w, M'_x also accepts w.

Case ② $x \notin A_{TM}$.

Case (2a) $x \neq \langle M, w \rangle$ for any TM M and string w. Then (by definition of F), $F(x) = const_{out}$

WTS $F(x) \notin HALT_{TM}$

and is not in $HALT_{TM}$ because



doesn't halt on (any string and in particular, not on) ε.

WTS $F(x) \notin HALT_{TM}$

Case 2b $x = \langle M, w \rangle$ for some TM M and string w but M doesn't accept w

Case 2bi M rejects w. By definition of F, $F(x) = \langle M'_x, w \rangle$. To check whether $F(x) \in HALT_{TM}$ we trace the computation of M'_x on w. This computation (by definition of M'_x) starts by running M on w. Since (by case assumption) M rejects w, M'_x loops on w so $\langle M'_x, w \rangle \notin HALT_{TM}$.

Case 2bii M loops on w. By definition of F, $F(x) = \langle M'_x, w \rangle$. To check if $F(x) \in HALT_{TM}$ we trace the computation of M'_x on w. This computation (by definition of M'_x) starts by running M on w. Since (by case assumption) M loops on w, M'_x also loops on w so $\langle M'_x, w \rangle \notin HALT_{TM}$.