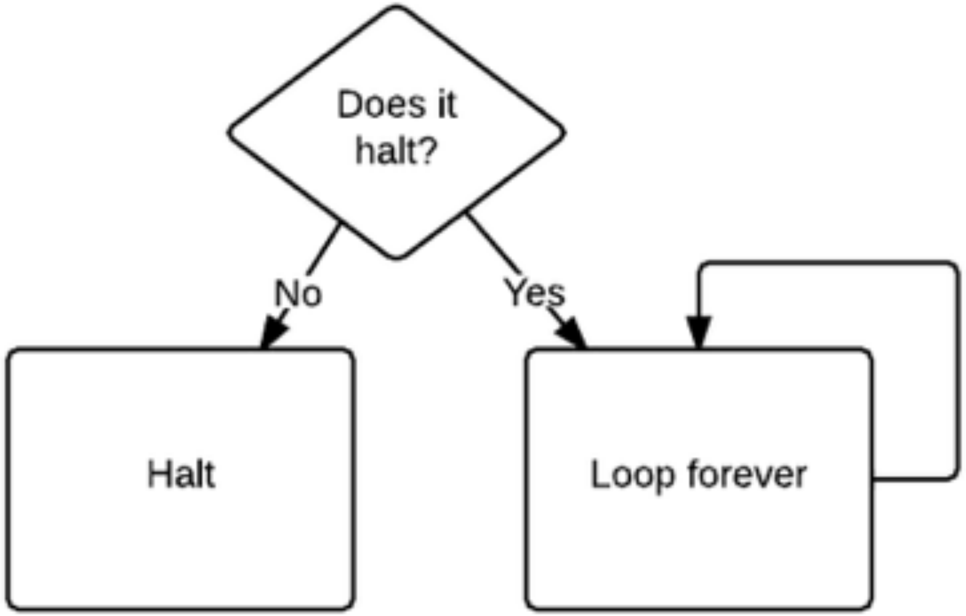
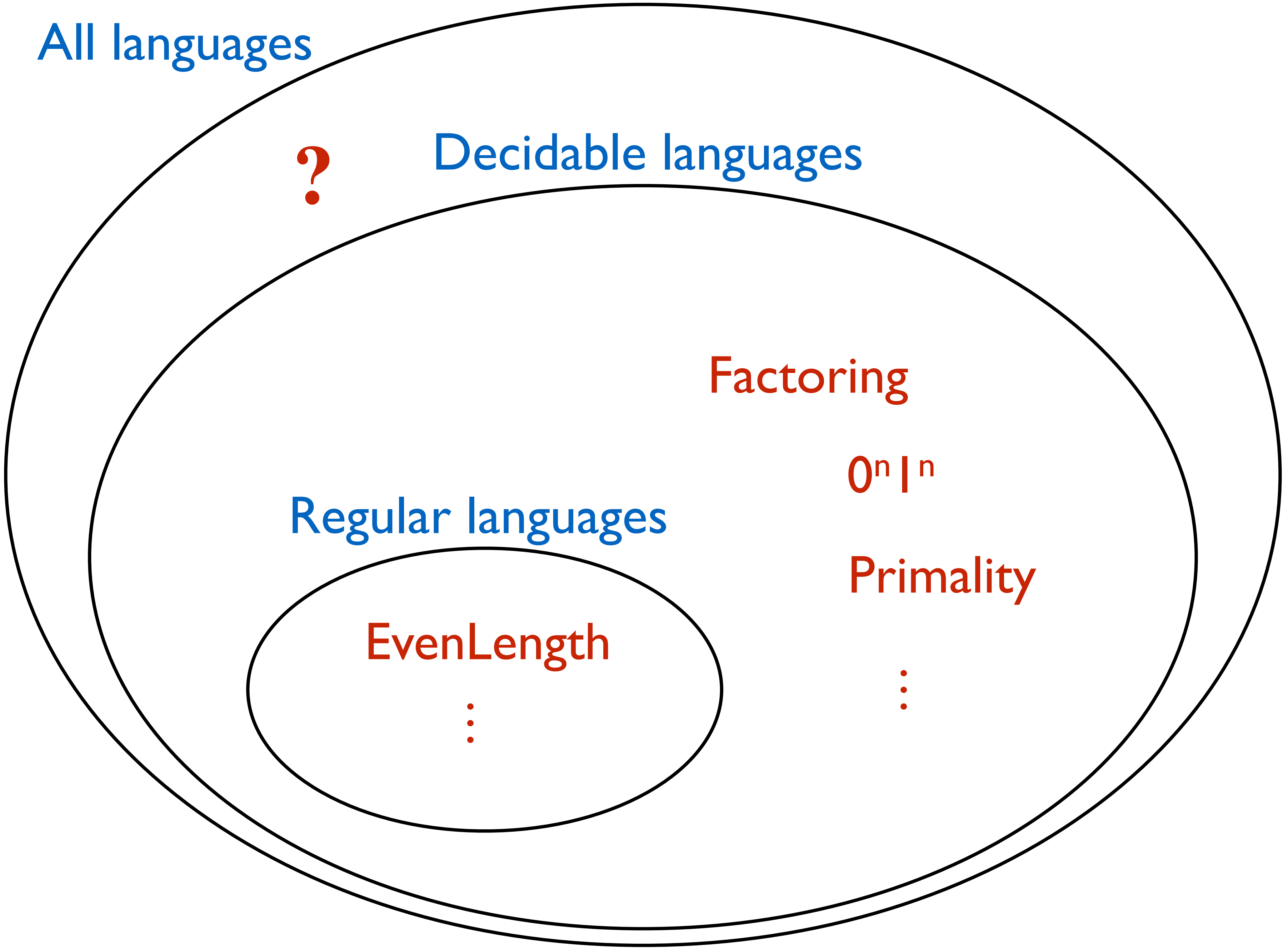
15-251
Great Theoretical Ideas in Computer Science

Lecture 6:
Turing's Legacy Continues - Undecidability



September 17th, 2015


All languages
?
Decidable languages
Factoring
$0^n 1^n$
Regular languages
Primality
EvenLength
⋮
⋮

3-Slide Review of Last Lecture

Comparing the cardinality of sets

$$|A| \leq |B|$$

$$\boxed{A \hookrightarrow B}$$

$$|A| \geq |B|$$

$$\boxed{A \twoheadrightarrow B}$$

$$|A| = |B|$$

$$\boxed{A \leftrightarrow B}$$

Definition: countable and uncountable sets

Definition:

- A set A is called *countable* if $|A| \leq |\mathbb{N}|$.
- A set A is called *countably infinite* if it is infinite and countable.
- A set A is called *uncountable* if it is **not** countable. (so $|A| > |\mathbb{N}|$)

One slide guide to countability questions

You are given a set A .

Is it countable or uncountable?

$|A| \leq |\mathbb{N}|$ or $|A| > |\mathbb{N}|$?

$|A| \leq |\mathbb{N}|$:

- show directly that $A \hookrightarrow \mathbb{N}$ or $\mathbb{N} \twoheadrightarrow A$
- show $|A| \leq |B|$, where

$$B \in \{\mathbb{Z}, \quad \mathbb{Z} \times \mathbb{Z}, \quad \mathbb{Q}, \quad \boxed{\Sigma^*}, \quad \mathbb{Q}[x]\}$$

$|A| > |\mathbb{N}|$:

- show directly using a diagonalization argument
- show $|A| \geq |\{0,1\}^\infty|$

Another thing to remember from last week

Encoding different objects with strings

Fix some alphabet Σ .

We use the $\langle \cdot \rangle$ notation to denote the encoding of an object as a string in Σ^*.

<u>**Examples:**</u>

$\langle M \rangle \in \Sigma^*$ is the encoding a TM M

$\langle D \rangle \in \Sigma^*$ is the encoding a DFA D

$\langle M_1, M_2 \rangle \in \Sigma^*$ is the encoding of a pair of TMs (M_1, M_2)

$\langle M, x \rangle \in \Sigma^*$ is the encoding a pair (M, x), where M is a TM, and $x \in \Sigma^*$.

