CS3102 Theory of Computation

www.cs.virginia.edu/~njb2b/cstheory/s2020

Warm up:

Recall:

- $ACCEPTS(w, x) = 1$ if w (as a TM description, call it $\mathcal{M}(w)$) accepts x, 0 otherwise

Why would $ACCEPTS$ be a useful function to implement?

Last Time

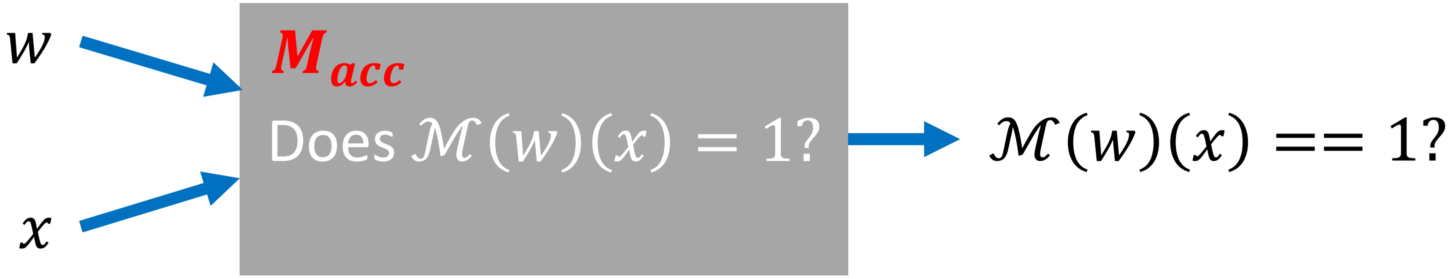
- An uncomputable problem
  - $ACCEPTS(w, x) = 1$ if w (as a TM description, call it $\mathcal{M}(w)$) accepts x, 0 otherwise
  - First attempt to solve:
    - Try running $\mathcal{M}(w)$, see what happens
  - Challenge: $\mathcal{M}(w)$ could not accept for two reasons
    - $\mathcal{M}(w)$ halts and returns 0
    - $\mathcal{M}(w)$ runs forever (how long do we wait to see?)

Proving $ACCEPTS$ is uncomputable

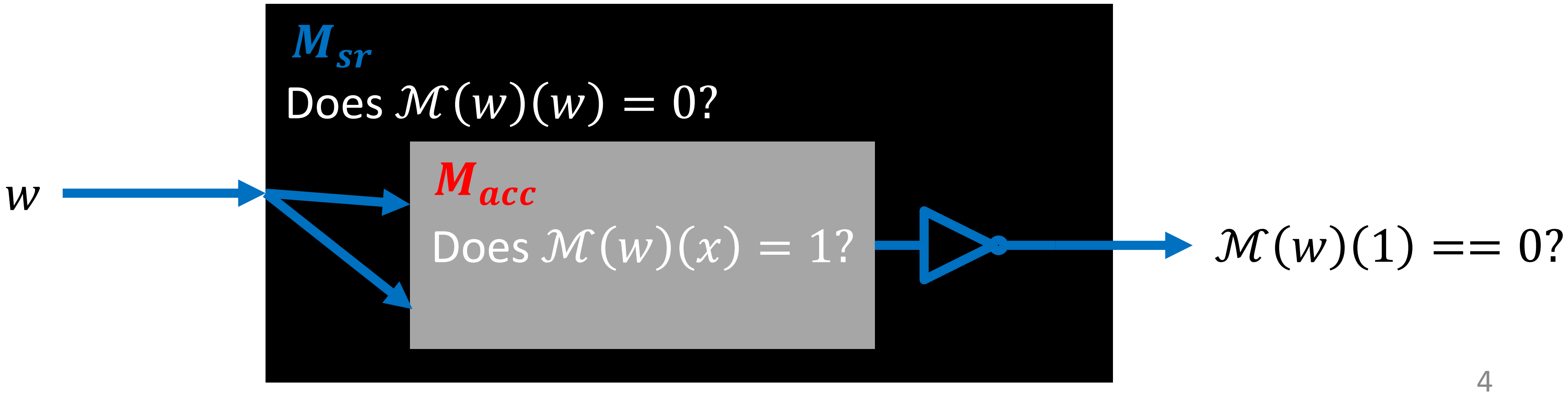
- Proof by contradiction:
  - Assume (toward contradiction) that there is an always-halting Turing machine to compute $ACCEPTS$
  - Show that we could use that Turing machine to build an impossible Turing machine
- What's the impossible machine?
  - $SelfReject$= The set of all Turing machine descriptions that don't accept themselves
  - $x \in SelfReject$ if and only if $\big(\mathcal{M}(x)\big)(x) = 0$

