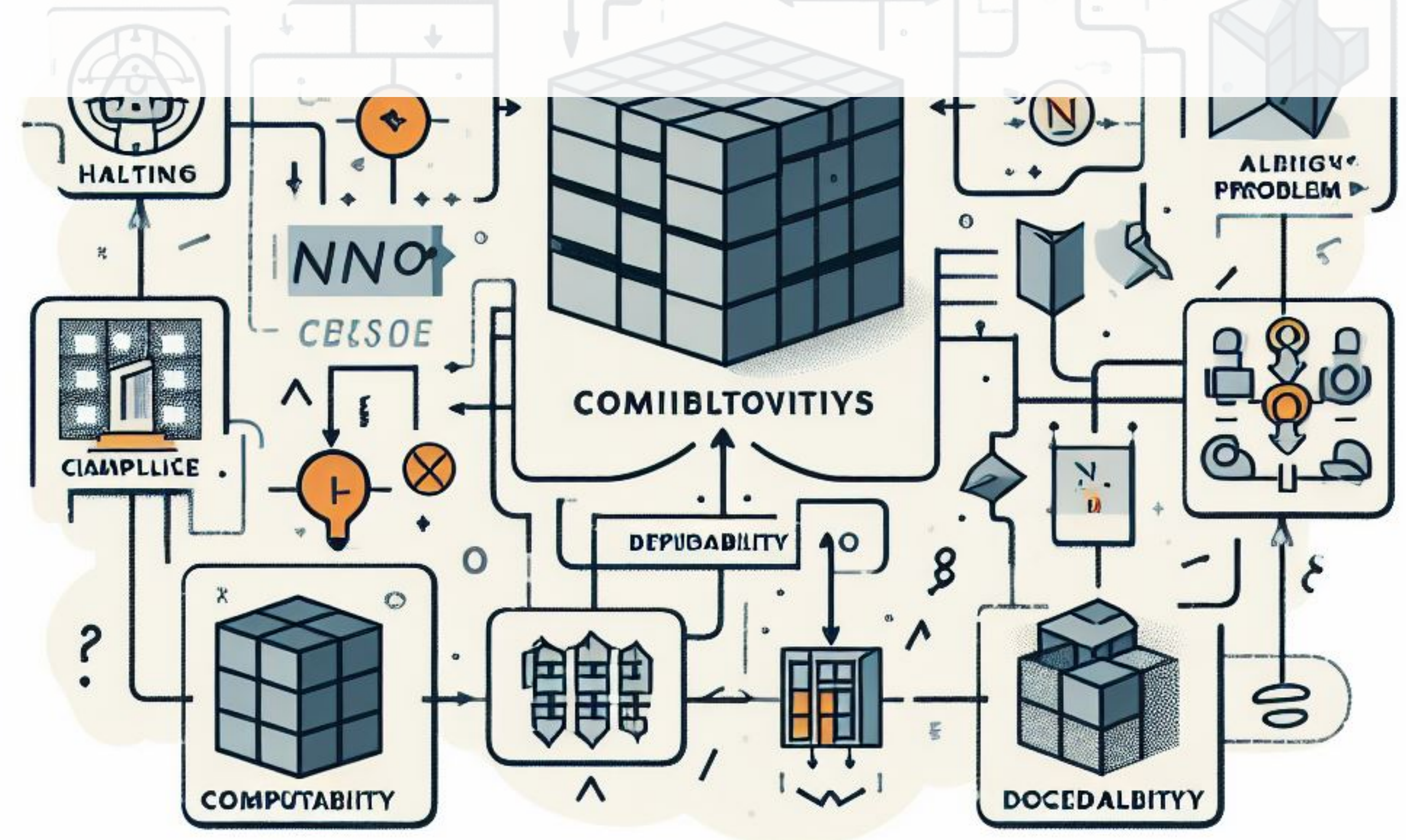
CMPU 240 · Theory of Computation

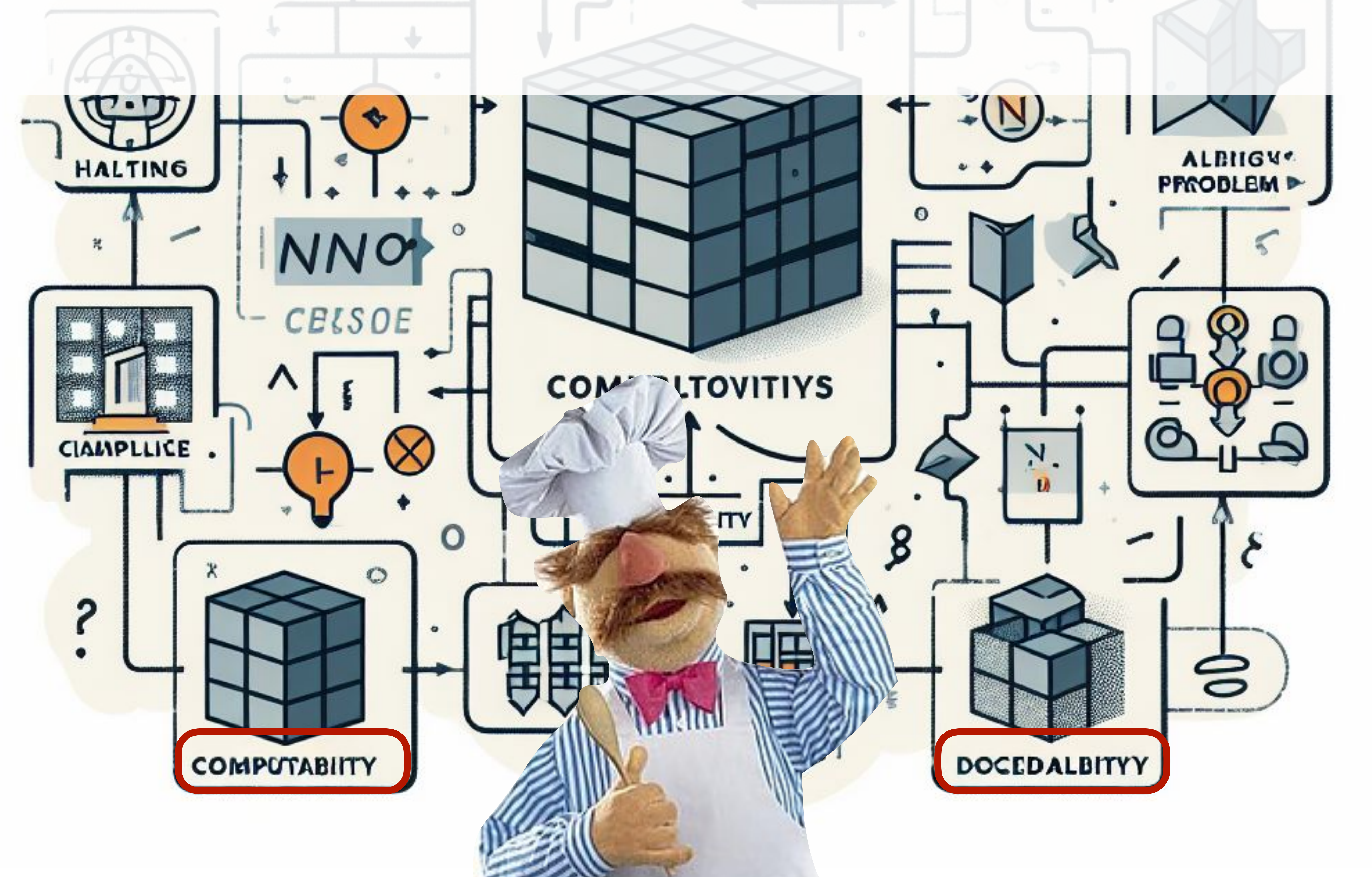
Reductions and Computability

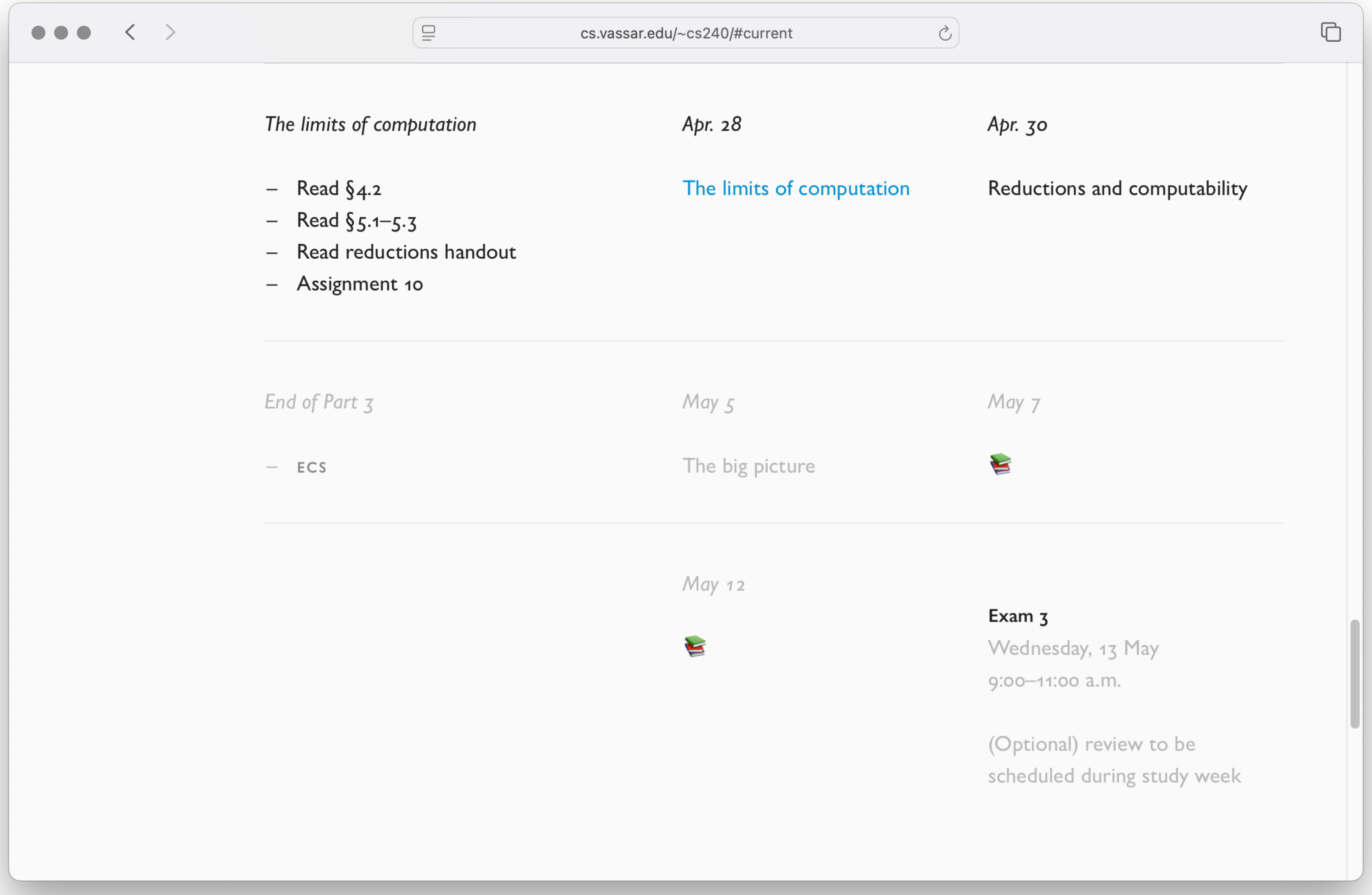
30 April 2026

Reductions and Computability



Reductions and Computability




cs.vassar.edu/~cs240/#current
The limits of computation
Apr. 28
Apr. 30
– Read §4.2
– Read §5.1–5.3
– Read reductions handout
– Assignment 10
The limits of computation
Reductions and computability
End of Part 3
May 5
May 7
– ECS
The big picture
May 12
Exam 3
Wednesday, 13 May
9:00–11:00 a.m.
(Optional) review to be
scheduled during study week

Assignment 10

Last assignment – out later today

Both the assignment and corrections are due on the usual schedule, even though the corrections will be due after the last day of class.