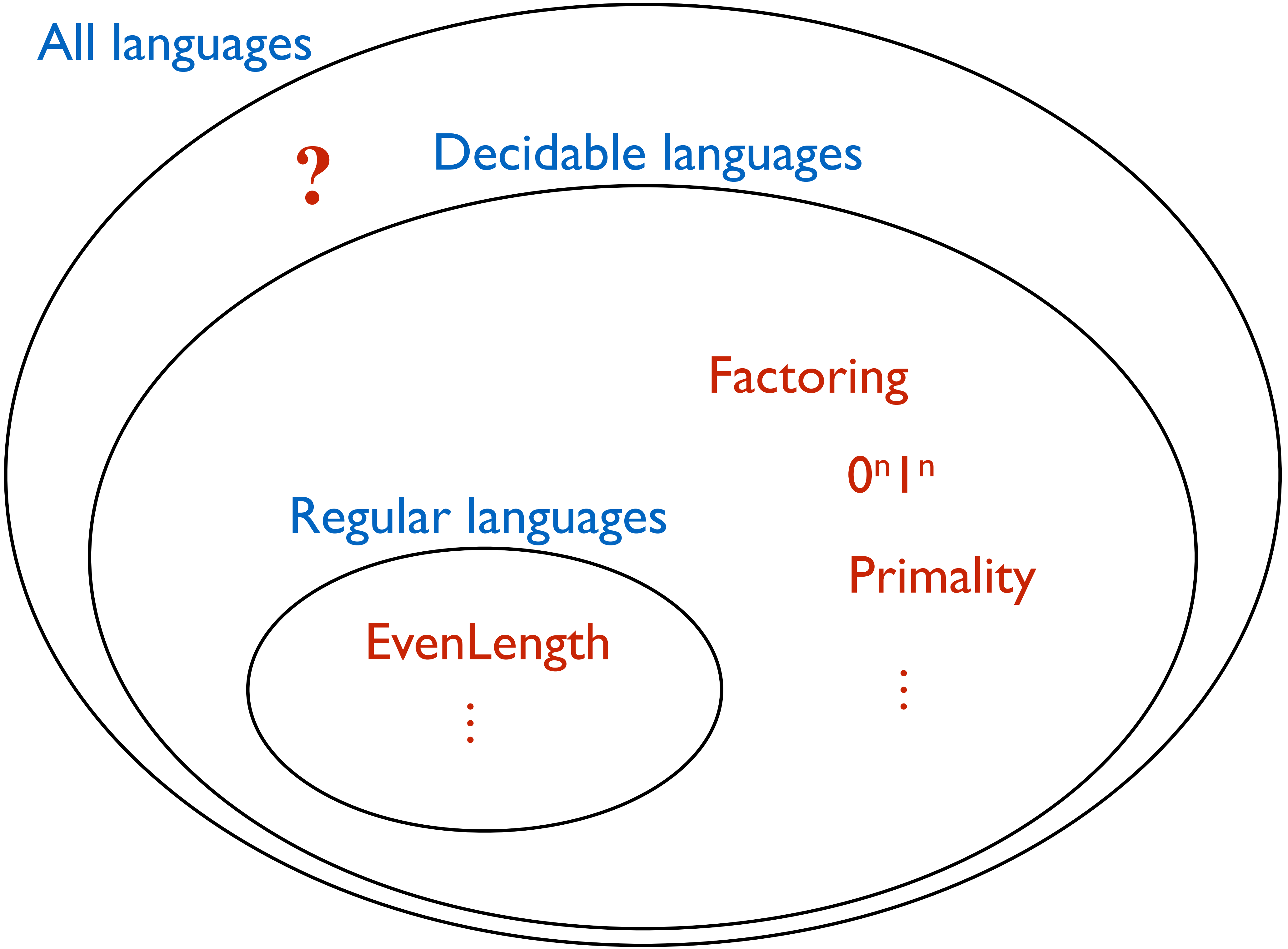
Poll

Let A be the set of all languages over $\Sigma = \{1\}$.

Select the correct ones:

- *A* is finite
- *A* is infinite
- *A* is countable
- *A* is uncountable

Applications to Computer Science


All languages
?
Decidable languages
Factoring
0ⁿ1ⁿ
Primality
Regular languages
EvenLength
⋮
⋮

Most problems are undecidable

Just count!

For any TM M, $\langle M \rangle \in \Sigma^*$.

So $\{M : M \text{ is a TM}\}$ is countable.

(the CS method)

So the set of decidable languages is countable.

How about the set of all languages?

$$\{L : L \subseteq \Sigma^*\} \;= \mathcal{P}(\Sigma^*)$$ is uncountable.

Maybe all undecidable languages are uninteresting ?

Working as a course assistant for 15-112

We need to write an autograder for
nthAwesomeHappyCarolPrime

Working as a course assistant for 15-112

We need to write an autograder for
isAwesomeHappyCarolPrime

student submission

isAwesomeHappyCarolPrime

the correct program

isAwesomeHappyCarolPrime

Do they accept and reject exactly the same inputs?

Working as a course assistant for 15-112

We need to write an autograder for isAwesomeHappyCarolPrime

Kosbie's version

Student submission

Accepts and rejects same strings?

True
or
False

Working as a course assistant for 15-112

We need to write an autograder for isAwesomeHappyCarolPrime

Koz, I can't figure it out.

Working as a course assistant for 15-112

Fine.
Just write an autograder
that checks if a given program
goes into an infinite loop.

Hmm.
This seems hard too.
Let me ask Prof. Procaccia

An explicit undecidable language

This is called the **halting problem.**

Theorem: The **halting problem** is undecidable.

Proof by Python

Halting Problem

Inputs: A Python program source code. 
An *input* to the program. x

Outputs: True if the program halts for the given *input*.
False otherwise.

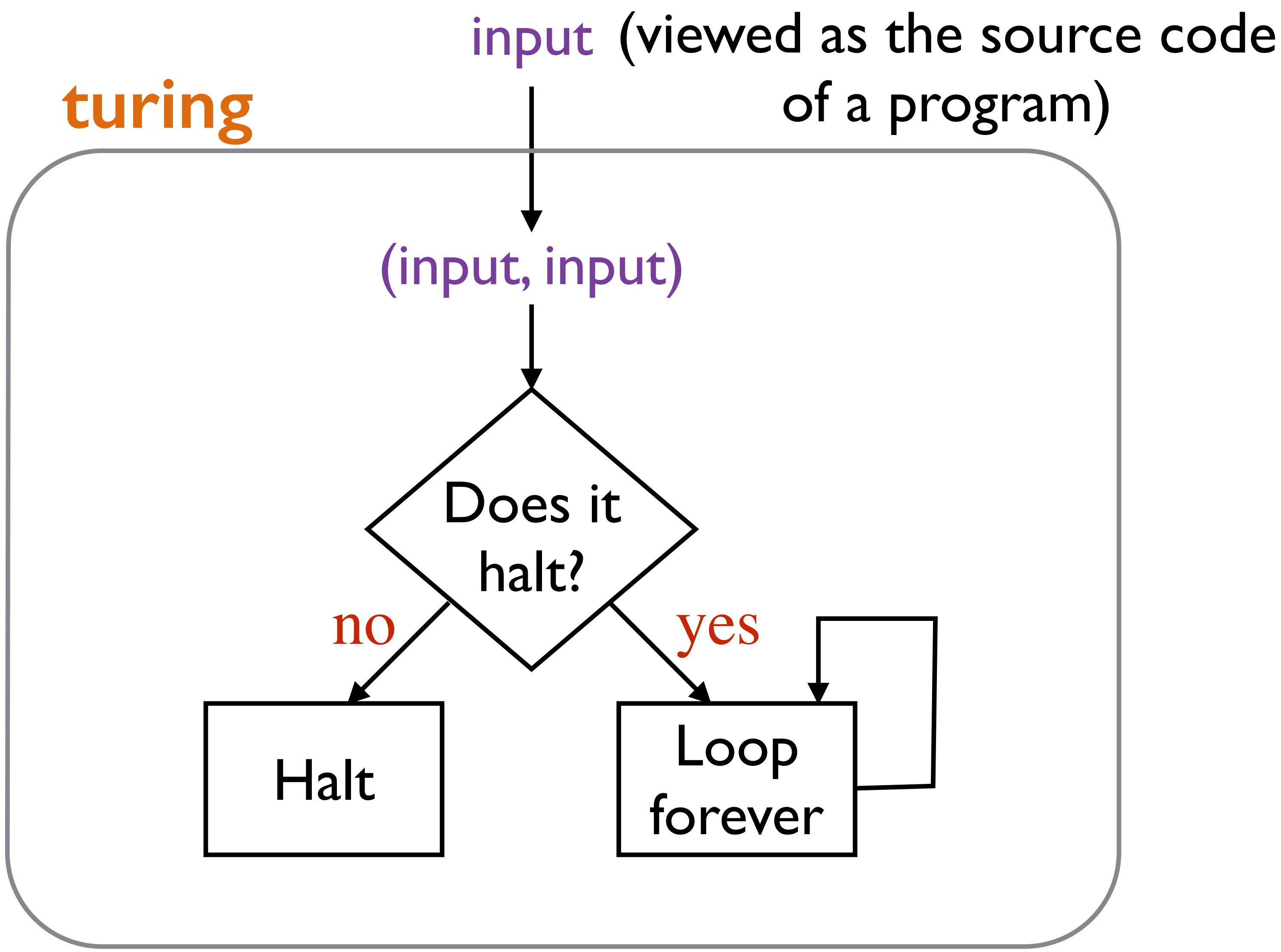
Proof by Python

Assume such a program exists:

```
def halt(program, inputToProgram):
    # program and inputToProgram are both strings
    # Returns True if program halts when run with inputToProgram
    # as its input.

def turing(program):
    if (halt(program, program)):
        while True:
            pass   # i.e. do nothing
    return None
```