Using $ACCEPTS$ to build $SelfReject$

Assume we have M_{acc}
which computes $ACCEPTS$:



We could then build M_{sr}
which computes $SelfReject$
like this:



Can M_{SR} exist?

- Let w_{SR} be the description of M_{SR}
  - $\mathcal{M}(w_{SR}) = M_{SR}$
- What should be $M_{SR}(w_{SR})$?
  - If $M_{SR}(w_{SR}) = 1$:
    - Since M_{SR} computes $SelfReject$ we conclude that w_{SR} is rejected by whatever machine it describes.
    - Since w_{SR} describes M_{SR}, it must have been that $M_{SR}(w_{SR}) = 0$
  - If $M_{SR}(w_{SR}) = 0$:
    - Since M_{SR} computes $SelfReject$ we conclude that w_{SR} is accepted by whatever machine it describes.
    - Since w_{sr} describes M_{SR}, it must have been that $M_{SR}(w_{SR}) = 1$
  - There's no answer that makes sense!
- Conclusion: M_{SR} can't be an always-halting Turing machine, so M_{acc} can't exist

How to show things aren't computable

1. Ask "can I have an always-halting Turing machine M_p for language/function/problem P?"

2. Show that, if M_p exists, it can be used to make an impossible machine M_{imp}

How do we know a machine is impossible?

Option 1: It contradicts itself (e.g. M_{SR})
Option 2: Someone has done this before (e.g. M_{acc})

