

CSE 105

THEORY OF COMPUTATION

Spring 2018

Discussion today: Ch 4 + 5
Group HW 6 due Saturday
Review Quiz "due" Sunday
* Group HW 7 due Tuesday *
Optional HW 8 - discussion wk 10

<http://cseweb.ucsd.edu/classes/sp18/cse105-ab/>

Review session Wednesday evening.

Today's learning goals

Sipser Ch 5.1, 5.3

- Define and explain core examples of computational problems, include A^{**} , E^{**} , EQ^{**} , $HALT_{TM}$ (for $**$ either DFA or TM)
- Explain what it means for one problem to reduce to another
- Define computable functions, and use them to give mapping reductions between computational problems.

| Decidable | Undecidable but recognizable | Undecidable and unrecognizable |
|-------------------|------------------------------|--------------------------------|
| A_{DFA} | A_{TM} | A_{TM}^c |
| E_{DFA} | | |
| EQ_{DFA} | | |

Give algorithm!

Diagonalization

Idea

Sipser pp. 215-216

If problem X is no harder than problem Y
...and if Y is **decidable**
...then X must also be **decidable**

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

“Problem X is no harder than problem Y” means

“Can convert questions about membership in X to questions about membership in Y”

Mapping reduction

Sipser p. 235

Problem A is ^{mapping reduces} mapping reducible to problem B means there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that for all strings x in Σ^*

x is in A iff $f(x)$ is in B

Computable function?

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

Computable functions (aka maps)

Which of the following functions are computable?

- A. The string x maps to the string xx .
- B. The string $\langle M \rangle$ (where M is a TM) maps to $\langle M' \rangle$ where M' is the Turing machine that acts like M does, except that if M tries to reject, M' goes into a loop; strings that are not the codes of TMs map to ϵ .
- C. The string x maps to y , where x is the binary representation of the number n and y is the binary representation of the number 2^n .
- D. All of the above.
- E. None of the above.

The halting problem!

$\text{HALT}_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \}$

accepts or rejects

$A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } \underline{w \text{ is in } L(M)} \}$

M accepts w

$A_{\text{TM}} \subsetneq \text{HALT}_{\text{TM}}$

How is HALT_{TM} related to A_{TM} ?

- ~~A. They're the same set.~~
- ~~B. HALT_{TM} is a subset of A_{TM}~~
- C. A_{TM} is a subset of HALT_{TM}
- ~~D. They have the same type of elements but no other relation.~~
- E. I don't know.

The halting problem!

$\text{HALT}_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \}$

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

But subset inclusion doesn't determine difficulty!

What about mapping reduction?

The halting problem!

$\text{HALT}_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \}$

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

Goal: build function $f: \Sigma^* \rightarrow \Sigma^*$ such that for every string x ,

x is in A_{TM}

iff

$f(x)$ is in HALT_{TM}

n.d. A_{TM} reduces to HALT_{TM}

Reducing A_{TM} to $HALT_{TM}$

Sipser Example 5.24

Desired function by cases:

$$f(x) =$$

$x \in A_{TM}$

• If $x = \langle M, w \rangle$ and w is in $L(M)$: map to $\langle M', w' \rangle$ in $HALT_{TM}$

• If $x = \langle M, w \rangle$ and w is not in $L(M)$: map to $\langle M', w' \rangle$ not in $HALT_{TM}$

• If $x \neq \langle M, w \rangle$: map to some string not in $HALT_{TM}$

$x \notin A_{TM}$

$f(x) \notin HALT_{TM}$

Reducing A_{TM} to $HALT_{TM}$ Sipser Example 5.24

Desired function by cases:

- If $x = \langle M, w \rangle$ and w is in $L(M)$: map to $\langle M', w' \rangle$ in $HALT_{TM}$
- If $x = \langle M, w \rangle$ and w is not in $L(M)$: map to $\langle M', w' \rangle$ not in $HALT_{TM}$
- If $x \neq \langle M, w \rangle$: map to some string not in $HALT_{TM}$
Pick some specific string constant not in $HALT_{TM}$

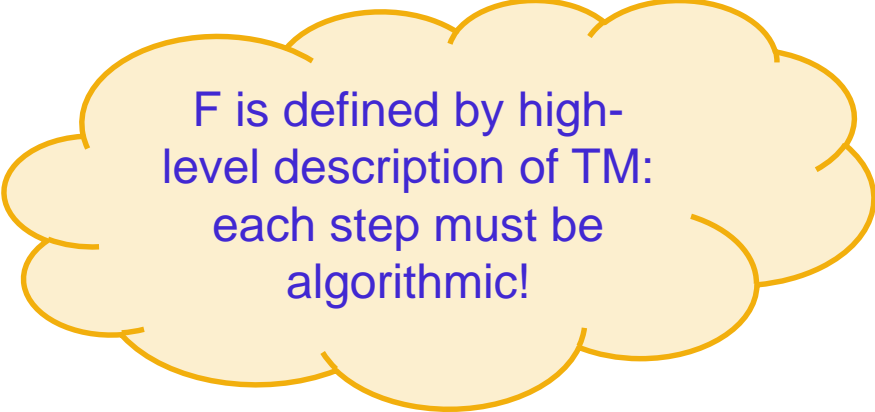
Reducing A_{TM} to $HALT_{TM}$

Sipser Example 5.24

Define **computable** function:

F = "On input x:

1. **Type-check** whether $x = \langle M, w \rangle$ for some TM M , and string w . If not, output **const_{out}**.
2. ...
3. ...
4. "



F is defined by high-level description of TM:
each step must be algorithmic!