Proof by Python



Proof by Python

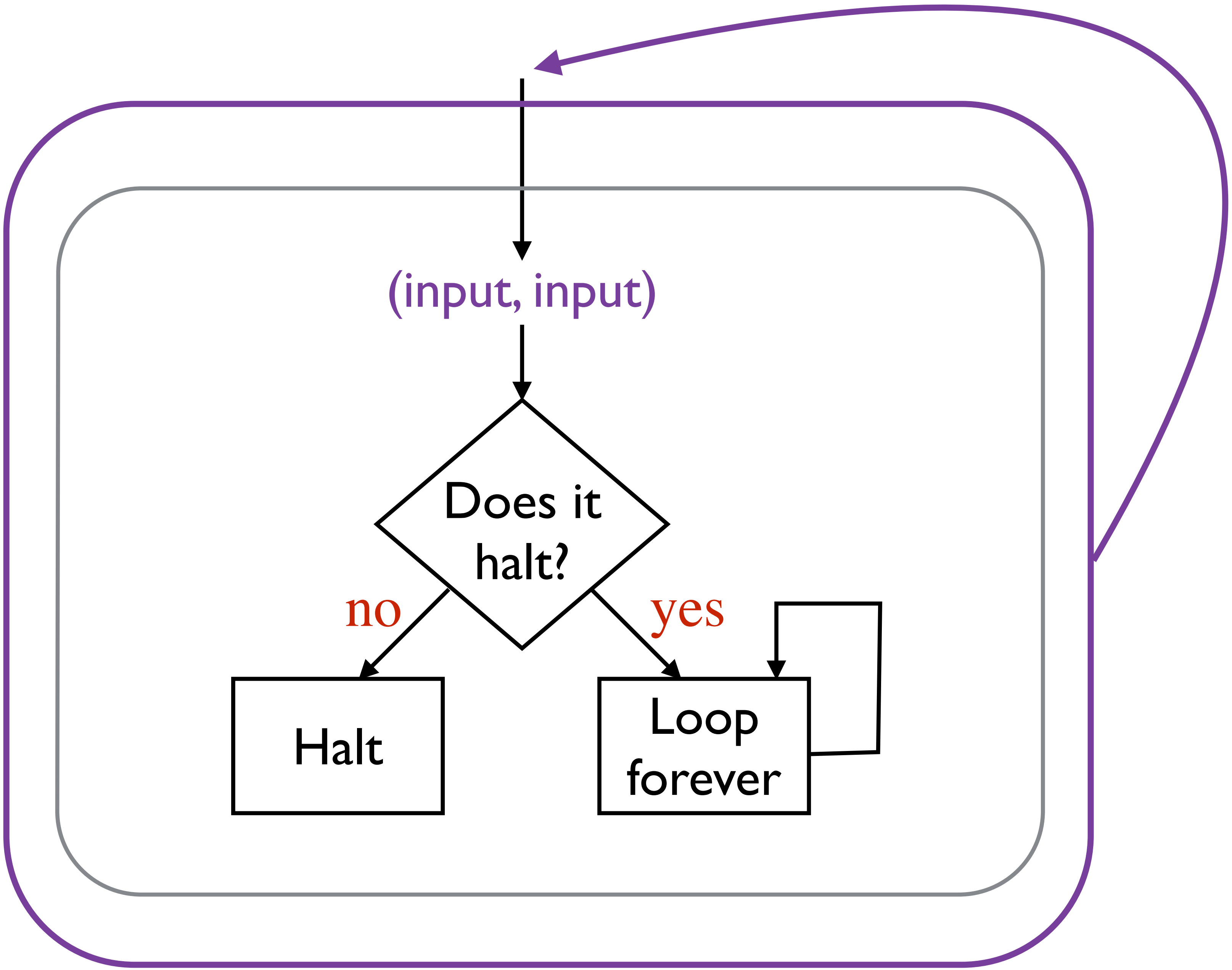
Assume such a program exists:

```
def halt(program, inputToProgram):
    # program and inputToProgram are both strings
    # Returns True if program halts when run with inputToProgram
    # as its input.

def turing(program):
    if (halt(program, program)):
        while True:
            pass   # i.e. do nothing
    return None
```

What happens when you call **turing**(*turing*) ?

Proof by Python



Proof by Python

Assume such a program exists:

```
def halt(program, inputToProgram):
    # program and inputToProgram are both strings
    # Returns True if program halts when run with inputToProgram
    # as its input.

def turing(program):
    if (halt(program, program)):
        while True:
            pass   # i.e. do nothing
    return None
```

What happens when you call **turing**(*turing*) ?

if **halt**(*turing*, *turing*) ----> **turing** doesn't halt

if **not** **halt**(*turing*, *turing*) ----> **turing** halts $\square$

That was a diagonalization argument

```
def turing(program):
    if (halt(program, program)):
        while True:
            pass   # i.e. do nothing
    return None
```

$$
\begin{array}{c|cccccc}
 & \langle f_1 \rangle & \langle f_2 \rangle & \langle f_3 \rangle & \langle f_4 \rangle & \cdots \\
\hline
f_1 & \boxed{\infty} & \infty & H & \infty & \\
f_2 & H & \boxed{H} & H & \infty & \\
f_3 & \infty & \infty & \boxed{H} & H & \cdots \\
f_4 & \infty & H & H & \boxed{\infty} & \\
\vdots & & \vdots & & & \\
\text{turing} & H & \infty & \infty & H & \cdots
\end{array}
$$

Halting problem is undecidable

Proof by a theoretical computer scientist:

$\text{HALT} = \{\langle M, x\rangle : M \text{ is a TM and it halts on input } x\}$

Suppose M_{HALT} decides $\text{HALT}.$

Consider the following TM (let's call it M_{TURING}):

M_{TURING}

Treat the input as $\langle M\rangle$ for some TM M .

Run M_{HALT} with input $\langle M, M\rangle$.

If it accepts, go into an infinite loop.

If it rejects, accept (i.e. halt).

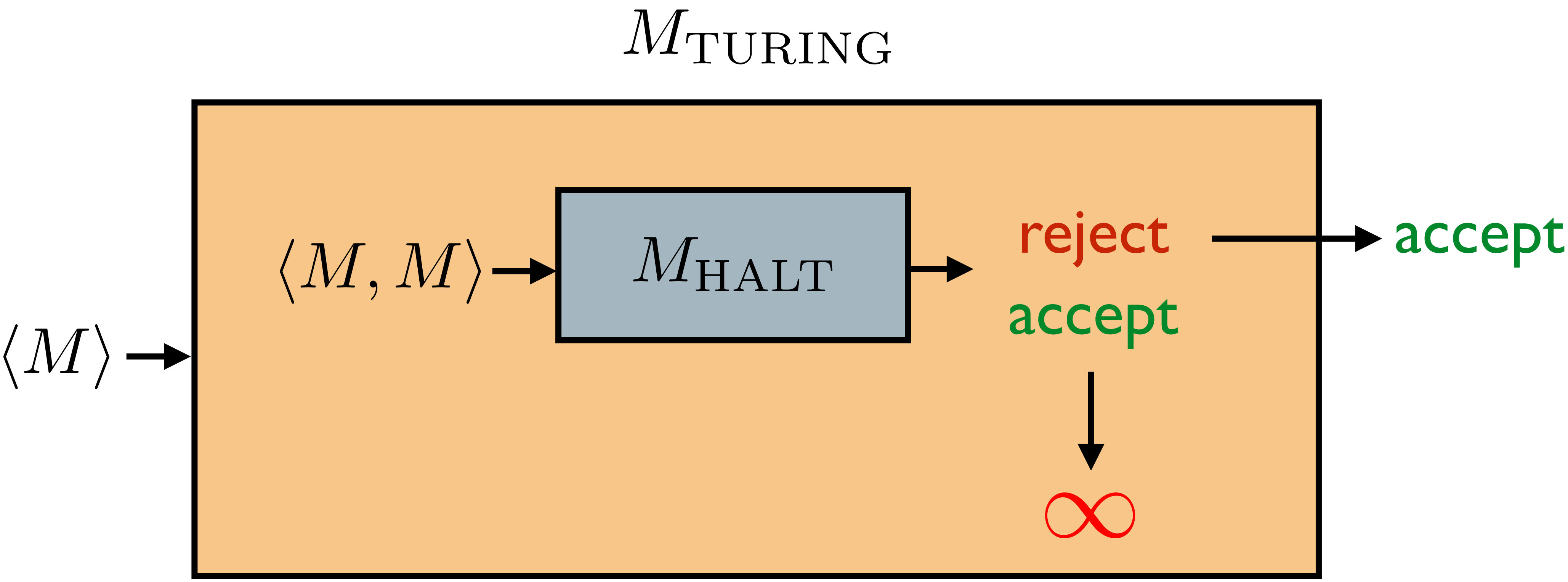
Halting problem is uncomputable

Proof by a theoretical computer scientist:

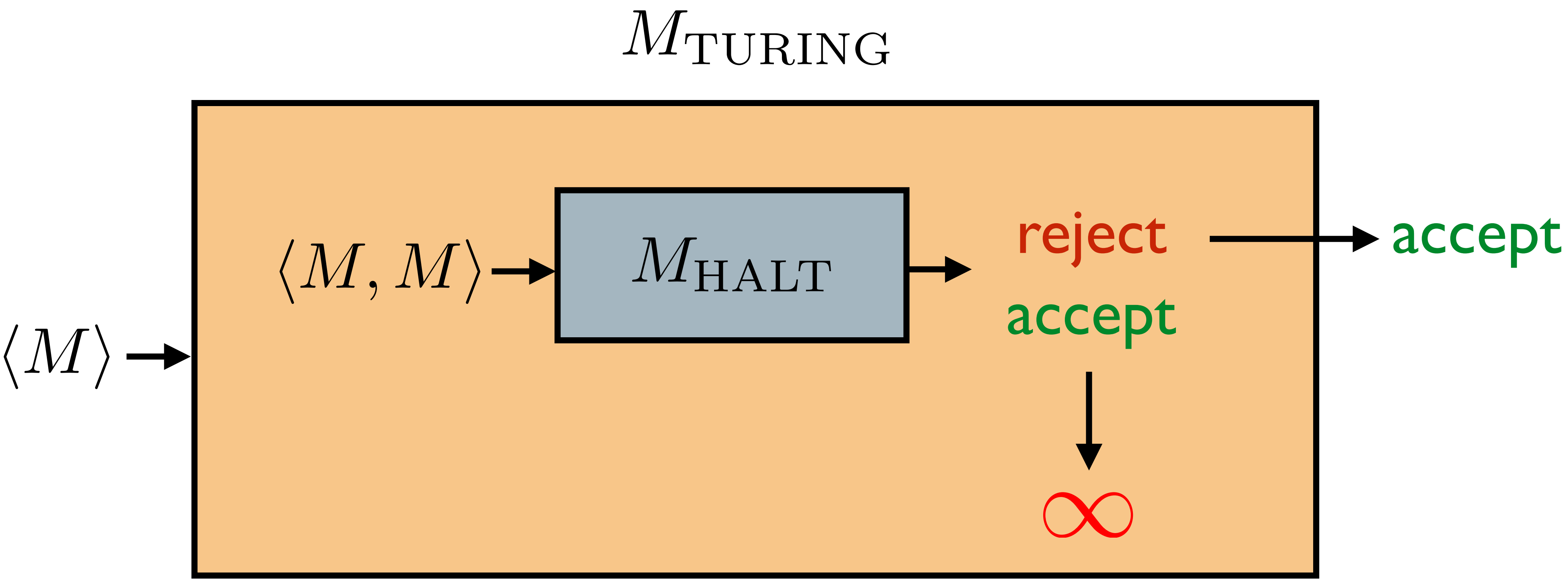
$$\text{HALT} = \{\langle M, x\rangle : M \text{ is a TM and it halts on input } x\}$$

Suppose M_{HALT} decides HALT.