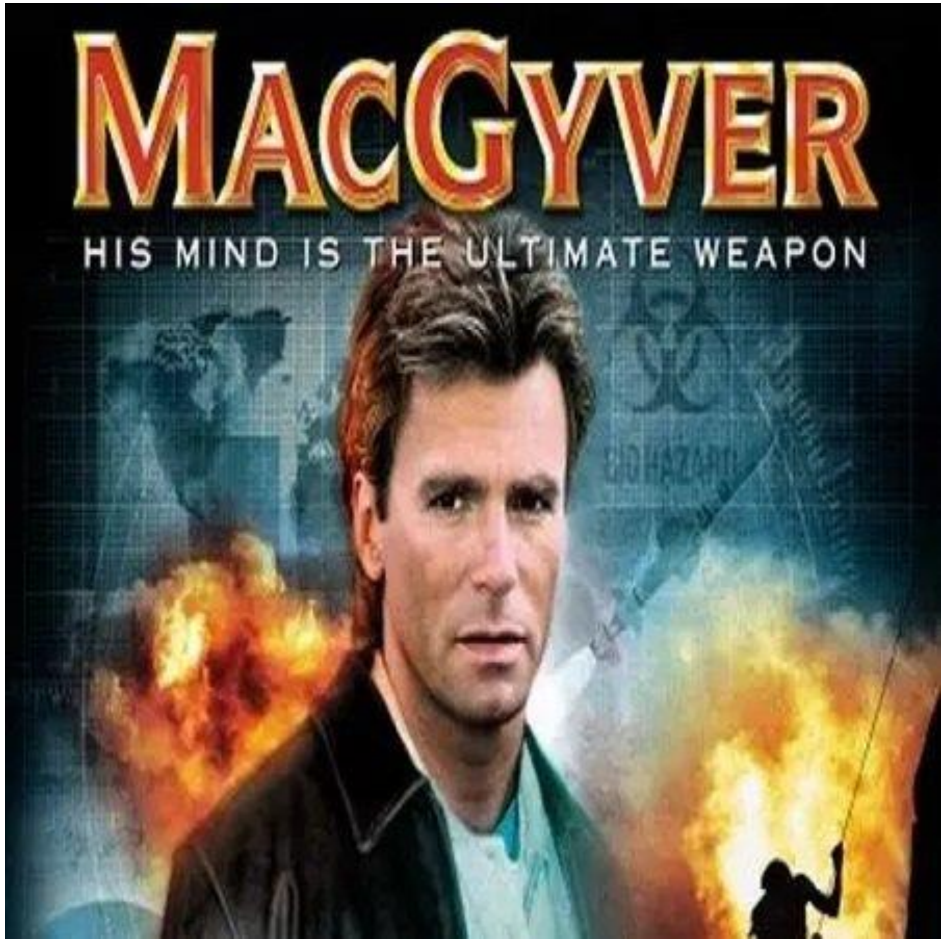
Proving Other Problems are Uncomputable

- Reduction
  - Convert some problem into a known uncomputable one (using only computable steps)

Proof by Reduction

Shows how two different problems relate to each other

MOVIE TIME!



Reduction Proofs

Opening a door

A reduces to B

Lighting a fire

Alcohol, wood, matches

Y can be used to make X

Keg cannon battering ram

that can solve **B**

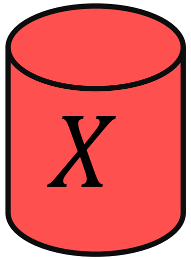
that can solve **A**

$\boldsymbol{A}$ is not a harder problem than $\boldsymbol{B}$

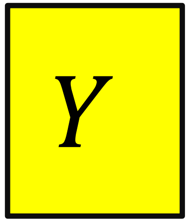
$$\boldsymbol{A \leq B}$$

The name “reduces” is confusing: it is in the *opposite* direction of the making

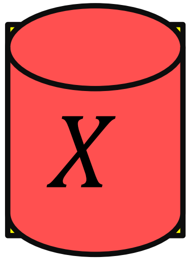
Proof of Impossibility by Reduction



1. X isn't possible
(e.g., X = some way to open the door)



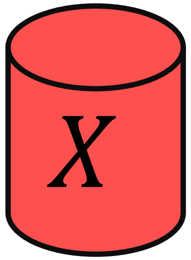
2. Assume Y is possible
(Y = some way to light a fire)



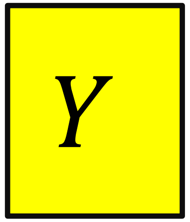
3. Show how to use Y to perform X.

4. X isn't possible, but Y could be used to perform X
conclusion: Y must not be possible either

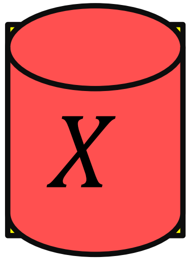
Proof of Impossibility by Reduction



1. Take X that does not exist.
e.g., X = Some TM that computes $ACCEPTS$



2. Assume Y exists.
Y = Some TM that computes B

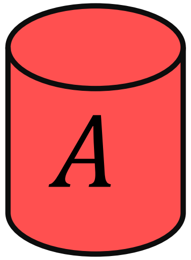


3. Show how to use Y to perform X.

4. X doesn't exist, but Y could be used to make X
    - conclusion: Y must not exist either, so B is impossible

MacGyver’s Reduction

Problem **know** is impossible



Opening a door

Solution for $\boldsymbol{A}$

Keg cannon battering ram

Aim duct at door, insert keg

Put fire under the Keg

Reduction

Problem we **think** is impossible

B

Lighting a fire

If

Solution for $\boldsymbol{B}$

Alcohol, wood, matches

Converse?

A reduces to B

Y

that can solve B
(Lighting a fire)

can be used to make

X

that can solve A
(opening a door)

A is not harder to solve than B

$$A \le B$$

Does this mean B is equally as hard as A? $A = B$

No!
Solving Y is only one way to solve X
There may be an easier way

Common Reduction Traps

- Be careful: the direction matters a great deal
  - Using a solver for B to solve A shows A is not harder than B

$\underline{\ \ A\ \ }$ Reduces to $\underline{\ \ B\ \ }$

- The transformation must use only things you ***can do***:
  - Otherwise it may be that B exists, but some other step doesn't!
  - Example:
    - A witch/wizard could open the door by waving a wand and casting a magic spell
    - MacGyver can't do magic, and is in a room that cannot be opened
    - Can we conclude that MacGyver can't wave a wand?