We'll do the end of course survey (ECS) at the end of class on Tuesday.

Where are we?

The class of Turing-decidable languages (**R**) represents problems that can truly be solved by a computer.

The class of Turing-recognizable languages (**RE**) represents problems where “yes” answers can be found – but “no” answers might be impossible to rule out.

We saw how to use self-reference to find languages that are *undecidable* – languages that are not in **R**.

```
def will_accept(fn: str, w: str) -> bool:
    """Returns True if fn(w) returns true.
    Otherwise, returns False."""
    ...


def trickster(input: str) -> bool:
    me = my_source()
    if will_accept(me, input):
        return False
    else:
        return True
```

A self-defeating object

Using that object against itself

THEOREM $A_{TM} \notin \mathbf{R}$.

PROOF By contradiction; assume that $A_{TM} \in \mathbf{R}$. Then there is some decider D for A_{TM}. We can represent D as a function

```
will_accept(fn: str, w: str) -> bool
```

that takes in the source code of a function `fn` and a string `w`, then returns true if `fn(w)` returns true and returns false otherwise.

Given this, consider this function:

```
def trickster(input: str) -> bool:
    me = my_source()
    if will_accept(me, input):
        return False
    else:
        return True
```

Since `will_accept` decides A_{TM} and `me` holds the source of `trickster`, we know that

`will_accept(me, input)` returns true
if and only if
`trickster(input)` returns true.

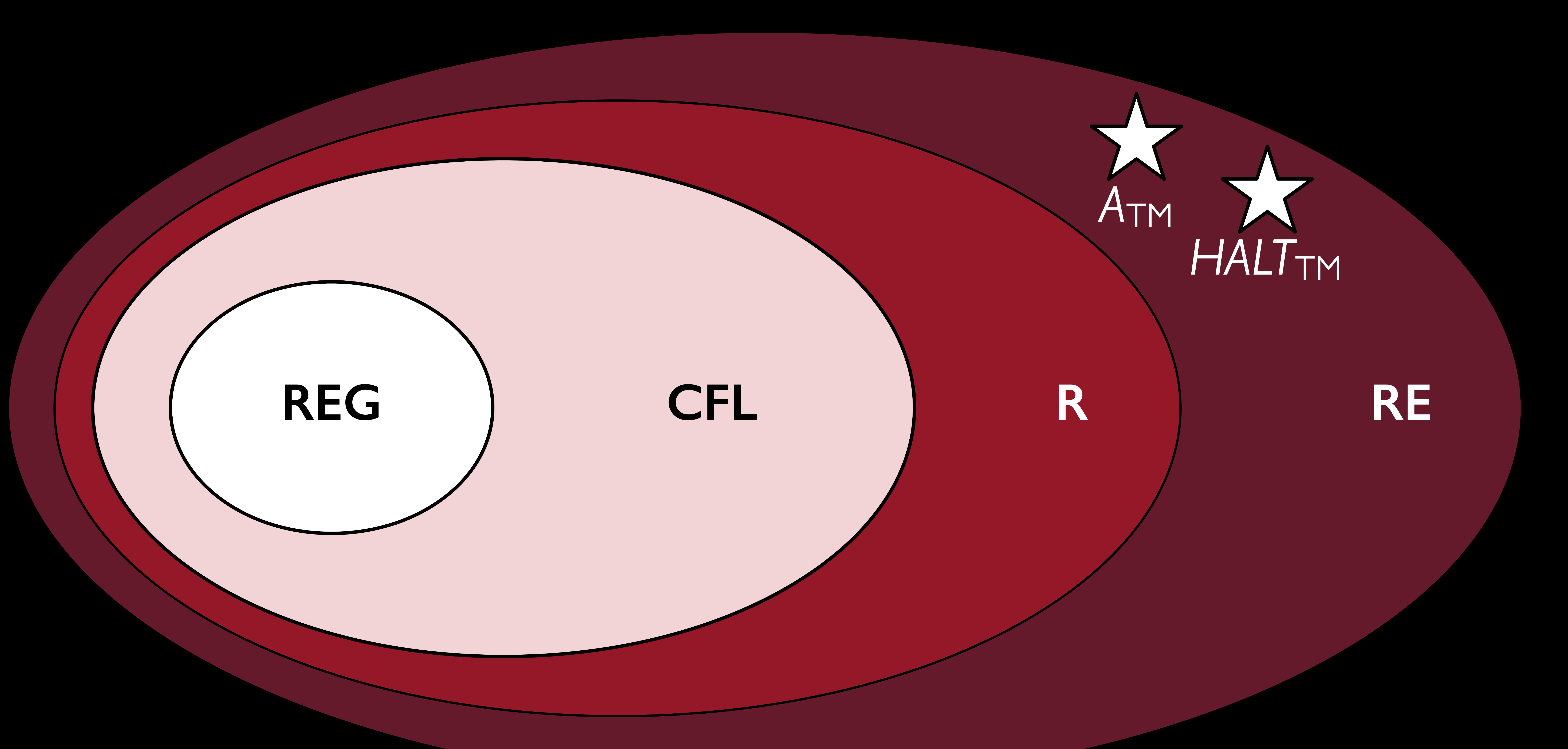
Given how `trickster` is written, we see that

`will_accept(me, input)` returns true
if and only if
`trickster(input)` doesn't return true.

This means that

`trickster(input)` returns true
if and only if
`trickster(input)` doesn't return true.

We've reached a contradiction, so our assumption was wrong and A_{TM} is undecidable. ■



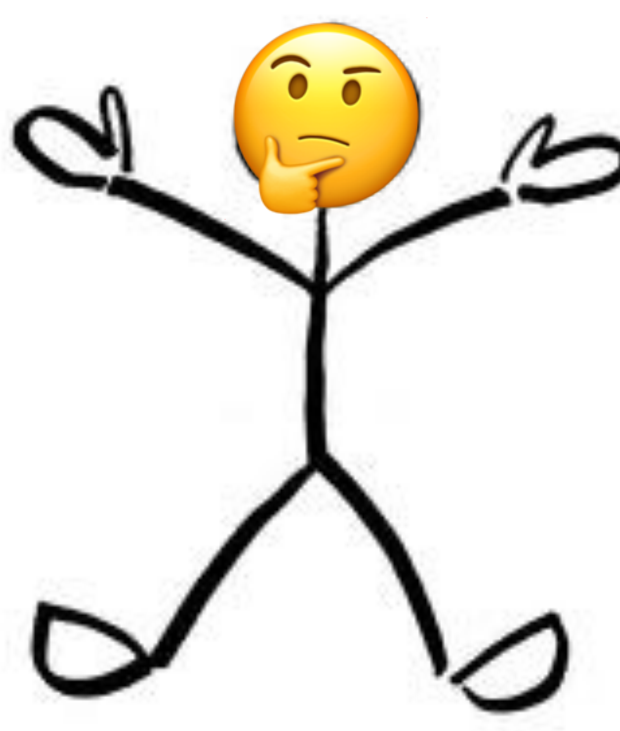
All languages

Reductions

We've already seen that A_{TM}, is ***undecidable*** – as is $HALT_{TM}$ and the secure voting problem.

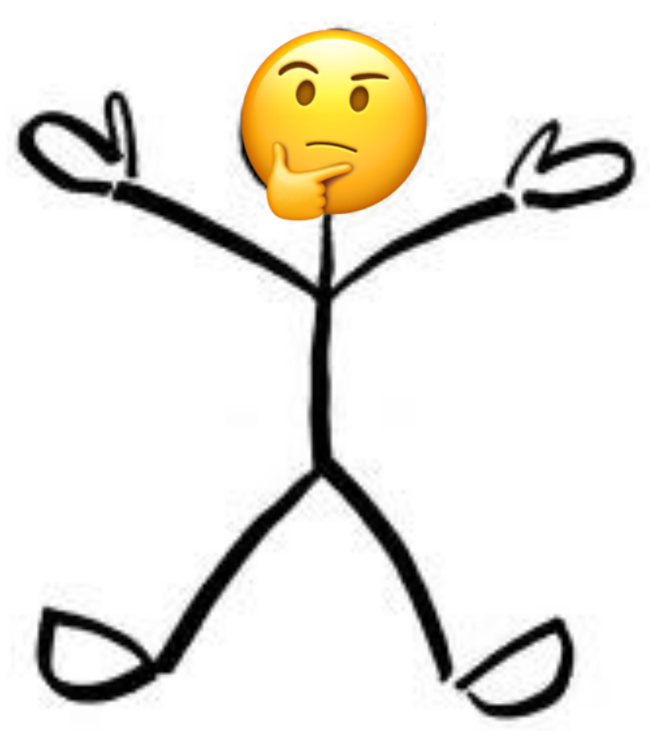
Can we use these results to show that other problems are also undecidable?

I wonder if I can lift that car…

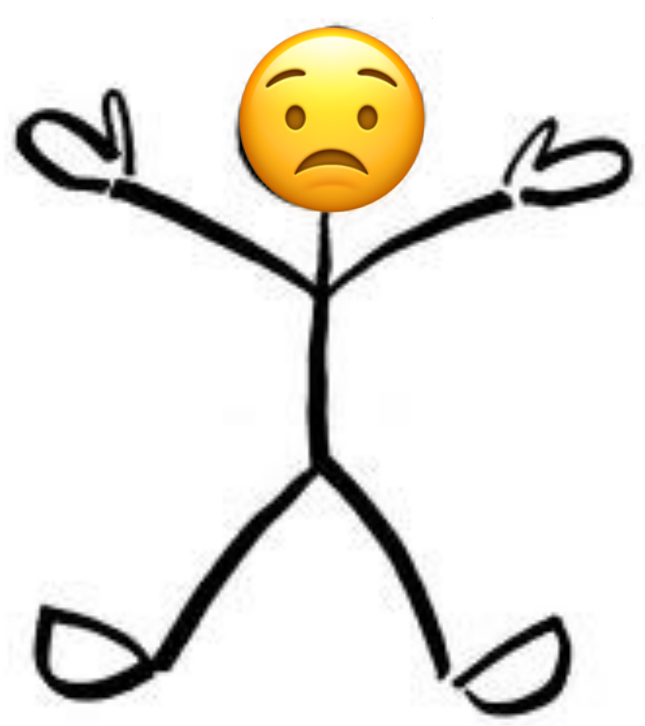


Nope! It turns out that cars are heavy!

I wonder if I can lift this fully loaded truck!



Oh no! It's got a car in the back!