Consider the following TM (let's call it M_{TURING}):



Halting problem is uncomputable



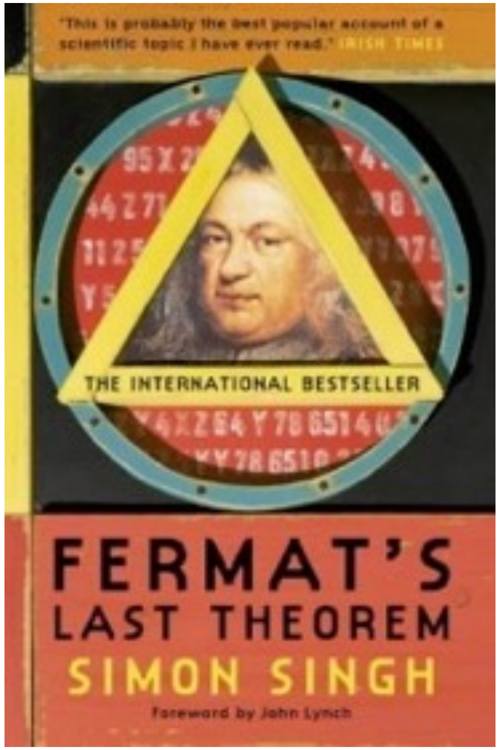
What happens when $\langle M_{\text{TURING}} \rangle$ is input to M_{TURING}?

So what?

- No guaranteed autograder program.

- Consider the following program:

```
def fermat():
    t = 3
    while (True):
        for n in xrange(3, t+1):
            for x in xrange(1, t+1):
                for y in xrange(1, t+1):
                    for z in xrange(1, t+1):
                        if (x**n + y**n == z**n): return (x, y, z, n)
        t += 1
```



<u>Question</u>: Does this program halt?

So what?

- Consider the following program (written in MAPLE):

```
numberToTest := 2;
flag := 1;
while flag = 1 do
    flag := 0;
    numberToTest := numberToTest + 2;
    for p from 2 to numberToTest do
        if IsPrime(p) and IsPrime(numberToTest−p) then
            flag := 1;
            break;      #exits the for loop
        end if
    end for
end do
```

Goldbach Conjecture

<u>**Question**</u>: Does this program halt?

So what?

- Reductions to other problems imply that those problems are undecidable as well.

Entscheidungsproblem

Is there a finitary procedure to determine the validity of a given logical expression?

e.g. $\neg \exists x, y, z, n \in \mathbb{N} : (n \geq 3) \wedge (x^n + y^n = z^n)$

(Mechanization of mathematics)

Hilbert's 10th Problem

Is there a program to determine if a given multivariate polynomial with integral coefficients has an integral solution?

So what?

Different laws of physics ----->

Different computational devices ----->

Every problem computable (?)

Can you come up with sensible laws of physics
such that the Halting Problem becomes computable?

Is there a way to show other languages are undecidable?

Reductions

A central concept used to compare the "difficulty" of languages/problems.

will differ based on context

Now we are interested in decidability vs undecidability
(computability vs uncomputability)

Let A and B be two languages.

Want to define: $A \leq B$ to mean

B is at least as hard as A (with respect to decidability).

i.e., B decidable $\Longrightarrow$ A decidable

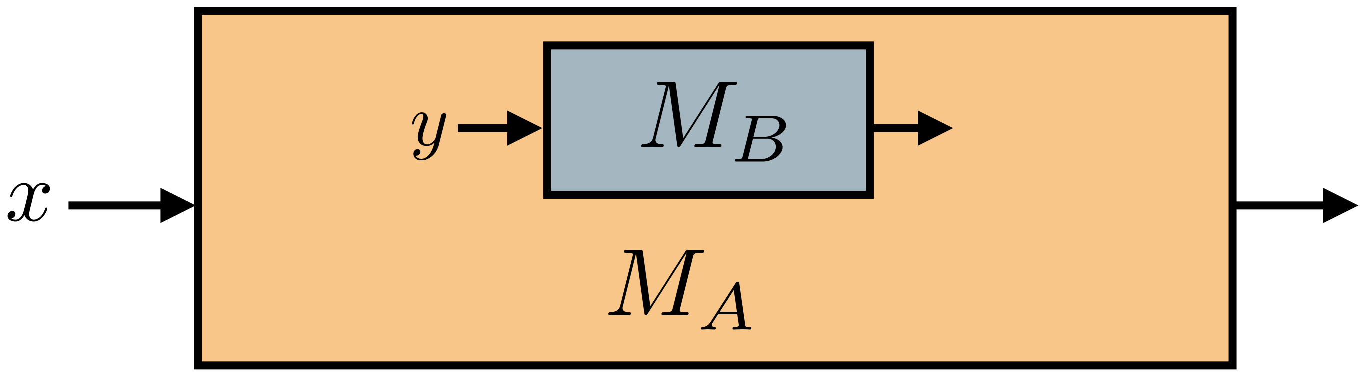
A undecidable $\Longrightarrow$ B undecidable

Reductions

Definition: Let A and B be two languages.

$A \leq_T B$ (A *reduces* to B)

if it is possible to decide A
using a TM that decides B as a subroutine.



To show $A \leq_T B$:

- <u>assume</u> the existence of M_B
- construct M_A that uses M_B as a subroutine.

you want to specify
the orange part

Reductions

```
def fooB(input):
    # assume some code exists
    # that solves the problem B

def fooA(input):
    # some code that solves the problem A
    # that makes calls to function fooB when needed
```

To show $A \leq_T B$:

Give me the code for fooA.

So to show a reduction, you give an algorithm.

Reduction example

A: Given a sequence of integers, and a number k,
is there an increasing subsequence of length at least k?

3, 1, 5, 2, 3, 6, 4, 8

B: Given two sequences of integers, and a number k,
is there a common inc. subsequence of length at least k?

3, 1, 5, 2, 3, 6, 4, 8

1, 5, 7, 9, 2, 4, 1, 0, 2, 0, 3, 0, 4, 0, 8

A reduces to **B**

Give me an algorithm to solve **A**
assuming an algorithm for **B** is given for free.

Reduction example

```
def fooB(seq1, seq2, k):
    # assume some code exists
    # that solves the problem B


def fooA(seq, k):
    return fooB(seq, sorted(seq), k)
```

3, 1, 5, 2, 3, 6, 4, 8

1, 2, 3, 4, 5, 6, 7, 8

Reductions

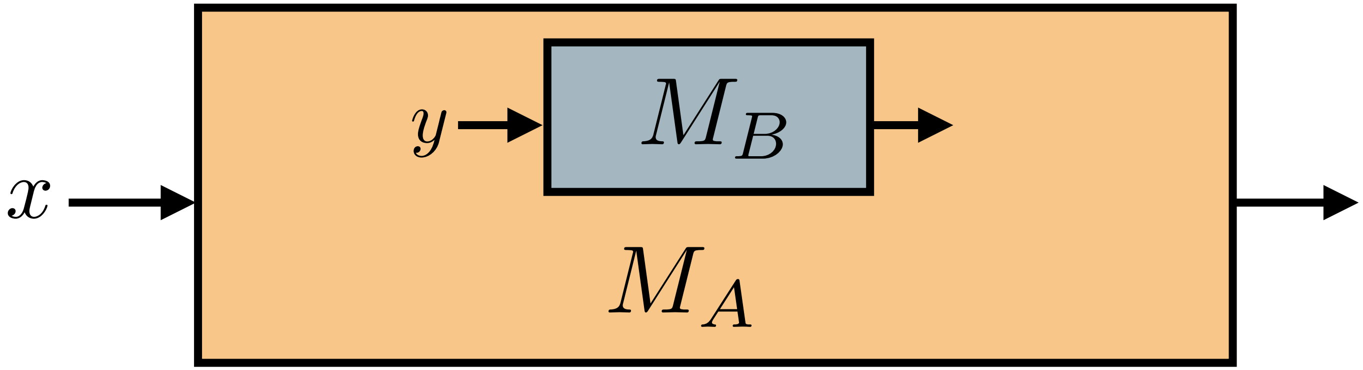
Wanted to define: $A \leq B$ to mean

B is at least as hard as A (with respect to decidability).

i.e., B decidable $\Longrightarrow$ A decidable

A undecidable $\Longrightarrow$ B undecidable

If $A \leq_T B$ (A *reduces* to B):