What “Can Do” Means

- Tools used in a reduction are limited by what you are proving
- Undecidability:
  - You are proving something about all TMs:
  - The transformation “can do” things a terminating TM “can do”

Spoiler alert!

- Complexity:
  - You are proving something about time required:
  - The time it takes to do the transformation is limited

Example

- $HALT(w,x) = \begin{cases} 1 \text{ if } \mathcal{M}(w)(x) \text{ halts} \\ 0 \text{ if } \mathcal{M}(w)(x) \text{ runs forever} \end{cases}$
  - Does the machine halt on this input?
- To show $HALT$ is uncomputable:
  - Show how to use a TM for $HALT$ to solve an uncomputable problem
    - Show $HALT \geq ACCEPTS$
    - Show $ACCEPTS$ reduces to $HALT$

$HALT$ Reduction

Problem **know** is impossible

A

$ACCEPTS$

Does $\mathcal{M}(w)$ accept x?

M_{acc} computes $ACCEPT$

Problem we **think** is impossible

B

$HALT$

Does $\mathcal{M}(w)$ halt on input x?

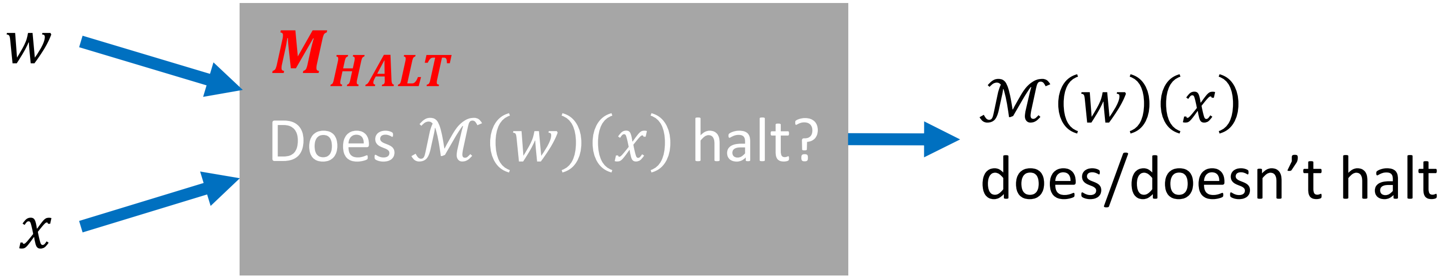
Assume

M_{halt} computes $HALT$

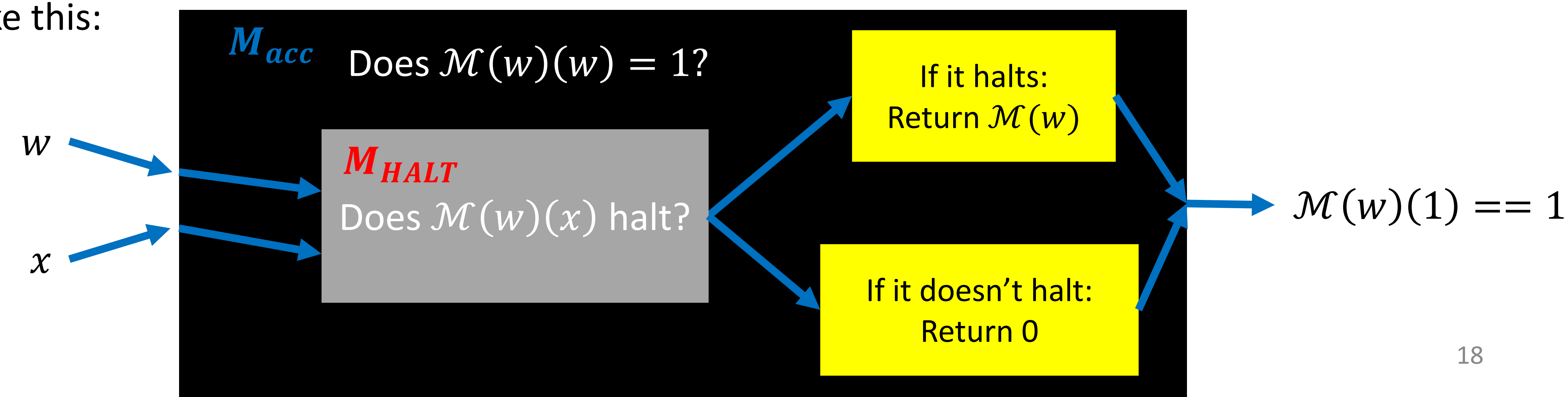
Reduction

Using $HALT$ to build $ACCEPTS$

Assume we have M_{HALT} which computes $HALT$:



We could then build M_{acc} which computes $ACCEPTS$ like this:



$HALT$ Reduction

Problem **know** is impossible

A

$ACCEPTS$

Does $\mathcal{M}(w)$ accept x?

Problem we **think** is impossible

$HALT$

B

Does $\mathcal{M}(w)$ halt on input x?

Give w, x as inputs to $HALT$

Assume

If $\mathcal{M}(w)(x)$ halts, return $\mathcal{M}(w)(x)$

Else return 0

M_{acc} computes $ACCEPT$

M_{halt} computes $HALT$

Reduction

Conclusion

- $ACCEPTS$ is not computable
- If $HALT$ was computable, an implementation could be used to compute $ACCEPTS$
- So it must be that $HALT$ is not computable