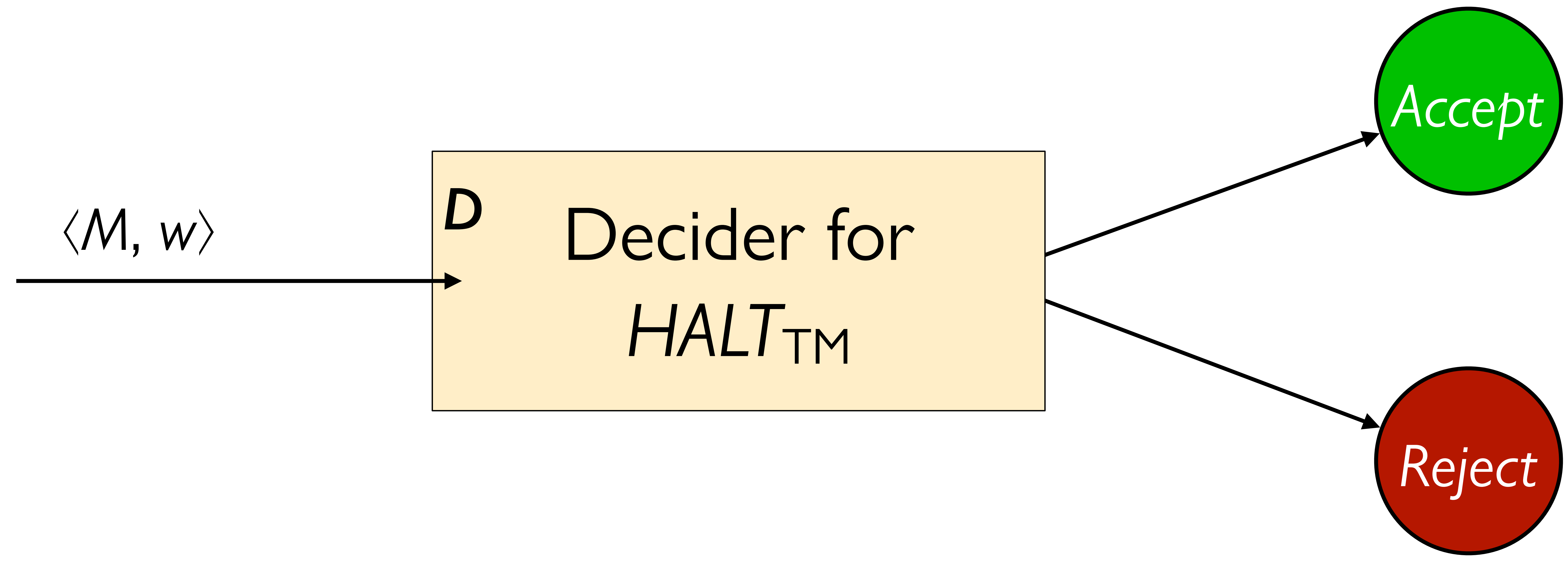
If I could lift the fully loaded truck, that would mean I could also lift the car.

I can't lift the truck.

A *reduction* that lets us show that one problem is at least as hard as another problem.

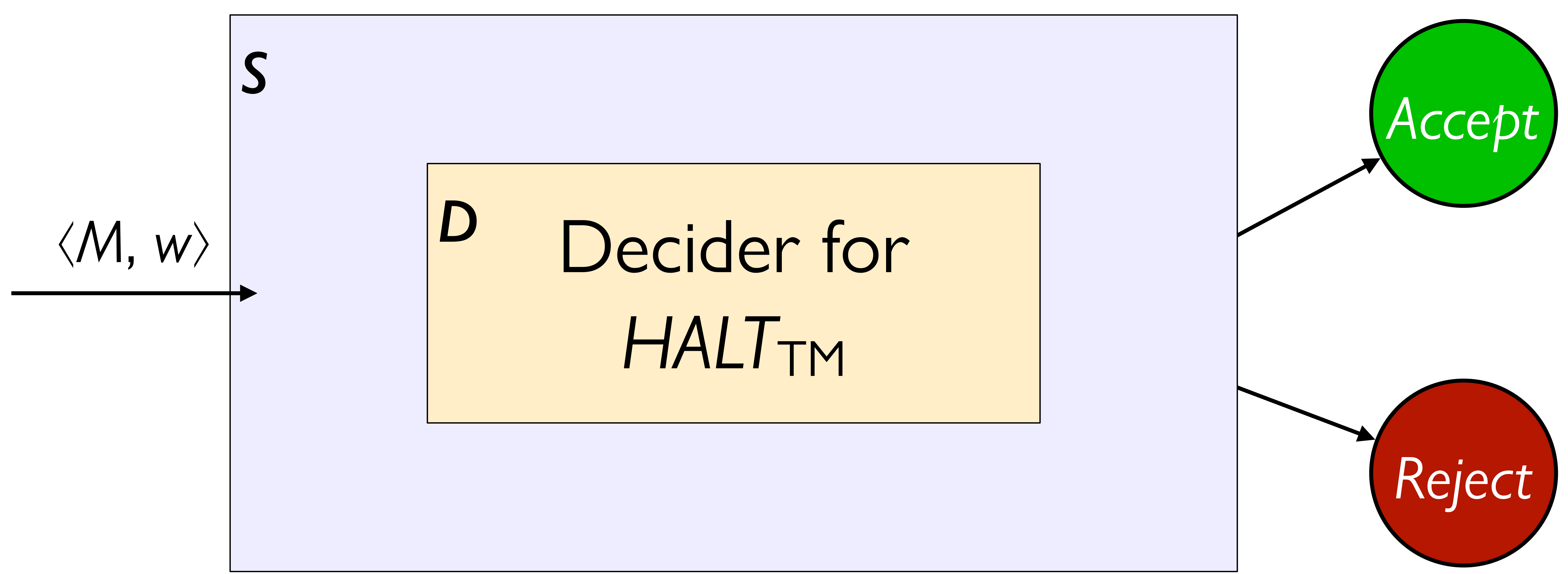
We proved the Halting Problem ($HALT_{\text{TM}}$) was undecidable by writing a proof showing that a decider for it is a self-defeating object, the same as we did for A_{TM}.

But the textbook showed that $HALT_{\text{TM}}$ was undecidable using a reduction from A_{TM}!



To prove that $HALT_{TM}$ is undecidable, you assume that *is* decidable – that is, there's a TM *D* that decides it.

D is what was named R in Sipser's reduction, but R makes me think "recognizer"



To prove that $HALT_{TM}$ is undecidable, you assume that *is* decidable – that is, there's a TM *D* that decides it.

Then you show that, armed with such a TM, you could implement another TM, *S*, that decides A_{TM}, which is a contradiction because we already know that A_{TM} is undecidable.

The direction of the reduction is the opposite of what most people intuitively think of first.

If you want to show that the Halting Problem is undecidable, you do ***not*** reduce the Halting Problem to A_{TM}; you reduce A_{TM} to the Halting Problem.

PROOF SKETCH Suppose *D* decides $HALT_{TM}$.

Then we could easily design a decider *S* for A_{TM}:

S = "On input ⟨*M*, *w*⟩,
Use *D* to check whether *M* loops on *w*.
If a loop is detected, *reject*.
If no loop is detected, we can safely simulate *M* on *w*.
If it accepts, *accept*.
If it rejects, *reject*."

Since we've proved that no such decider *S* can exist, that means *D* can't exist either!

Undecidable problems about programs

We want to show that it's undecidable whether some program P has some property ϕ.

We'll do so by reduction from A_{TM}.

You can use any known undecidable language, but usually A_{TM} or $HALT_{TM}$ are the easiest choices!

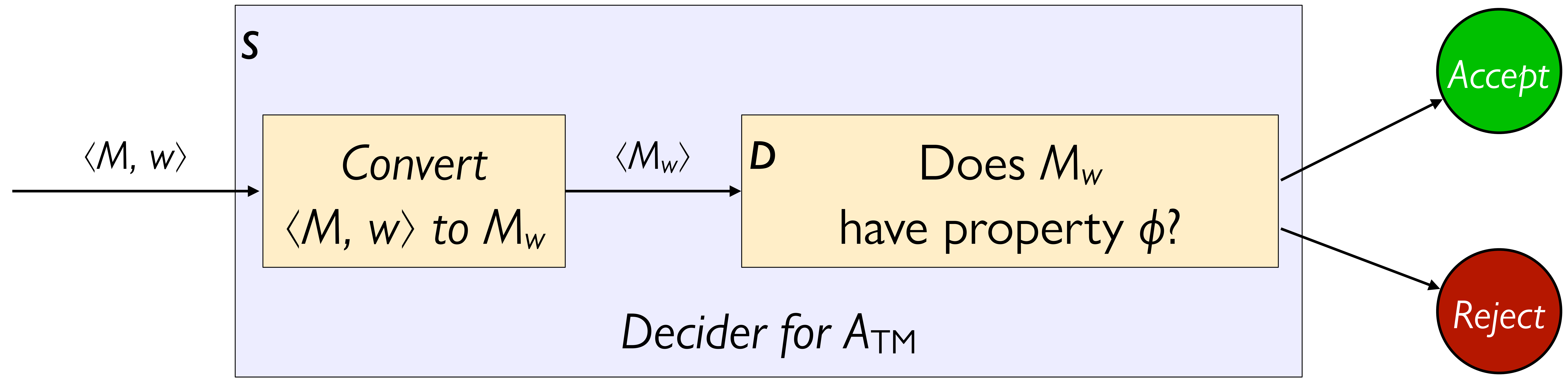
The proof goes like this:

Assume there is some TM D that can decide whether some program P has property ϕ. Then we want to use D to implement a TM S decides A_{TM}.

That implementation usually has three steps:

S = "On input $\langle M, w \rangle$,

1. Convert $\langle M, w \rangle$ into a program M_w.
2. Run D on $\langle M_w \rangle$.
3. If D accepts, *accept*; if D rejects, *reject*."



S = "On input ⟨M, w⟩,

1. Convert ⟨M, w⟩ into a program M_w.
2. Run D on ⟨M_w⟩.
3. If D accepts, *accept*; if D rejects, *reject*."

The proof goes like this:

Assume there is some TM D that can decide whether some program P has property ϕ. Then we want to use D to implement a TM S decides A_{TM}.

That implementation usually has three steps:

S = "On input $\langle M, w \rangle$,

1. Convert $\langle M, w \rangle$ into a program M_w.
2. Run D on $\langle M_w \rangle$.
3. If D accepts, *accept*; if D rejects, *reject*."

In our reduction from A_{TM} to the Halting Problem, Step 3 was more complex, but usually it's very simple, like this.

The proof goes like this:

Assume there is some TM D that can decide whether some program P has property ϕ. Then we want to use D to implement a TM S decides A_{TM}.

That implementation usually has three steps:

S = "On input $\langle M, w \rangle$,

1. Convert $\langle M, w \rangle$ into a program M_w.
2. Run D on $\langle M_w \rangle$.
3. If D accepts, *accept*; if D rejects, *reject*."

Step 2 is usually "Run D on $\langle M_w \rangle$" but might require one or more additional inputs depending on what D wants

The proof goes like this:

Assume there is some TM D that can decid[e] whether some program P has property ϕ. T[…] want to use D to implement a TM S decide[…]

That implementation usually has three step[s]

S = "On input $\langle M, w \rangle$,

1. Convert $\langle M, w \rangle$ into a program M_w.
2. Run D on $\langle M_w \rangle$.
3. If D accepts, *accept*; if D rejects, *reject*.

In our reduction from A_{TM} to the Halting Problem, Step 1 was trivial – $\langle M, w \rangle$ maps to itself – but, in general, Step 1 really has to do something: It has to change the TM and string into M_w, which acts as an "adapter" turning the property we want to detect (whether M accepts w) into the property that D is able to detect (ϕ).

The proof goes like this:

Assume there is some TM D that can decide whether some program P has property ϕ. Then we want to use D to implement a TM S decides A_{TM}.

That implementation usually has three steps:

S = "On input $\langle M, w \rangle$,

1. Convert $\langle M, w \rangle$ into a program M_w.
2. Run D on $\langle M_w \rangle$.
3. If D accepts, *accept*; if D rejects, *reject*."