B decidable $\Longrightarrow$ A decidable

A undecidable $\Longrightarrow$ B undecidable

Reductions

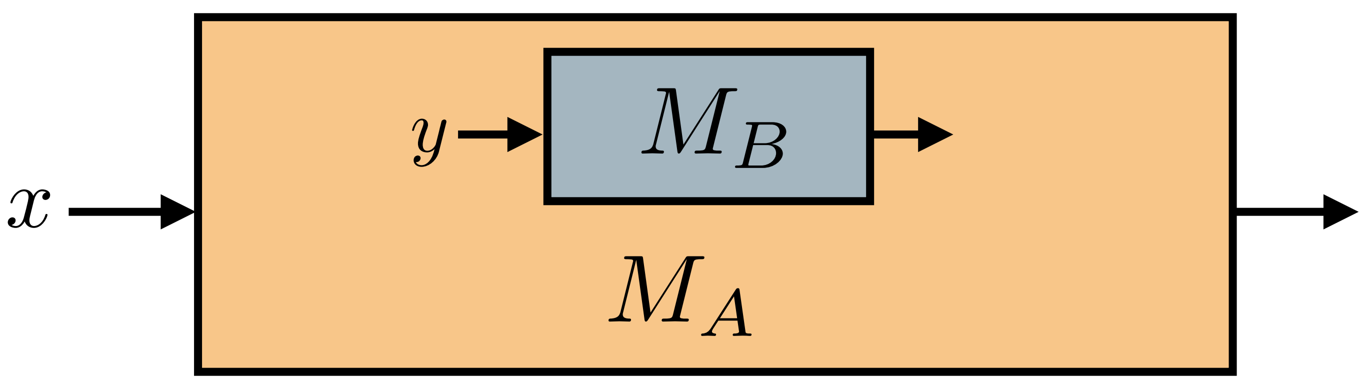
<u>Wanted to define:</u> $A \leq B$ to mean

B is at least as hard as A (with respect to decidability).

i.e., B decidable $\implies$ A decidable

A undecidable $\implies$ B undecidable

If $A \leq_T B$ (A *reduces* to B):



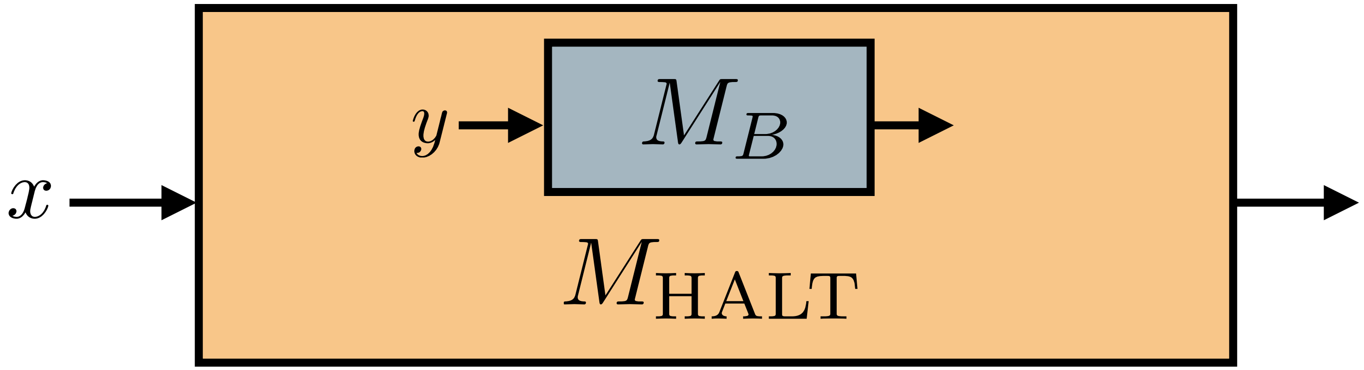
"The task of solving A *reduces* to the task of solving B ."

Reductions

We know HALT is undecidable.

If $\mathrm{HALT} \leq_T B$

B is undecidable!



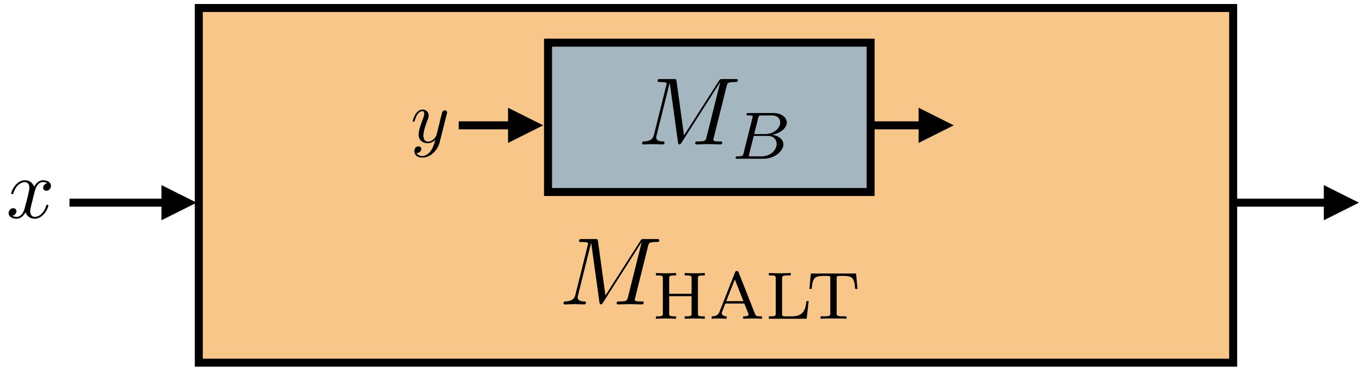
(You want to come up with an algorithm that solves the HALTING problem, assuming M_B exists.)

Reductions

We know HALT is undecidable.

If $\mathrm{HALT} \leq_T B$

B is undecidable!



To show B is undecidable, i.e. M_B cannot exist:

- assume it does exist
- then show how to decide HALT

Proving other languages are undecidable via reductions

Example 1: ACCEPTS

Theorem:

$\text{ACCEPTS} = \{\langle M, x \rangle : M \text{ is a TM that accepts } x\}$ is **undecidable**.

$\langle M, x \rangle$ is in the language $\Longrightarrow$

x leads to an accept state in M.

$\langle M, x \rangle$ is not in the language $\Longrightarrow$

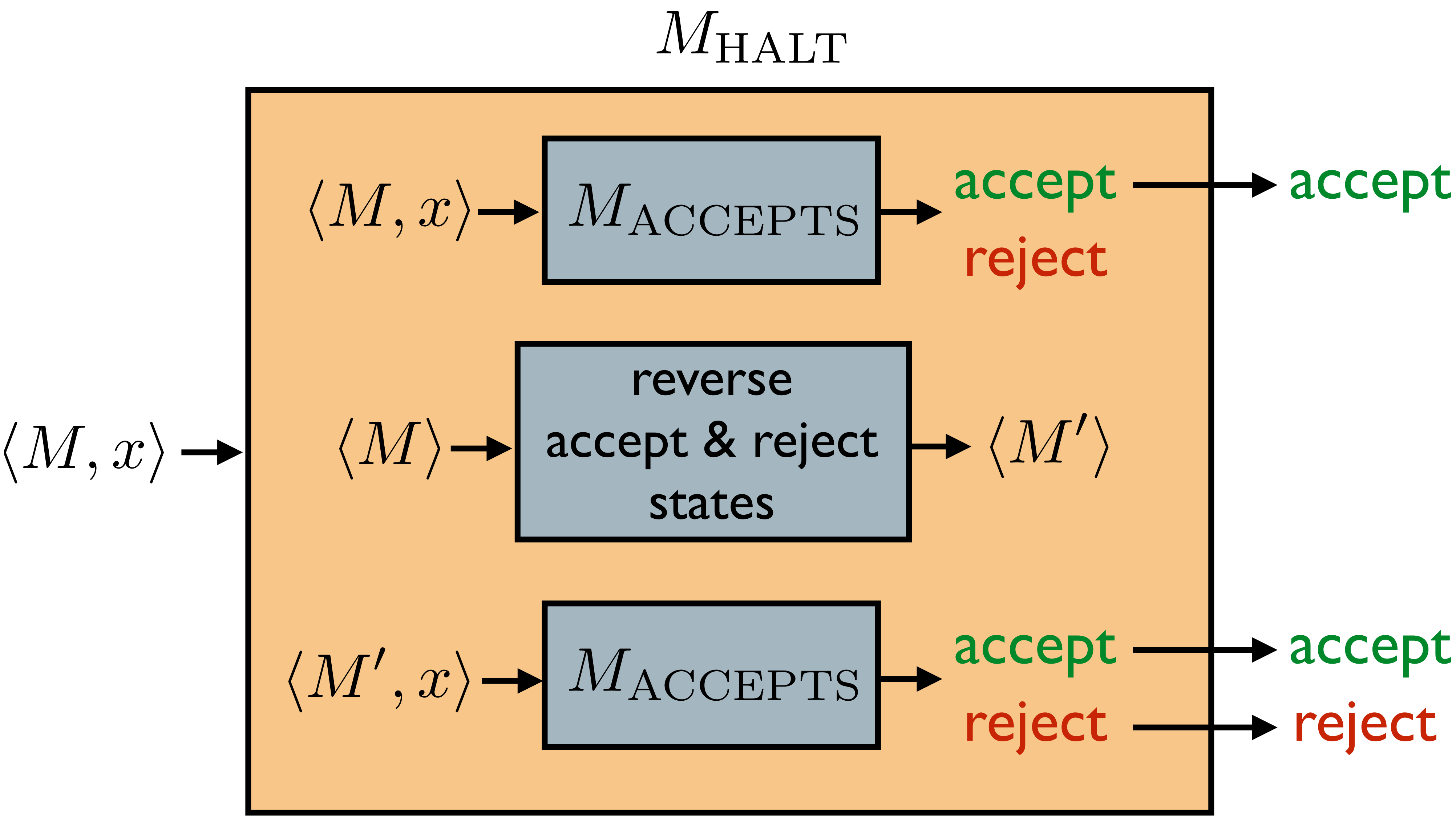
x leads to a reject state, **or** M loops forever.