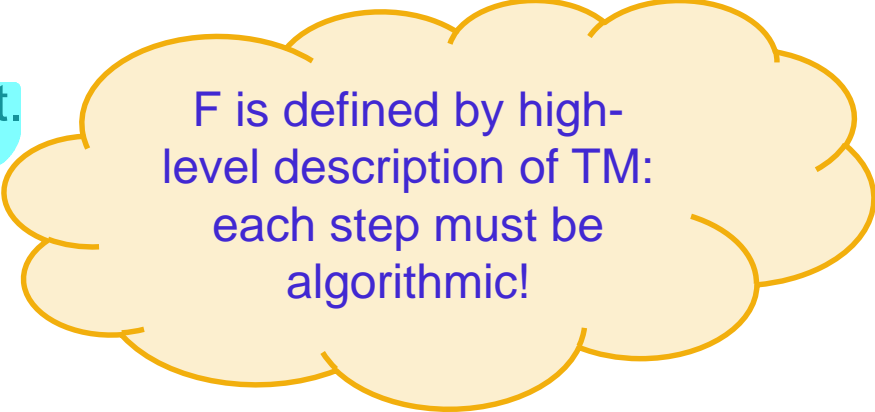
Reducing A_{TM} to $HALT_{TM}$

Sipser Example 5.24

Define **computable** function:

F = "On input x:

1. Type-check whether $x = \langle M, w \rangle$ for some TM M, and string w. If not, **output** $const_{out}$.
2. ~~Simulate M on w.~~ *uh oh*
3. ~~If accepts, accept. If rejects, reject.~~
4. "



F is defined by high-level description of TM:
each step must be algorithmic!

Reducing A_{TM} to $HALT_{TM}$

Sipser Example 5.24

Define **computable** function:

Does M accept w ?

$F =$ "On input x :

1. Type-check whether $x = \langle M, w \rangle$ for some TM M , and string w . If not, output $const_{out}$.

2. **Construct** the following machine M'

$M' =$ "On input x :

1. Run M on x . *Similar to M*
2. If M accepts, accept.
3. If M rejects, enter a loop. *diff.*

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

F is defined by high-level description of TM: each step must be algorithmic!

$L(M) = L(M')$ but

M may accept/reject/loop on input
 M' may only accept/loop on input

Reducing A_{TM} to $HALT_{TM}$

Sipser Example 5.24

Check how function behaves by cases:

- If $x = \langle M, w \rangle$ and w is in $L(M)$: map to $\langle \underline{M}', w' \rangle$ in $HALT_{TM}$?
so M' simulates M on w and since M accepts w , M' will too
- If $x = \langle M, w \rangle$ and w is not in $L(M)$: map to $\langle \underline{M}', w' \rangle$ not in $HALT_{TM}$?
so M' loops and if M loops/rejects on w , M' loops on w
so $F(x) = \langle \underline{M}', w' \rangle \notin HALT_{TM}$.
- If $x \neq \langle M, w \rangle$: map to some string not in $HALT_{TM}$?

i.e. $x \in A_{TM} \iff F(x) \in HALT_{TM}$,

Other direction?

Goal: build function that $f: \Sigma^* \rightarrow \Sigma^*$ such that for every string x ,

x is in HALT_{TM}

iff

$f(x)$ is in A_{TM}

What function should be used for $f(x)$ in the reduction?

- ~~A.~~ Use the function F from previous reduction
- ~~B.~~ Use the inverse of the function F from previous reduction
- C. Use a different computable function
- ~~D.~~ Impossible to find a computable function that works!

HALT_{TM} mapping reduces to A_{TM}?

Need $f: \Sigma^* \rightarrow \Sigma^*$ s.t.

$$x \in \text{HALT}_{\text{TM}} \iff f(x) \in A_{\text{TM}}$$

if computable

Given $c \notin A_{\text{TM}}$.

$f =$ "On input x ."

1. Typecheck to see if $x = \langle M, w \rangle$, M TM, w string
if not, output c . Otherwise: $x = \langle M, w \rangle$.

2. Build $M' =$ "On input y ."

1. Run M on y

2. If M accepts, accept.

3. Output $\langle M', w \rangle$ " if M rejects, accept "

Check: $x \in \text{HALT}_{\text{TM}} \iff f(x) \in A_{\text{TM}}$

Next time

Pre-class reading Example 5.24, Theorems 5.22