When P is another Turing machine, Step 1 involves creating a third TM besides D and S.

A problem about Python programs

Let's show that it is undecidable whether a given Python program *P* deletes any files.

It would be great for security if this were decidable, but, unfortunately, it's not!

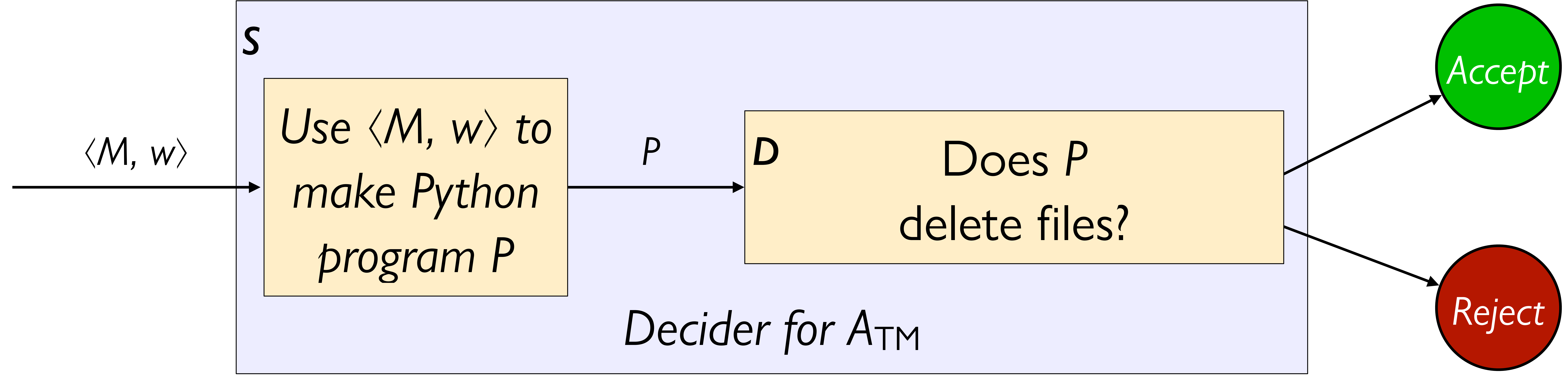
Suppose for the sake of contradiction that this is decidable. That is, there exists a TM D that accepts a Python program P if and only if P would delete any files.

We'll build a universal decider S that – somehow – uses D to decide A_{TM}.

Suppose for the sake of contradiction that this is decidable. That is, there exists a TM D that accepts a Python program P if and only if P would delete any files.

We'll build a universal decider S that – somehow – uses D to decide A_{TM}.

We can't feed $\langle M, w \rangle$ to D because D wants a Python program. So, we need to convert M and w into a Python program.



We can't feed $\langle M, w \rangle$ to D because D wants a Python program. So, we need to convert M and w into a Python program. We'll call it P rather than M_w since it's a Python program rather than a TM.

THEOREM It's undecidable whether a Python program deletes any files.

PROOF By contradiction; we assume there is a decider D that can take a Python program as input and accept if it deletes any files. Let the Python function `simulate` simulate a TM and returns `True` for accept or `False` for reject; this function can also potentially loop.

Then the decider for A_{TM} can be implemented as

S = "On input $\langle M, w \rangle$,

1. Construct the Python program, called P:

```
import os
if simulate(M, w):
   os.system("rm -rf *")
```

where M and w are filled in with data structures representing M and w, respectively.

2. Run D on P.
3. If D detected deletion of a file, *accept*.
4. Otherwise, *reject*."

To see that S is a decider for A_{TM}, let's walk through the possible cases:

1. If M accepts w, then P would wipe out your files. D detects this, and so S accepts.
2. If M rejects w, then P does not wipe out your files. D does not detect any writes, and so S rejects.
3. Similarly, if M loops on w, then P would run forever but would not wipe out your files. D does not detect any deletions, and so S rejects.

Thus S decides A_{TM} as desired, which is a contradiction. We conclude that it's undecidable whether a given Python program deletes any files. ■

The "adapter"

In general, when you're trying to prove that detecting property ϕ is undecidable, the program P usually has to do the following things:

1. Simulate M on w.
2. If M accepts, then exhibit property ϕ.
3. If M rejects, then don't exhibit property ϕ.
4. You must also set it up so that if M loops, property ϕ is not exhibited.

Proof by reduction: $REGULAR_{TM}$ is undecidable

Can we detect whether a TM recognizes a regular language?

Let $REGULAR_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$.

Can we detect whether a TM recognizes a regular language?

Let $REGULAR_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$.

This is undecidable, even though it doesn't have a simple mapping to A_{TM} or $HALT_{TM}$.

Proof idea: Suppose $REGULAR_{TM}$ is decidable by some machine D.

Proof idea: Suppose $REGULAR_{TM}$ is decidable by some machine D.

Given a TM M and a string w, construct a TM M' with these properties:

If M halts on w, then $L(M')$ is regular.

If M loops on w, then $L(M')$ is not regular.

Proof idea: Suppose $REGULAR_{TM}$ is decidable by some machine D.

Given a TM M and a string w, construct a TM M' with these properties:

If M halts on w, then $L(M')$ is regular.

If M loops on w, then $L(M')$ is not regular.

How do we build a machine with these properties?

Proof idea: Suppose $REGULAR_{TM}$ is decidable by some machine D.

Given a TM M and a string w, construct a TM M' with these properties:

If M halts on w, then $L(M') = \Sigma^*$.

If M loops on w, then $L(M')$ is not regular.

How do we build a machine with these properties?

Proof idea: Suppose $REGULAR_{TM}$ is decidable by some machine D.

Given a TM M and a string w, construct a TM M' with these properties:

If M halts on w, then $L(M') = \Sigma^*$.

If M loops on w, then $L(M') = \{0^n1^n \mid n \in \mathbb{N}_0\}$.

How do we build a machine with these properties?

Proof idea: Suppose $REGULAR_{TM}$ is decidable by some machine D.

Given a TM M and a string w, construct a TM M' with these properties:

If M halts on w, then $L(M') = \Sigma^*$.

If M loops on w, then $L(M') = \{0^n 1^n \mid n \in \mathbb{N}_0\}$.

How do we build a machine with these properties?

Proof idea: Suppose $REGULAR_{TM}$ is decidable by some machine D.

Given a TM M and a string w, construct a TM M' with these properties:

If M halts on w, then $L(M') = \Sigma^*$.

If M loops on w, then $L(M') = \{0^n 1^n \mid n \in \mathbb{N}_0\}$.

Proof idea: Suppose $REGULAR_{TM}$ is decidable by some machine D.

Given a TM M and a string w, construct a TM M' with these properties:

If M halts on w, then $L(M') = \Sigma^*$.

If M loops on w, then $L(M') = \{0^n1^n \mid n \in \mathbb{N}_0\}$.

Have D decide whether or not $L(M')$ is regular.

If $L(M')$ is regular, M halts on w.

If $L(M')$ is not regular, M loops on w.

Proof idea: Suppose $REGULAR_{TM}$ is decidable by some machine D.

Given a TM M and a string w, construct a TM M' with these properties:

If M halts on w, then $L(M') = \Sigma^*$.

If M loops on w, then $L(M') = \{0^n1^n \mid n \in \mathbb{N}_0\}$.

Have D decide whether or not $L(M')$ is regular.

If $L(M')$ is regular, M halts on w.

If $L(M')$ is not regular, M loops on w.

We can use D to decide $HALT_{TM}$, which is a contradiction.

Proof idea: Suppose $REGULAR_{TM}$ is decidable by some machine D.

Given a TM M and a string w, construct a TM M' with these properties:

If M halts on w, then $L(M') = \Sigma^*$.

If M loops on w, then $L(M') = \{0^n 1^n \mid n \in \mathbb{N}_0\}$.

*Ok, but **really**, how do we make a machine that changes its language like this?*

Have D decide whether or not $L(M')$ is regular.

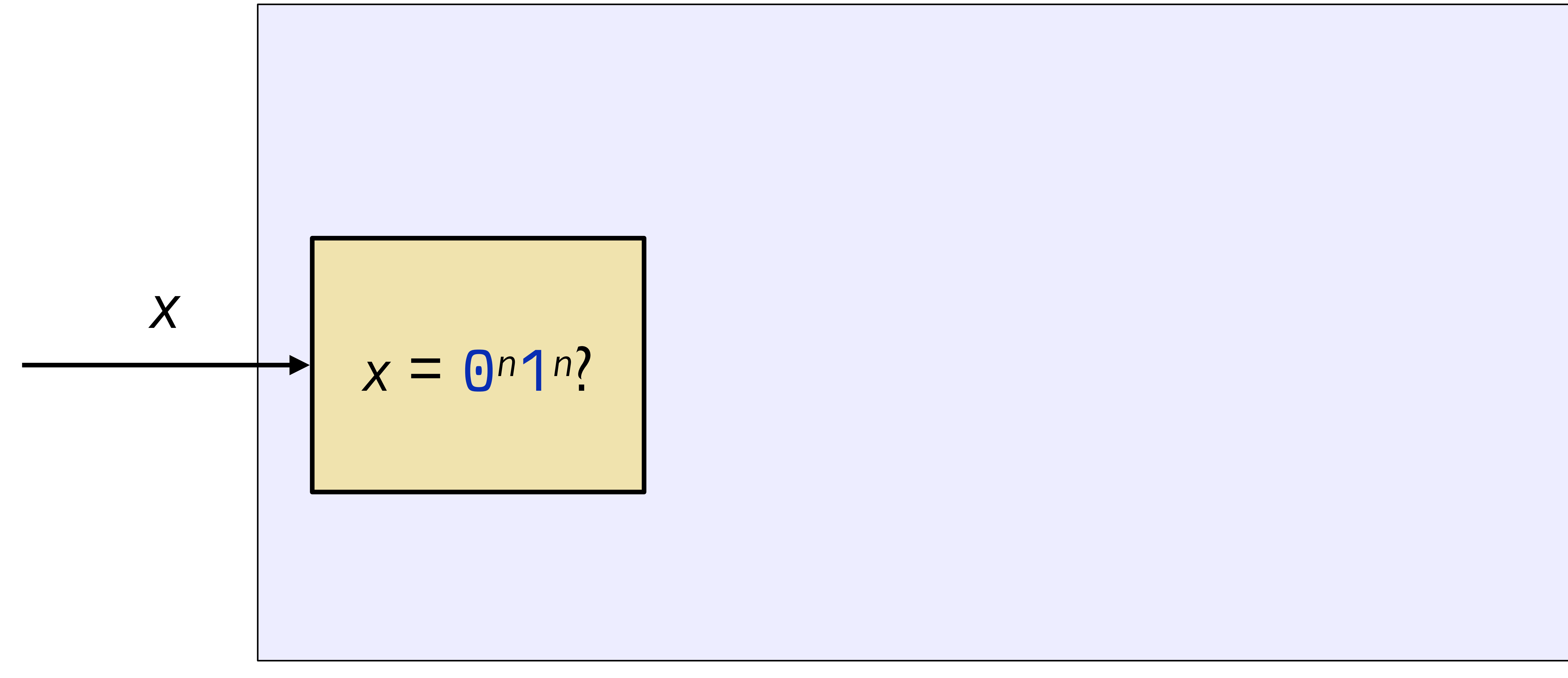
If $L(M')$ is regular, M halts on w.

If $L(M')$ is not regular, M loops on w.