$\langle M, x \rangle \in \text{HALT}$ iff x leads to an accept or reject state.

Example 1: ACCEPTS

$\text{ACCEPTS} = \{\langle M, x \rangle : M \text{ is a TM that accepts } x\}$

Proof: (by picture)



Example 1: ACCEPTS

$\text{ACCEPTS} = \{\langle M, x\rangle : M \text{ is a TM that accepts } x\}$

Proof:

We will show $\text{HALT} \leq_T \text{ACCEPTS}$.

Let M_{ACCEPTS} be a TM that decides ACCEPTS.

Here is a TM that decides HALT:

On input $\langle M, x\rangle$, run $M_{\text{ACCEPTS}}(\langle M, x\rangle)$.

If it accepts, accept.

Reverse the accept and rejects states of M. Call it M'.

Run $M_{\text{ACCEPTS}}(\langle M', x\rangle)$.

If it accepts (M rejects x), accept.

Reject.

Example 1: ACCEPTS

$\text{ACCEPTS} = \{\langle M, x \rangle : M \text{ is a TM that accepts } x\}$

Proof:

Argue that if $\langle M, x \rangle \in \text{HALT}$

the machine accepts it.

And if $\langle M, x \rangle \notin \text{HALT}$

the machine rejects it.

$\square$

Interesting Observation

To show a negative result (that there is no algorithm)

we are showing a positive result (that there is a reduction)

Example 2: EMPTY

Theorem:

$\text{EMPTY} = \{\langle M \rangle : M \text{ is a TM that accepts no strings}\}$

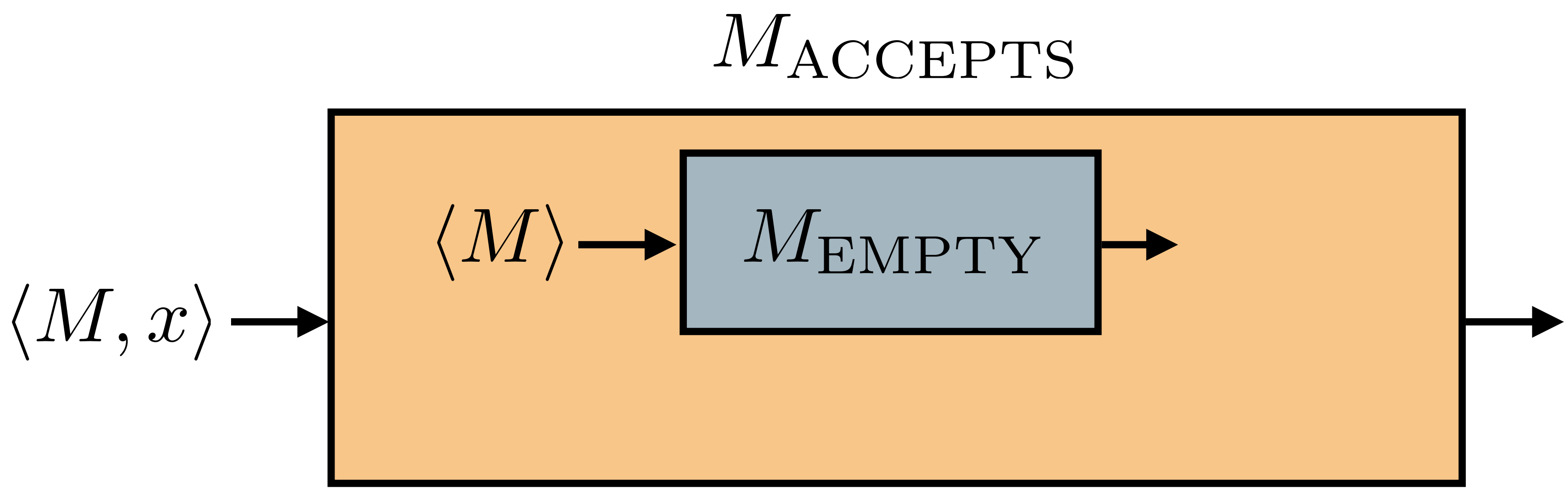
is **undecidable**.

Suffices to show $\text{ACCEPTS} \leq_T \text{EMPTY}$

Example 2: EMPTY

$$\text{EMPTY} = \{\langle M \rangle : M \text{ is a TM that accepts no strings}\}$$

$$\text{ACCEPTS} = \{\langle M, x \rangle : M \text{ is a TM that accepts } x\}$$

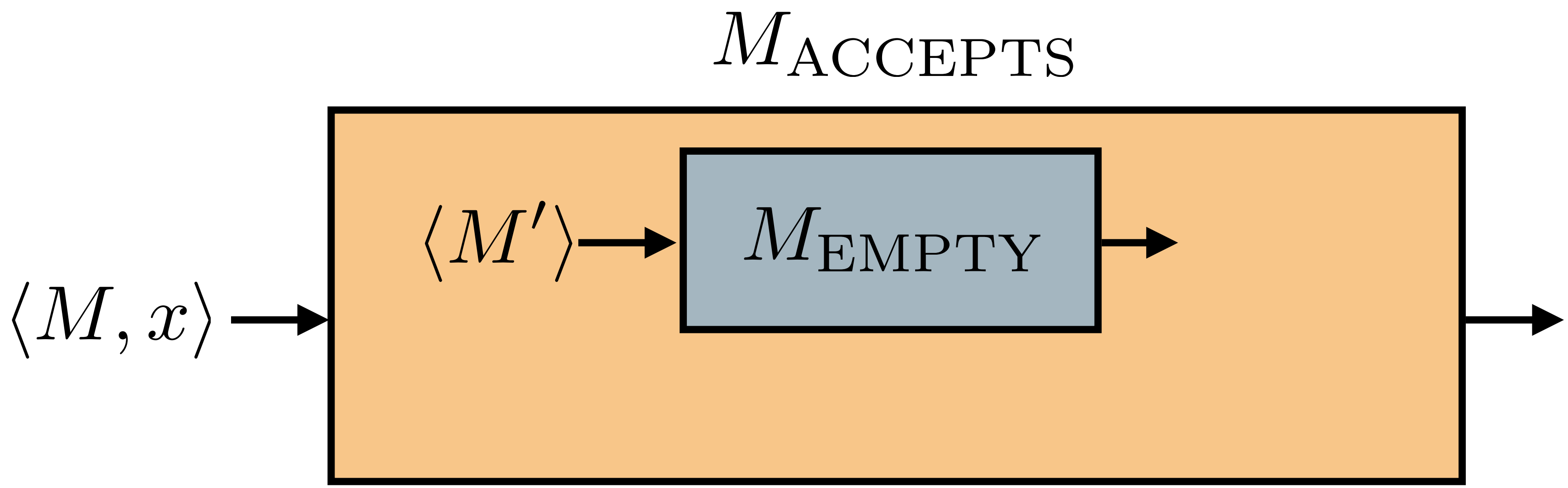


If we feed $\langle M \rangle$ into M_{EMPTY}, won't quite work.

if $M_{\text{EMPTY}}(\langle M \rangle)$ accepts, we can reject

if $M_{\text{EMPTY}}(\langle M \rangle)$ rejects , we don't know

Example 2: EMPTY



We want M' such that:

if $M_{\text{EMPTY}}(\langle M' \rangle)$ accepts, we reject

if $M_{\text{EMPTY}}(\langle M' \rangle)$ rejects , we accept

Construct M's.t.: if M accepts x , M' <u>only</u> accepts x .

if M rejects x , M' rejects everything.

Example 2: EMPTY

```
def M_ACCEPTS(< M, x >):

    def M'(y):
        if(y != x): reject
        run M(y)
        if it accepts, accept
        if it rejects, reject

    run M_EMPTY(< M' >)
        if it accepts, reject
        if it rejects, accept
```

Creating an input that will be fed into M_EMPTY

It depends on the inputs M and x.

maybe a better name for M' is M_x.

Note: M_ACCEPTS defines M', it does not run it!