Example: $FINITE$

- $FINITE(w) = \begin{cases} 1 \text{ if } L\big(\mathcal{M}(w)\big) \text{is finite} \\ 0 \text{ if } L\big(\mathcal{M}(w)\big) \text{is infinite} \end{cases}$
- To show $FINITE$ is uncomputable
  - Show how to use a TM for $FINITE$ to solve $HALT$
    - $FINITE \geq HALT$
    - $HALT$ reduces to $FINITE$

$FINITE$ Reduction

Problem **know** is impossible

A

$HALT$

Does $\mathcal{M}(w)$ halt on input x?

M_{HALT} computes $HALT$

Reduction

Problem we **think** is impossible

$FINITE$

B

Is $\mathrm{L}\big(\mathcal{M}(w)\big)$ finite?

Assume

M_{finite} computes $FINITE$

Using $HALT$ to build $ACCEPTS$

Assume we have M_{FINITE} which computes $FINITE$:

w

M_{FINITE}

Is $\mathrm{L}(\mathcal{M}(w))$ finite?

$L(\mathcal{M}(w))$ is/isn't finite

We could then build M_{HALT} which computes $HALT$ like this:

M_{HALT}

Does $\mathcal{M}(w)(w) = 1$?

w

x

Build this machine:

M_{wx}:

1) run $\mathcal{M}(w)(x)$
2) return 1

M_{FINITE}

Is $\mathrm{L}(M_{wx})$ finite?

$\mathcal{M}(w)(1) == 1$

What's the Language of M_{wx}?

- If $\mathcal{M}(w)(x)$ halts:
  - M_{wx} always returns 1
  - $L(M_{wx}) = \Sigma^*$
  - $L(M_{wx})$ is infinite
- If $\mathcal{M}(w)(x)$ doesn't halt:
  - M_{wx} gets "stuck" in step 1 and never returns
  - $L(M_{wx}) = \emptyset$
  - $|L(M_{wx})| = 0$

Build this machine:
M_{wx}:
1) run $\mathcal{M}(w)(x)$
2) return 1

$FINITE$ Reduction

Problem **know** is impossible

A

$HALT$

Does $\mathcal{M}(w)$ halt on input x?

M_{HALT} computes $HALT$

Build (but don't run) M_{wx}

Give M_{wx} as input to M_{finite}

Reduction

Problem we **think** is impossible

$FINITE$

B

Is $\mathrm{L}(\mathcal{M}(w))$ finite?

Assume

M_{finite} computes $FINITE$