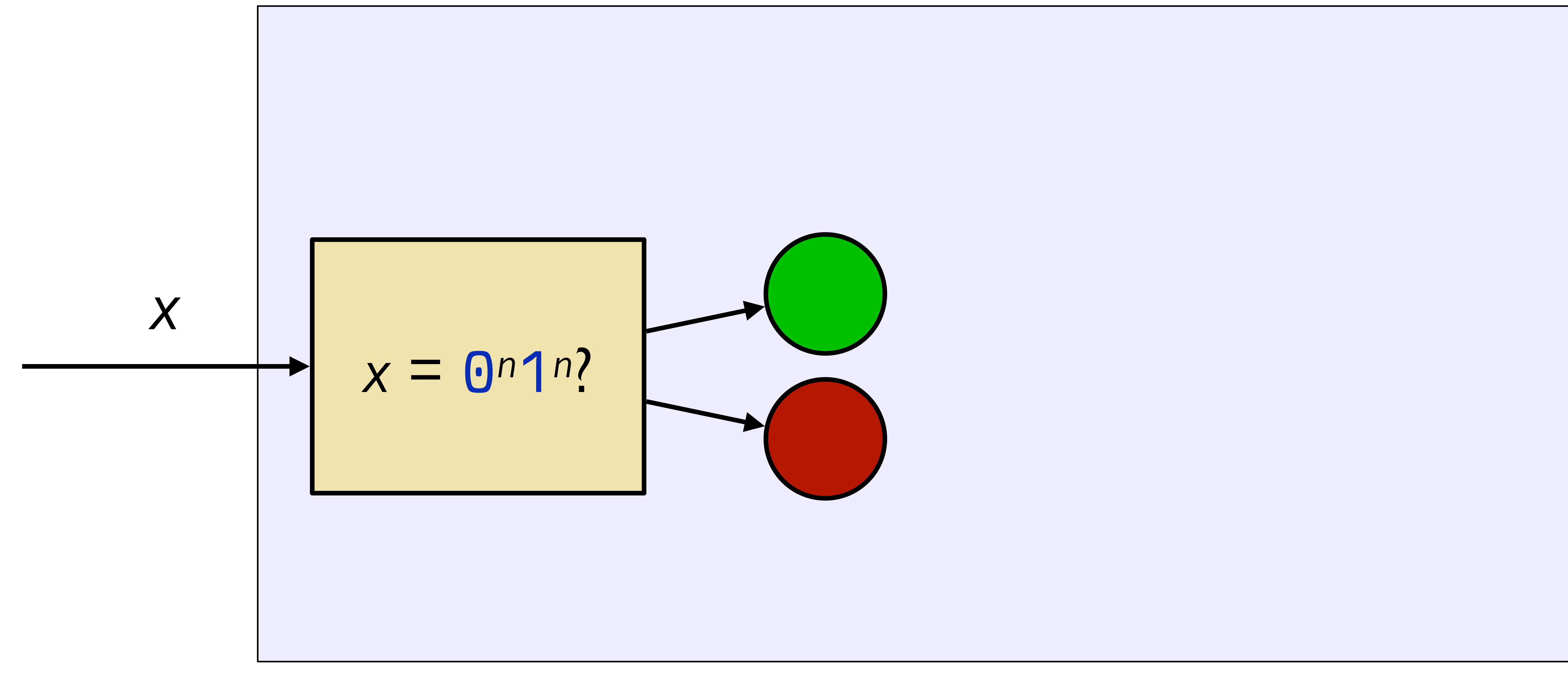
We can use D to decide $HALT_{TM}$, which is a contradiction.