Example 2: EMPTY

def M_ACCEPTS(< M, x >):

def M'(y):

if(y != x): **reject**

run M(y)

if it accepts, **accept**

if it rejects, **reject**

run M_EMPTY(< M' >)

if it accepts, **reject**

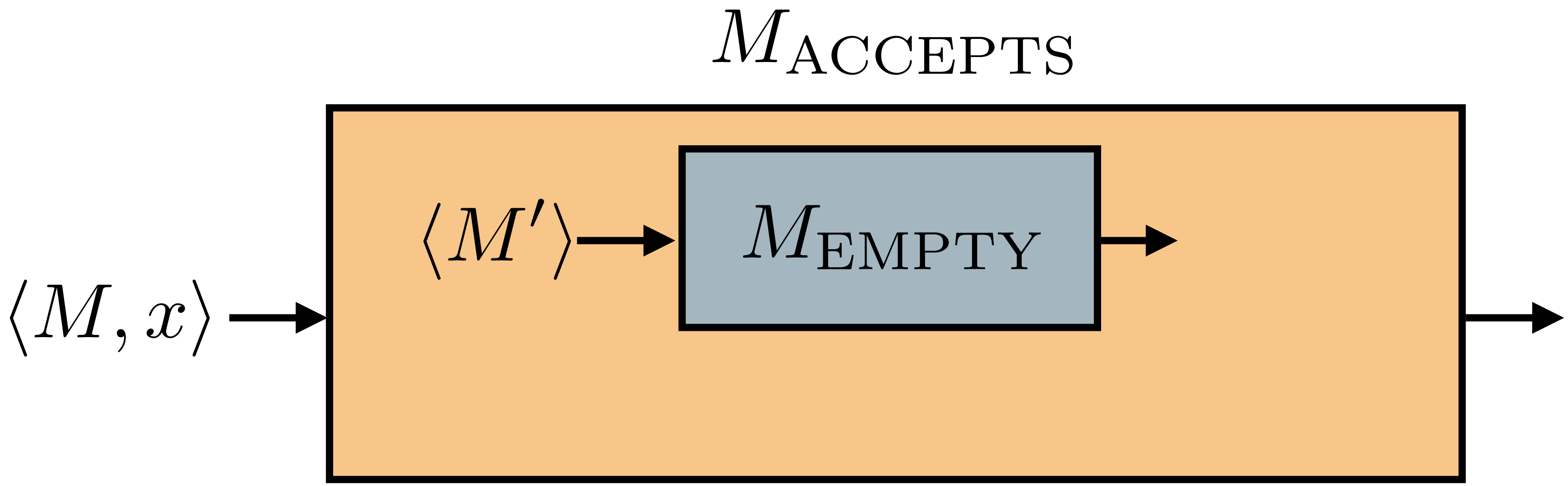
if it rejects, **accept**

If M accepts x:

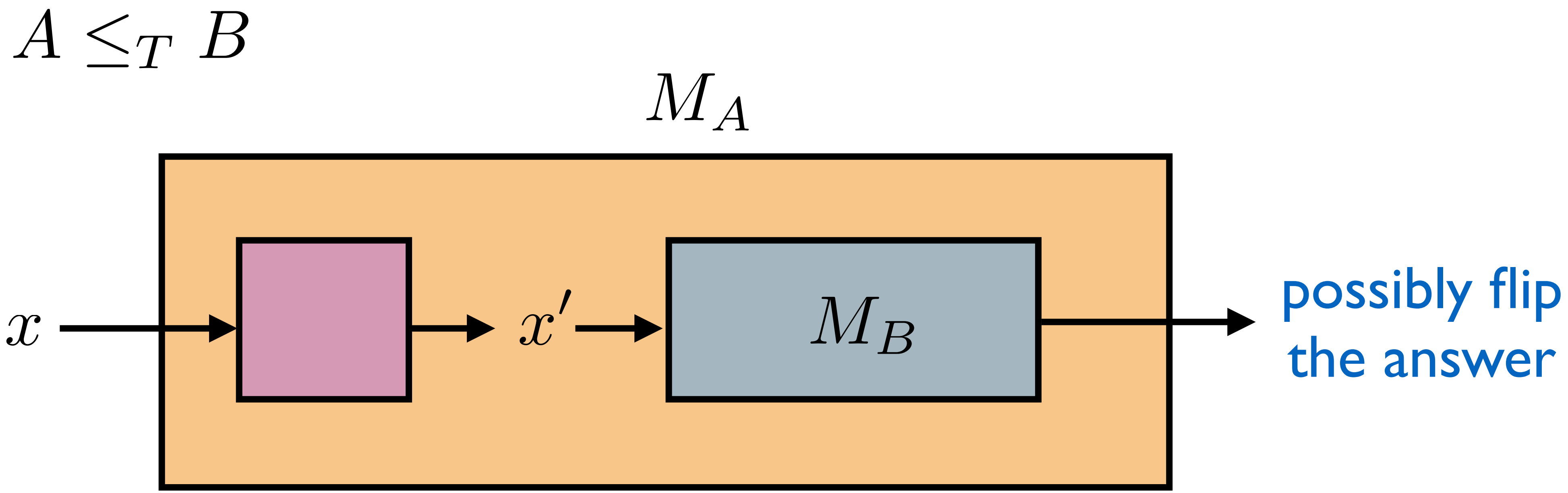
$$L(M') = \{x\}$$

If M rejects x:

$$L(M') = \emptyset$$



This structure is very common



Example 3: EQ

Theorem:

$\text{EQ} = \{\langle M_1, M_2 \rangle : M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

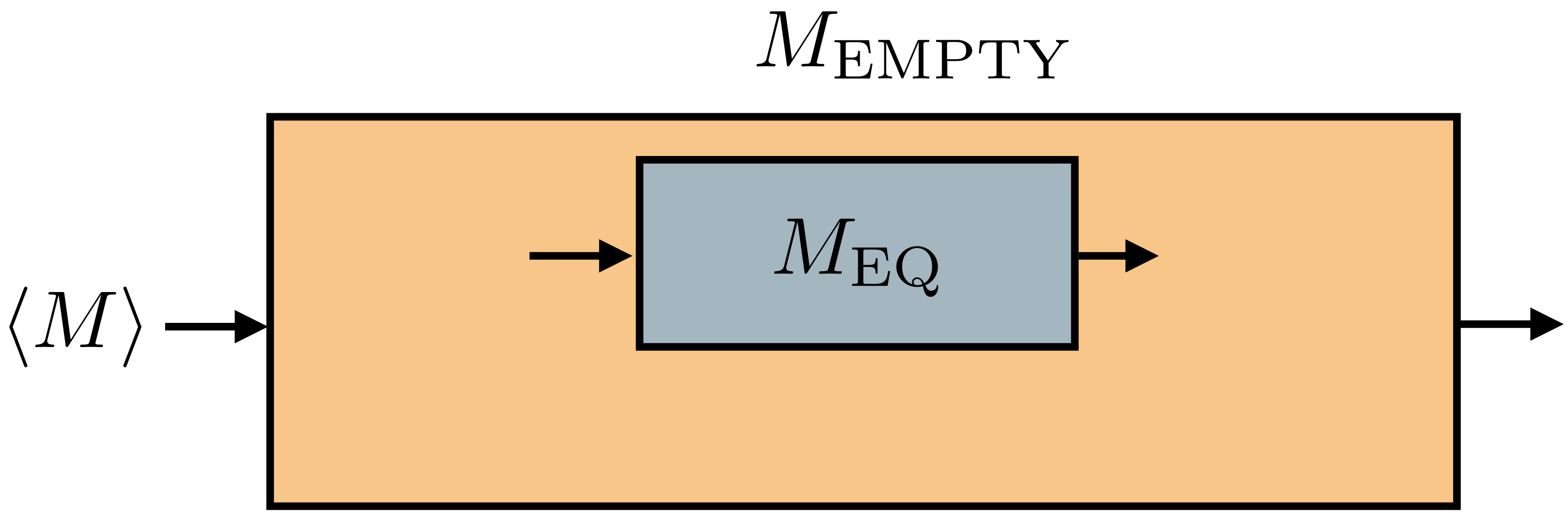
is **undecidable**.

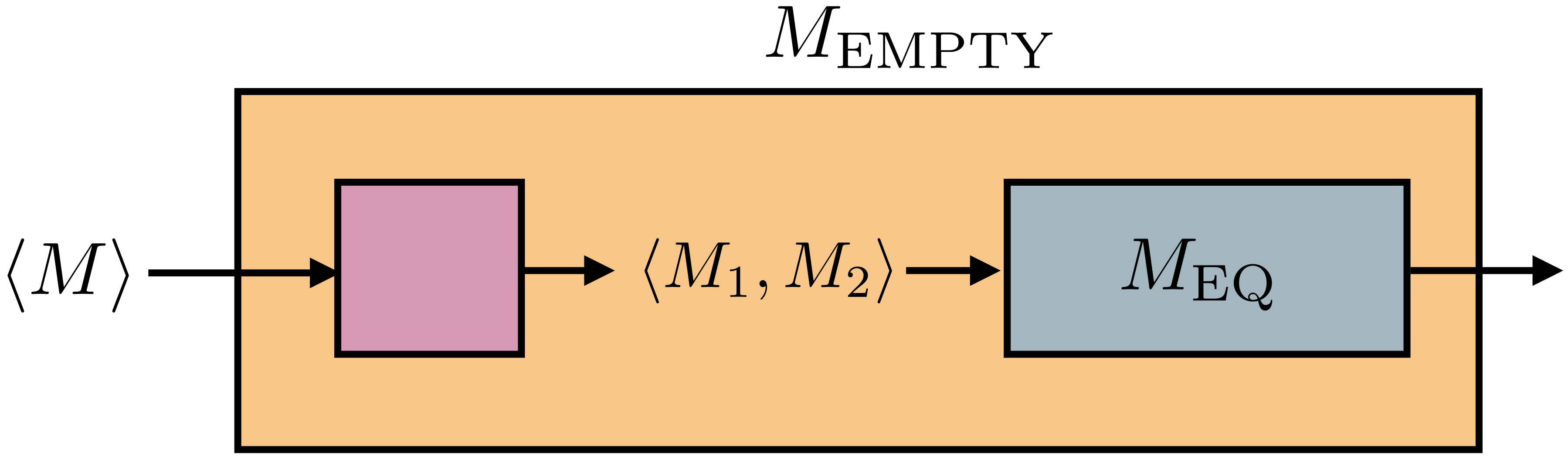
Suffices to show $\text{EMPTY} \leq_T \text{EQ}$