The mysterious machine

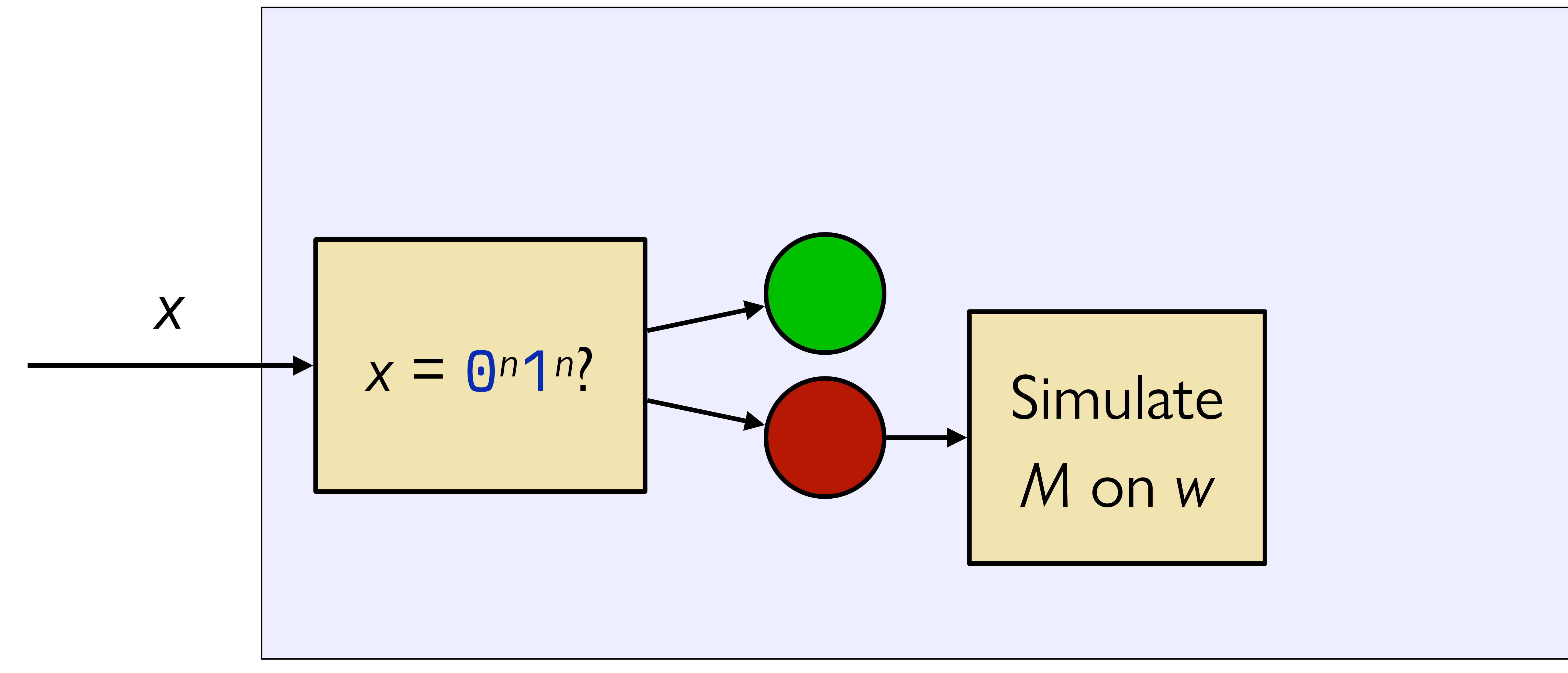
The mysterious machine



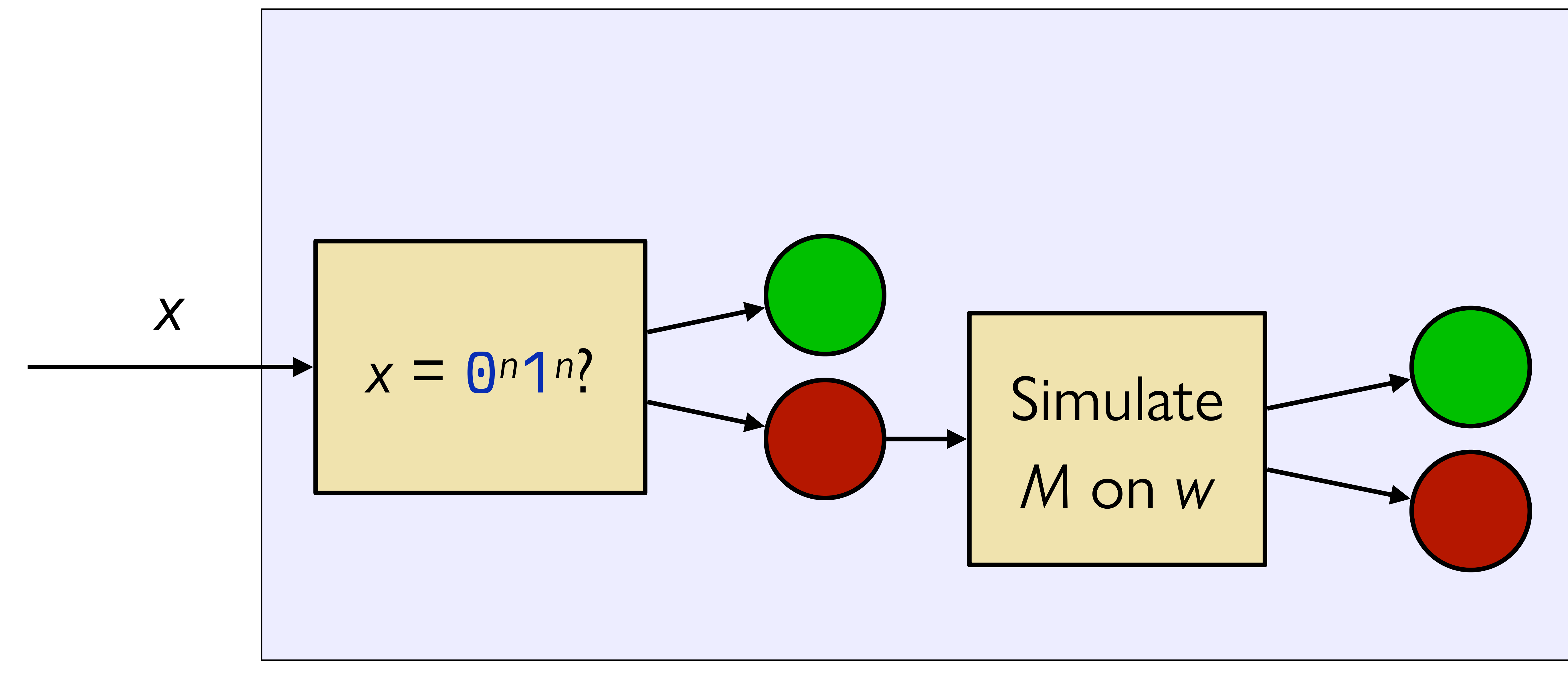
The mysterious machine



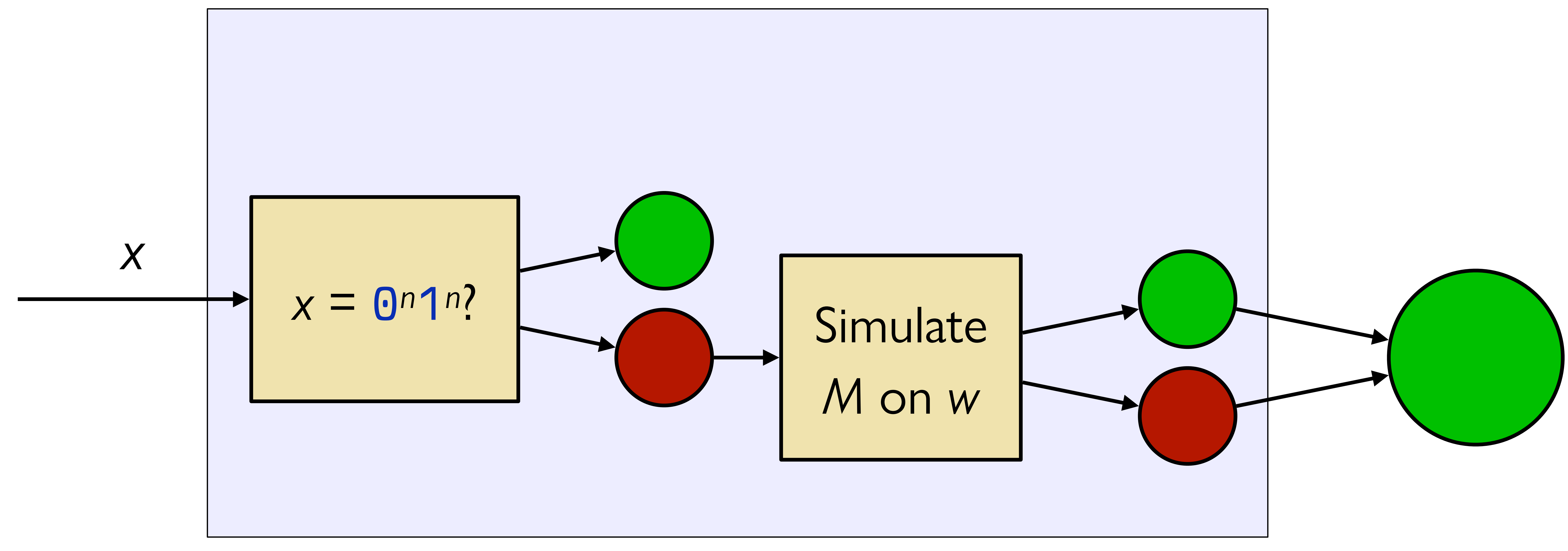
The mysterious machine



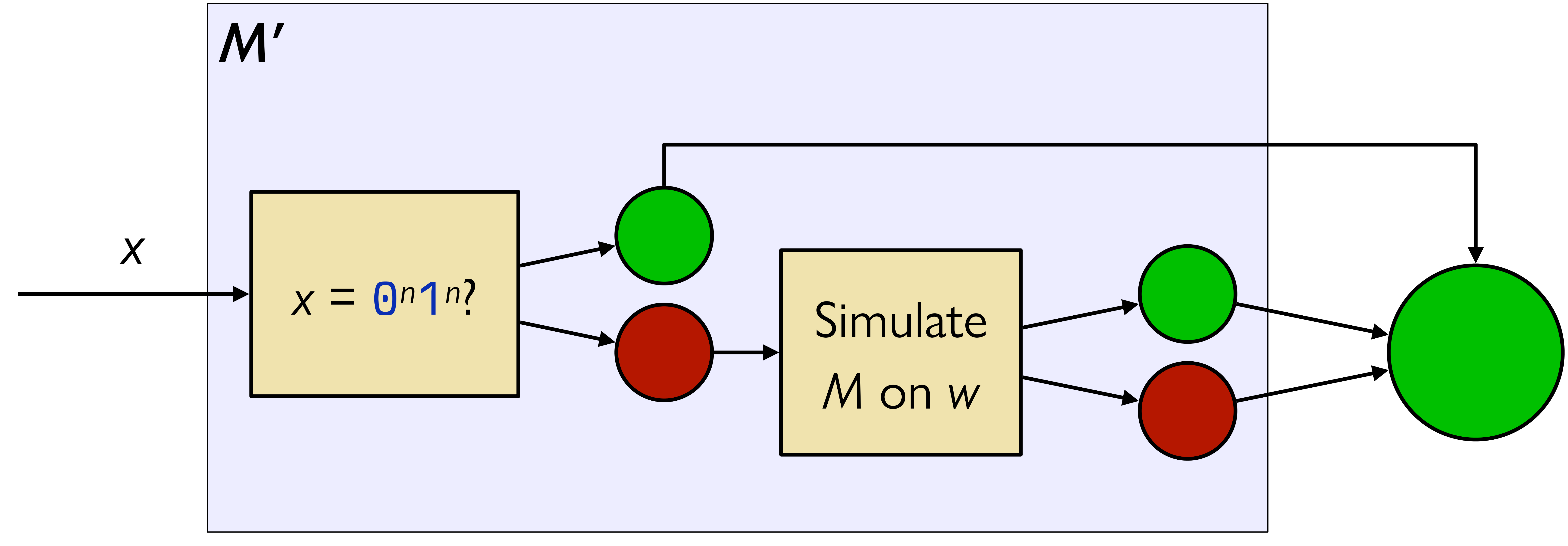
The mysterious machine



The mysterious machine



The mysterious machine



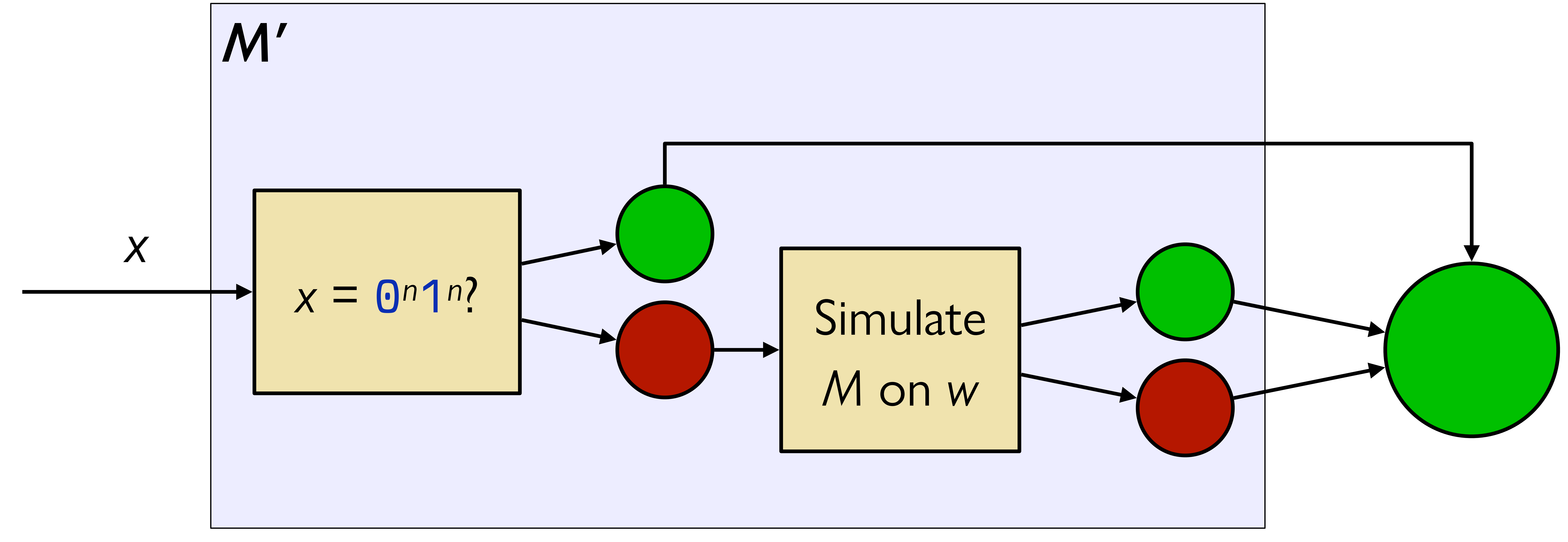
M' = "On input x:

If $x = 0^n1^n$, *accept*.

Otherwise, run M on w.

If M halts, *accept*."

The mysterious machine



If M halts on w, M′ accepts all strings – its language is Σ.*

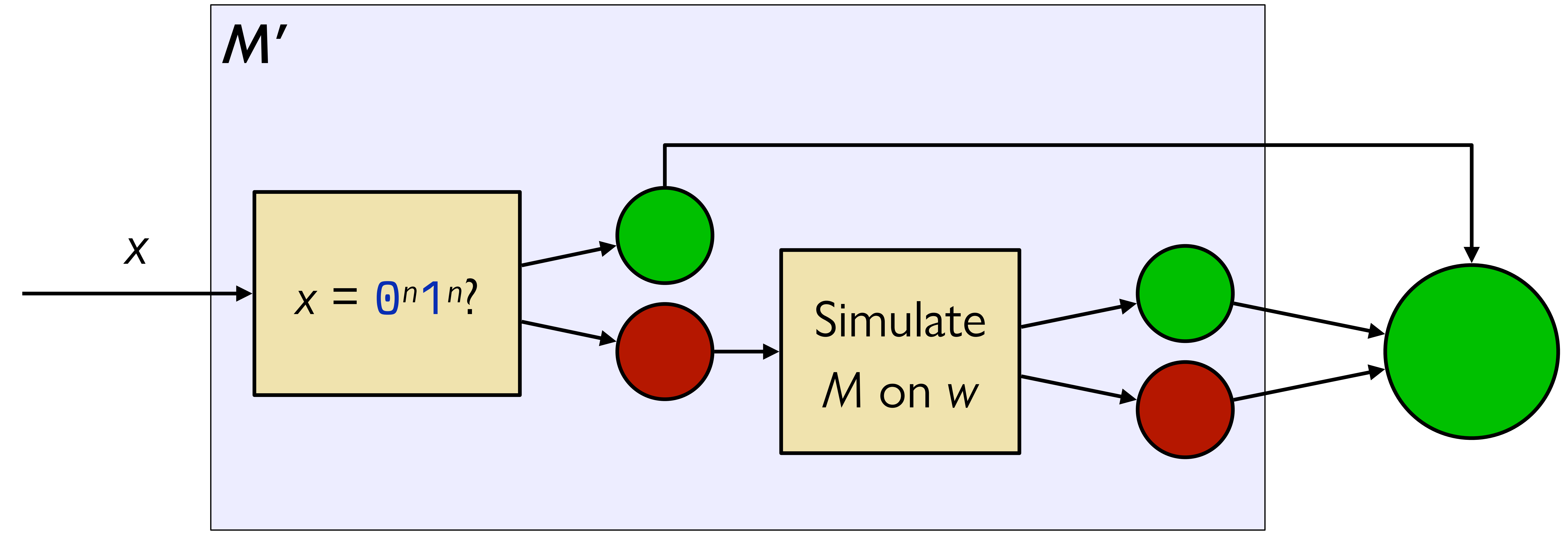
M' = "On input x:

If $x = 0^n1^n$, *accept*.

Otherwise, run M on w.

If M halts, *accept*."

The mysterious machine



If M halts on w, M′ accepts all strings – its language is Σ.*

M' = "On input x:

If $x = 0^n1^n$, *accept*.

Otherwise, run M on w.

If M halts, *accept*."

If M loops on w, M′ only accepts strings of the form 0^n1^n.

THEOREM *$REGULAR_{TM}$* is undecidable.

PROOF By contradiction; assume D decides $REGULAR_{TM}$. Consider the following machine H:

> H = "On input $\langle M, w\rangle$:
>
> Construct the machine M' = "On input x:
>
> If x has the form 0^n1^n, accept.
>
> Otherwise, run M on w.
>
> If M halts on w, accept."
>
> Run D on $\langle M'\rangle$.
>
> If D accepts, accept; if D rejects, reject."

We claim that H is a decider and that $L(H) = HALT_{TM}$. To see that H is a decider, note that after H constructs M', H runs D on $\langle M'\rangle$. Since D is a decider, D always halts. If D accepts, H accepts, and if D rejects, H rejects. Thus H halts on all inputs.

To see that $L(H) = HALT_{TM}$, note that H accepts $\langle M, w\rangle$ iff D accepts $\langle M'\rangle$. Since D decides $REGULAR_{TM}$, D accepts $\langle M'\rangle$ iff $L(M')$ is regular. We claim that $L(M')$ is regular iff M halts on w. To see this, note that if M halts on w, M' accepts all strings, either because the string has form 0^n1^n or because it accepts in the final step after M halts. Thus $L(M') = \{0^n1^n \mid n \in \mathbb{N}_0\}$, which is not regular. Thus H accepts $\langle M, w\rangle$ iff M halts on w iff $\langle M, w\rangle \in HALT_{TM}$, so $L(H) = HALT_{TM}$.

We have reached a contradiction, because we know that $HALT_{TM}$ is undecidable. Thus our assumption was wrong and $REGULAR_{TM}$ is undecidable. ■

Rice’s Theorem

The story so far

Consider the following problems:

Does M accept w?

Does M halt on w?

Is $L(M)$ regular?

The story so far

Consider the following problems:

Does M accept w? *Undecidable!*

Does M halt on w?

Is $L(M)$ regular?

The story so far

Consider the following problems:

Does M accept w?	*Undecidable!*
Does M halt on w?	*Undecidable!*
Is $L(M)$ regular?	

The story so far

Consider the following problems:

Does M accept w?	*Undecidable!*
Does M halt on w?	*Undecidable!*
Is $L(M)$ regular?	

The story so far

Consider the following problems:

Does M accept w?	*Undecidable!*
Does M halt on w?	*Undecidable!*
Is $L(M)$ regular?	*Undecidable!*

The story so far

Consider the following problems:

Does M accept w?	*Undecidable!*
Does M halt on w?	*Undecidable!*
Is $L(M)$ regular?	*Undecidable!*

The story so far

Consider the following problems:

Does M accept w?	*Undecidable!*
Does M halt on w?	*Undecidable!*
Is $L(M)$ regular?	*Undecidable!*

There seems to be a trend here.

It turns out that most interesting questions about properties of Turing machines – and thus of computer programs – are undecidable!

A ***property of an* RE *language*** is some trait that may apply to **RE** languages.

For example:

Does $L = \varnothing$?

Is L regular?

Is L context-free?

Does L contain any string of length exactly 137?

We can describe a property of an **RE** language as the set of **RE** languages with that property.

P is the set of **RE** languages with property *P*.

If *P* is a property of **RE** languages, consider the language

$$L_P = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \in \mathbf{P}\}$$

L_P is the set of TMs that recognize a language that has property *P*.

Note that membership in L_P depends only on the *language* of a TM, not the *description* of that TM!

$$\text{If } L(M_1) = L(M_2)\text{, then } \langle M_1 \rangle \in L_P \text{ iff } \langle M_2 \rangle \in L_P$$

If two different TMs have the same language, they both have the same property.

$$L_{\text{even}} = \{\langle M \rangle \mid L(M) \text{ is finite and } |L(M)| \text{ is even}\}$$

This is a property of **RE** languages, because it depends *purely* on the language of the TM and not on the TM itself.

Specifically, if $L(M_1) = L(M_2)$,
then $\langle M_1 \rangle \in L_{\text{even}}$ iff $\langle M_2 \rangle \in L_{\text{even}}$

$L_{\text{even}} = \{\langle M \rangle \mid L(M) \text{ is finite and } |L(M)| \text{ is even}\}$

This is a property of **RE** languages, because it depends *purely* on the language of the TM and not on the TM itself.

Specifically, if $L(M_1) = L(M_2)$,
then $\langle M_1 \rangle \in L_{\text{even}}$ iff $\langle M_2 \rangle \in L_{\text{even}}$

$L_{\text{evenQ}} = \{\langle M \rangle \mid M \text{ has an even number of states}\}$

This is *not* a property of **RE** languages, because it does not depend purely on the language of the TM.

Specifically, if $L(M_1) = L(M_2)$,
then it may be possible for $\langle M_1 \rangle \in L_{\text{evenQ}}$ but $\langle M_2 \rangle \notin L_{\text{evenQ}}$

A property of **RE** languages is called ***trivial*** if *all* **RE** languages have the property or *no* **RE** languages have the property, e.g.,

$\{\langle M \rangle \mid L(M)$ is **RE**$\}$ is trivial

$\{\langle M \rangle \mid L(M)$ is not **RE**$\}$ is trivial

A property of **RE** languages is called ***nontrivial*** if there exist TMs M_1 and M_2 such that $\langle M_1 \rangle \in L_P$, but $\langle M_2 \rangle \notin L_P$, e.g.,

$\{\langle M \rangle \mid L(M)$ is infinite$\}$ is nontrivial

$\{\langle M \rangle \mid L(M)$ is regular$\}$ is nontrivial

$\{\langle M \rangle \mid L(M)$ is decidable$\}$ is nontrivial

Rice's Theorem

Any nontrivial property of the **RE** languages is *undecidable*.

Can we apply Rice's Theorem to this language?

$$L_{\text{ne}} = \{\langle M \rangle \mid L(M) \neq \varnothing\}$$

Can we apply Rice's Theorem to this language?

$$L_{\text{ne}} = \{\langle M \rangle \mid L(M) \neq \varnothing\}$$

We can apply Rice's Theorem if two conditions hold:

Can we apply Rice's Theorem to this language?

$$L_{\text{ne}} = \{\langle M \rangle \mid L(M) \neq \varnothing\}$$

We can apply Rice's Theorem if two conditions hold:

L_{ne} is nontrivial:

$$\exists\, M_1 \,.\, \exists\, M_2 \,.\, \langle M_1 \rangle \in L_{\text{ne}} \wedge \langle M_2 \rangle \notin L_{\text{ne}}$$

Can we apply Rice's Theorem to this language?

$$L_{\text{ne}} = \{\langle M \rangle \mid L(M) \neq \varnothing\}$$

We can apply Rice's Theorem if two conditions hold:

✅ *L_{ne} is nontrivial:*

$$\exists\, M_1 \,.\, \exists\, M_2 \,.\, \langle M_1 \rangle \in L_{\text{ne}} \wedge \langle M_2 \rangle \notin L_{\text{ne}}$$

Can we apply Rice's Theorem to this language?

$$L_{\text{ne}} = \{\langle M \rangle \mid L(M) \neq \varnothing\}$$

We can apply Rice's Theorem if two conditions hold:

✅ *L_{ne} is nontrivial:*

$$\exists M_1 \,.\, \exists M_2 \,.\, \langle M_1 \rangle \in L_{\text{ne}} \wedge \langle M_2 \rangle \notin L_{\text{ne}}$$

L_{ne} is a property of RE languages:

If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{\text{ne}}$ iff $\langle M_2 \rangle \in L_{\text{ne}}$

Can we apply Rice's Theorem to this language?

$$L_{\text{ne}} = \{\langle M \rangle \mid L(M) \neq \varnothing\}$$

We can apply Rice's Theorem if two conditions hold:

✅ *L_{ne} is nontrivial:*

$\exists\, M_1 \,.\, \exists\, M_2 \,.\, \langle M_1 \rangle \in L_{\text{ne}} \land \langle M_2 \rangle \notin L_{\text{ne}}$

✅ *L_{ne} is a property of RE languages:*

If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{\text{ne}}$ iff $\langle M_2 \rangle \in L_{\text{ne}}$

Can we apply Rice's Theorem to this language?

$$L_{\text{ne}} = \{\langle M \rangle \mid L(M) \neq \varnothing\}$$

We can apply Rice's Theorem if two conditions hold:

✅ *L_{ne} is nontrivial:*

$$\exists\, M_1 \,.\, \exists\, M_2 \,.\, \langle M_1 \rangle \in L_{\text{ne}} \wedge \langle M_2 \rangle \notin L_{\text{ne}}$$

✅ *L_{ne} is a property of RE languages:*

If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{\text{ne}}$ iff $\langle M_2 \rangle \in L_{\text{ne}}$

Rice's Theorem applies, so L_{ne} is undecidable!

Can we apply Rice's Theorem to this language?

$L_{\text{es}} = \{\langle M \rangle \mid M \text{ has an even number of states}\}$

Can we apply Rice's Theorem to this language?

$$L_{\text{es}} = \{\langle M \rangle \mid M \text{ has an even number of states}\}$$

We can apply Rice's Theorem if two conditions hold:

Can we apply Rice's Theorem to this language?

$$L_{es} = \{\langle M \rangle \mid M \text{ has an even number of states}\}$$

We can apply Rice's Theorem if two conditions hold:

L_{es} is nontrivial:

$$\exists M_1 . \exists M_2 . \langle M_1 \rangle \in L_{es} \wedge \langle M_2 \rangle \notin L_{es}$$

Can we apply Rice's Theorem to this language?

$$L_{es} = \{\langle M \rangle \mid M \text{ has an even number of states}\}$$

We can apply Rice's Theorem if two conditions hold:

✅ ***L_{es} is nontrivial:***

$$\exists\, M_1 \,.\, \exists\, M_2 \,.\, \langle M_1 \rangle \in L_{es} \wedge \langle M_2 \rangle \notin L_{es}$$

Can we apply Rice's Theorem to this language?

$$L_{es} = \{\langle M \rangle \mid M \text{ has an even number of states}\}$$

We can apply Rice's Theorem if two conditions hold:

✅ *L_{es} is nontrivial:*

$\exists\, M_1 \,.\, \exists\, M_2 \,.\, \langle M_1 \rangle \in L_{es} \land \langle M_2 \rangle \notin L_{es}$

L_{es} is a property of RE languages:

If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{es}$ iff $\langle M_2 \rangle \in L_{es}$

Can we apply Rice's Theorem to this language?

$$L_{es} = \{\langle M \rangle \mid M \text{ has an even number of states}\}$$

We can apply Rice's Theorem if two conditions hold:

✅ *L_{es} is nontrivial:*

$$\exists\, M_1 \,.\, \exists\, M_2 \,.\, \langle M_1 \rangle \in L_{es} \land \langle M_2 \rangle \notin L_{es}$$

❌ *L_{es} is a property of RE languages:*

If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{es}$ iff $\langle M_2 \rangle \in L_{es}$

Can we apply Rice's Theorem to this language?

$$L_{es} = \{\langle M \rangle \mid M \text{ has an even number of states}\}$$

We can apply Rice's Theorem if two conditions hold:

✅ *L_{es} is nontrivial:*

$$\exists\, M_1 \,.\, \exists\, M_2 \,.\, \langle M_1 \rangle \in L_{es} \wedge \langle M_2 \rangle \notin L_{es}$$

❌ *L_{es} is a property of RE languages:*

If $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in L_{es}$ iff $\langle M_2 \rangle \in L_{es}$

Rice's Theorem does not apply, so we can't draw any conclusion using it!

Rice's Theorem tells us that *all* of the following problems are undecidable:

$L_{\text{palindrome}} = \{\langle M \rangle \mid \text{every string in } L(M) \text{ is a palindrome}\}$

$L_{\text{allodd}} = \{\langle M \rangle \mid \text{every string in } L(M) \text{ has odd length}\}$

$L_{\text{CFL}} = \{\langle M \rangle \mid L(M) \text{ is a context-free language}\}$

$L_{\text{short}} = \{\langle M \rangle \mid L(M) \text{ has no strings of length greater than } 5\}$

$L_{\text{decidable}} = \{\langle M \rangle \mid L(M) \text{ is decidable}\}$

$E_{\text{TM}} = \{\langle M \rangle \mid L(M) = \varnothing\}$

The proof of Rice's theorem is a generalization of the reductions we've seen so far.

We won't have time to go through the proof in detail, but the general idea is: If L_P is a nontrivial property of **RE** languages, show that we can reduce $HALT_{\text{TM}}$ to L_P.

Beyond **R** and **RE**

The **RE** languages are the ones where *membership* can be proven (although *non-membership* may be impossible to prove).

So, what does a non-**RE** language look like?

Intuitively, a language is ***not*** in **RE** if there's no general way to prove that a given string $w \in L$ actually belongs to L.

In other words, even if you knew that a string was in the language, you may never be able to convince anyone of it!

The first non-**RE** language we're going to see is one that is, essentially, constructed specifically to not be an **RE** language.

Recall: We say that M is a recognizer for L if the following is true:

$$\forall\, w \in \Sigma^* .\ (w \in L \Leftrightarrow M \text{ accepts } w)$$

This description applies to all strings, including strings that – by pure coincidence – happen to be the encoding of a TM.

Recall: We say that M is a recognizer for L if the following is true:

$$\forall w \in \Sigma^* . (w \in L \Leftrightarrow M \text{ accepts } w)$$

This description applies to all strings, including strings that – by pure coincidence – happen to be the encoding of a TM.

Idea: Let's think about different Turing machines and how they behave when they're given a Turing machine as input.

M_0

M_1

M_2

M_3

M_4

M_5

$\vdots$

M_0

M_1

M_2

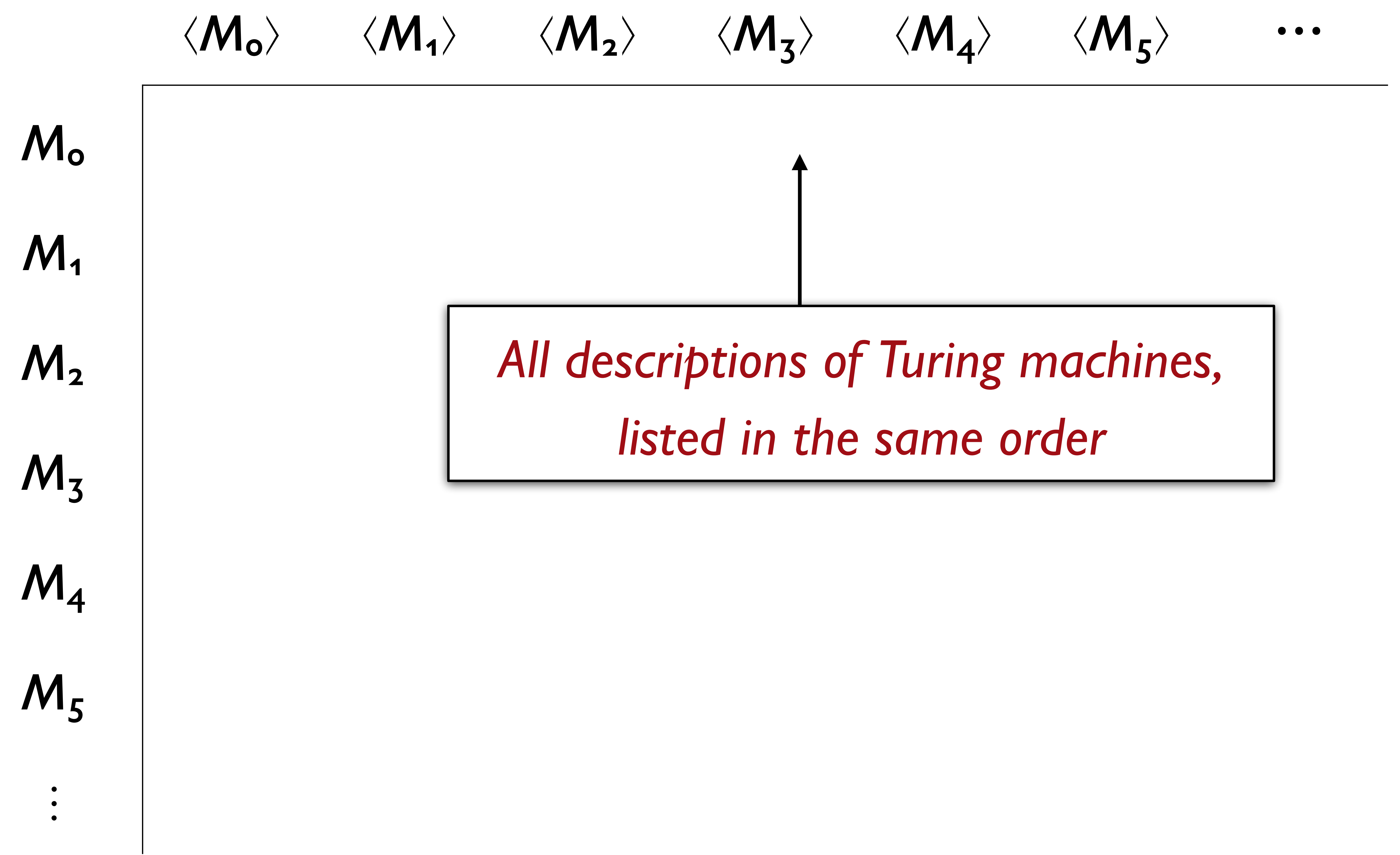
M_3

M_4

M_5

⋮

All Turing machines, listed in some order


⟨M0⟩ ⟨M1⟩ ⟨M2⟩ ⟨M3⟩ ⟨M4⟩ ⟨M5⟩ …
M0
M1
M2
M3
M4
M5
⋮
All descriptions of Turing machines,
listed in the same order

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1							
M_2							
M_3							
M_4							
M_5							
$\vdots$							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2							
M_3							
M_4							
M_5							
$\vdots$							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3							
M_4							
M_5							
$\vdots$							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4							
M_5							
$\vdots$							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4	Acc	No	Acc	No	Acc	No	$\cdots$
M_5							
$\vdots$							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4	Acc	No	Acc	No	Acc	No	$\cdots$
M_5	No	No	Acc	Acc	No	No	$\cdots$
$\vdots$							

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4	Acc	No	Acc	No	Acc	No	$\cdots$
M_5	No	No	Acc	Acc	No	No	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4	Acc	No	Acc	No	Acc	No	$\cdots$
M_5	No	No	Acc	Acc	No	No	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	…

Acc	Acc	Acc	No	Acc	No	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	…

No	No	No	Acc	No	Acc	…

Flip all "accept"s to "no"s and vice versa.

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4	Acc	No	Acc	No	Acc	No	$\cdots$
M_5	No	No	Acc	Acc	No	No	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

No	No	No	Acc	No	Acc	$\cdots$

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	⋯
M_0	Acc	No	No	Acc	Acc	No	⋯
M_1	Acc	Acc	Acc	Acc	Acc	Acc	⋯
M_2	Acc	Acc	Acc	Acc	Acc	Acc	⋯
M_3	No	Acc	Acc	No	Acc	Acc	⋯
M_4	Acc	No	Acc	No	Acc	No	⋯
M_5	No	No	Acc	Acc	No	No	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋯

No	No	No	Acc	No	Acc	⋯

What TM has this behavior?

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4	Acc	No	Acc	No	Acc	No	$\cdots$
M_5	No	No	Acc	Acc	No	No	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

No	No	No	Acc	No	Acc	$\cdots$

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	…

No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4	Acc	No	Acc	No	Acc	No	$\cdots$
M_5	No	No	Acc	Acc	No	No	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

No	No	No	Acc	No	Acc	$\cdots$

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	…

No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	…

No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4	Acc	No	Acc	No	Acc	No	$\cdots$
M_5	No	No	Acc	Acc	No	No	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

No	No	No	Acc	No	Acc	$\cdots$

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	…
	No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	…

No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	…

No	No	No	Acc	No	Acc	…

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4	Acc	No	Acc	No	Acc	No	$\cdots$
M_5	No	No	Acc	Acc	No	No	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

No	No	No	Acc	No	Acc	$\cdots$

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_0	Acc	No	No	Acc	Acc	No	…
M_1	Acc	Acc	Acc	Acc	Acc	Acc	…
M_2	Acc	Acc	Acc	Acc	Acc	Acc	…
M_3	No	Acc	Acc	No	Acc	Acc	…
M_4	Acc	No	Acc	No	Acc	No	…
M_5	No	No	Acc	Acc	No	No	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	…

No	No	No	Acc	No	Acc	…

No TM has this behavior!

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4	Acc	No	Acc	No	Acc	No	$\cdots$
M_5	No	No	Acc	Acc	No	No	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

No	No	No	Acc	No	Acc	$\cdots$

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_0	Acc	No	No	Acc	Acc	No	$\cdots$
M_1	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_2	Acc	Acc	Acc	Acc	Acc	Acc	$\cdots$
M_3	No	Acc	Acc	No	Acc	Acc	$\cdots$
M_4	Acc	No	Acc	No	Acc	No	$\cdots$
M_5	No	No	Acc	Acc	No	No	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$

No	No	No	Acc	No	Acc	$\cdots$

"The language of all TMs that do not accept their own descriptions"

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	⋯
M_0	Acc	No	No	Acc	Acc	No	⋯
M_1	Acc	Acc	Acc	Acc	Acc	Acc	⋯
M_2	Acc	Acc	Acc	Acc	Acc	Acc	⋯
M_3	No	Acc	Acc	No	Acc	Acc	⋯
M_4	Acc	No	Acc	No	Acc	No	⋯
M_5	No	No	Acc	Acc	No	No	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋯

No	No	No	Acc	No	Acc	⋯

$\{\langle M \rangle \mid M \text{ is a TM that does not accept } \langle M \rangle\}$

The *diagonalization language* L_D is defined as

$$L_D = \{\langle M \rangle \mid M \text{ is a TM and } M \text{ does not accept } \langle M \rangle\}$$

We constructed this language to be different from the language of every TM.

Therefore, $L_D \notin$ **RE**!

$$L_D = \{\langle M \rangle \mid M \text{ is a TM and } M \text{ does not accept } \langle M \rangle\}$$

THEOREM: $L_D \notin$ **RE.**

PROOF: By contradiction; assume that $L_D \in$ **RE**. This means that there is a Turing machine R such that $L(R) = L_D$.

Now, focus on what happens if we run the TM R on its own encoding, $\langle R \rangle$. Since R is a recognizer for L_D, we see that

R accepts $\langle R \rangle$ if and only if $\langle R \rangle \in L_D$.

By the definition of L_D, we know that

$\langle R \rangle \in L_D$ if and only if R does not accept $\langle R \rangle$.

Combining the two statements above tell us that

R accepts $\langle R \rangle$ if and only if R does not accept $\langle R \rangle$.

This is impossible. We've reached a contradiction, so our assumption was wrong and $L_D \notin$ **RE**. ■

A_{TM}

$HALT_{TM}$

Regular languages

CFLs

R

RE

All languages

Many more problems are also *unrecognizable*, including the secure voting machine problem we already proved was undecidable!

What this means

On a deeper philosophical level, the fact that non-**RE** languages exist supports the following claim:

There are statements that are true but not provable.

Intuitively, given any non-**RE** language, there will be some string in the language that *cannot* be proven to be in the language.

This result can be formalized as a result called *Gödel's incompleteness theorem*, one of the most important mathematical results of all time.

Where we stand

The Church–Turing thesis tells us that TMs give us a mechanism for studying computation in the abstract.

Universal computers – computers as we know them – are not just a stroke of luck. The existence of the universal TM ensures that such computers exist.

Self-reference is an inherent consequence of computational power.

Undecidable problems exist partially as a consequence of the above and indicate that there are statements whose truth can't be determined by computational processes.

Unrecognizable problems are out there and can be discovered via diagonalization. They imply there are limits to mathematical proof.

Reductions let us prove a connection between problems, showing they're of the same difficulty.

Acknowledgments

This lecture incorporates material from:

David Chiang, University of Notre Dame

Keith Schwarz, Stanford University

Michael Sipser, *Introduction to the Theory of Computation*