Example 3: EQ

$\mathrm{EQ} = \{\langle M_1, M_2\rangle : M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$

$\mathrm{EMPTY} = \{\langle M\rangle : M \text{ is a TM that accepts no strings}\}$

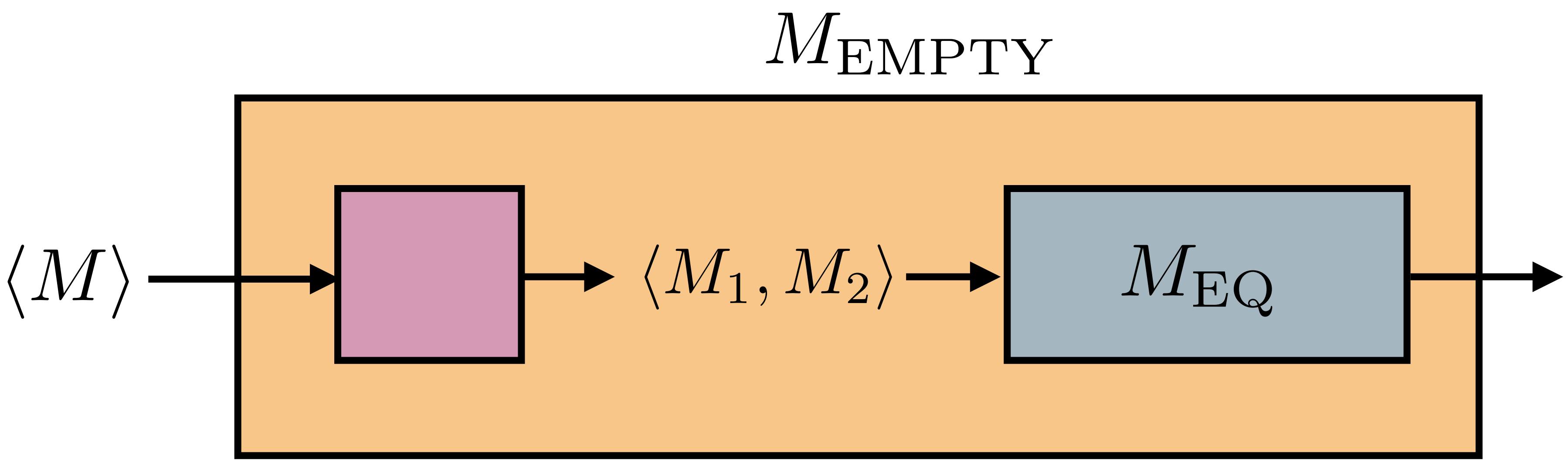




Example 3: EQ

$$\mathrm{EQ} = \{\langle M_1, M_2\rangle : M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

$$\mathrm{EMPTY} = \{\langle M\rangle : M \text{ is a TM that accepts no strings}\}$$



Let $M_1 = M$

Let M_2 be the TM that rejects everything, i.e. $L(M_2) = \emptyset$

Example 3: EQ

def M_EMPTY(< M >):

def M'(y):
reject

$$L(M') = \emptyset$$

run M_EQ(< M, M' >)
if it accepts, **accept**
if it rejects, **reject**

$$\mathrm{HALT} \leq_T \mathrm{ACCEPTS} \leq_T \mathrm{EMPTY} \leq_T \mathrm{EQ}$$

HALT reduces to EMPTY

def M_HALT(< M, x >):

> **def** M’(y):
>
> **run** M(x)
> **accept**

run M_EMPTY(< M’ >)
if it accepts, **reject**
if it rejects, **accept**

If M halts on x:

$$L(M') = \Sigma^*$$

If M does not halt on x:

$$L(M') = \emptyset$$

HALT reduces to EQ

def M_HALT(< M, x >):

def M'(y):

reject

def M''(y):

run M(x)

accept

run M_EQ(< M', M'' >)

if it accepts, **reject**

if it rejects, **accept**

$$L(M') = \emptyset$$

If M halts on x:

$$L(M'') = \Sigma^*$$

If M does not halt on x:

$$L(M'') = \emptyset$$

Undecidable problems not involving Turing Machines

Entscheidungsproblem

Determining the validity of a given FOL sentence.

e.g. $\neg \exists x, y, z, n \in \mathbb{N} : (n \geq 3) \wedge (x^n + y^n = z^n)$

Undecidable!

Proved in 1936 by Turing.

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable numbers, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In § 8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel†. These results

† Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I", *Monatshefte Math. Phys.*, 38 (1931), 173–198.

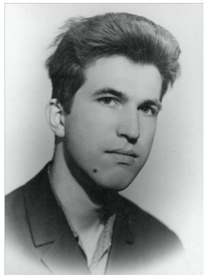
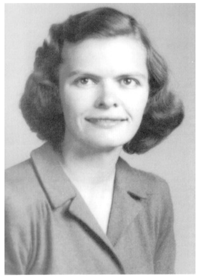
Hilbert's 10th Problem

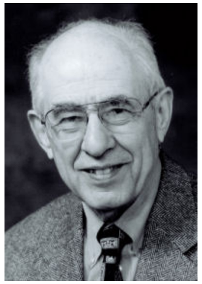
Determining if a given multivariate polynomial with integral coefficients has an integer root.

e.g. $5xy^2z + 8yz^3 + 100x^{99}$

Undecidable!

Proved in 1970 by Matiyasevich-Robinson-Davis-Putnam.



Does it have a **real** root? **Decidable!**

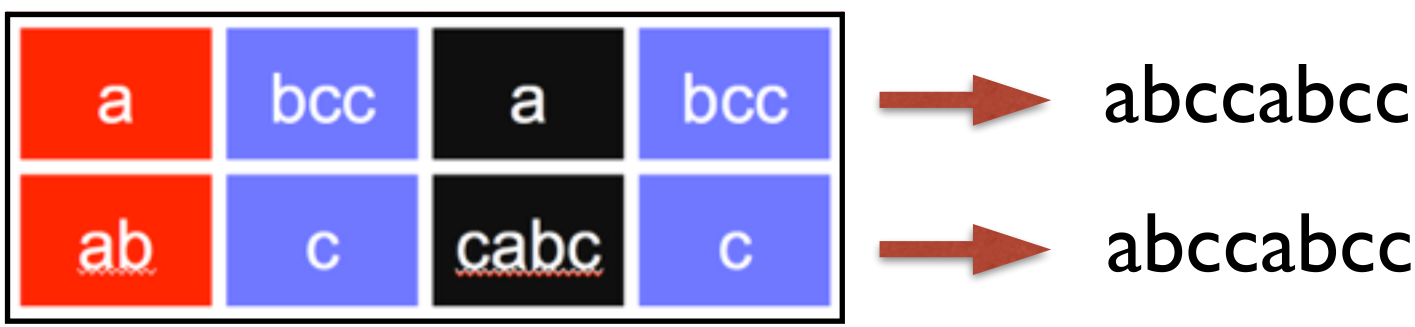
Proved in 1951 by Tarski.

Does it have a **rational** root? **No one knows!**

Post's Correspondence Problem

Input: A finite collection of "dominoes"
having strings written on each half.

Output: Accept if it is possible to match the strings.



Undecidable!

Proved in 1946 by Post.

Post's Correspondence Problem

Corresponding language is

$$\text{PCP} = \{\langle\text{Domino Set}\rangle : \text{there's a match}\}$$

Proof idea:

Show $\text{ACCEPTS} \leq_T \text{PCP}$.

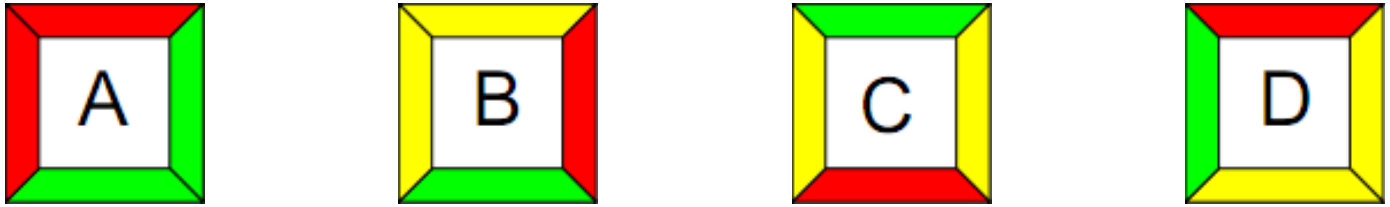
i.e. you want to solve ACCEPTS
assuming you can solve PCP.

$$\langle M, x\rangle \longrightarrow \langle\text{Domino Set}\rangle$$

Create a domino set such that only matches are computation traces of M that end in an accept state.

Wang Tiles

Input: A finite collection of "Wang Tiles" (squares)
with colors on the edges.



Output: Accept iff it is possible to make an infinite grid
from copies of the given squares,
where touching sides must color-match.

Undecidable!

Proved in 1966 by Berger.

Modular Systems

Input: A finite set of rules of the form

"from $ax+b$, can derive $cx+d$" **where** $a,b,c,d \in \mathbb{Z}$,

and a starting integer u, and a target integer v.

Output: Accept iff v can be derived starting from u.

e.g.

"from $2x$ derive x" "from $2x+1$ derive $6x+4$"

$v = 1$

Undecidable!

Proved in 1989 by Börger.

Mortal Matrices

Input: Two 21x21 matrices of integers A and B.

Output: Accept iff it is possible to multiply A and B together (multiple times in any order) to get to the 0 matrix.

Undecidable!

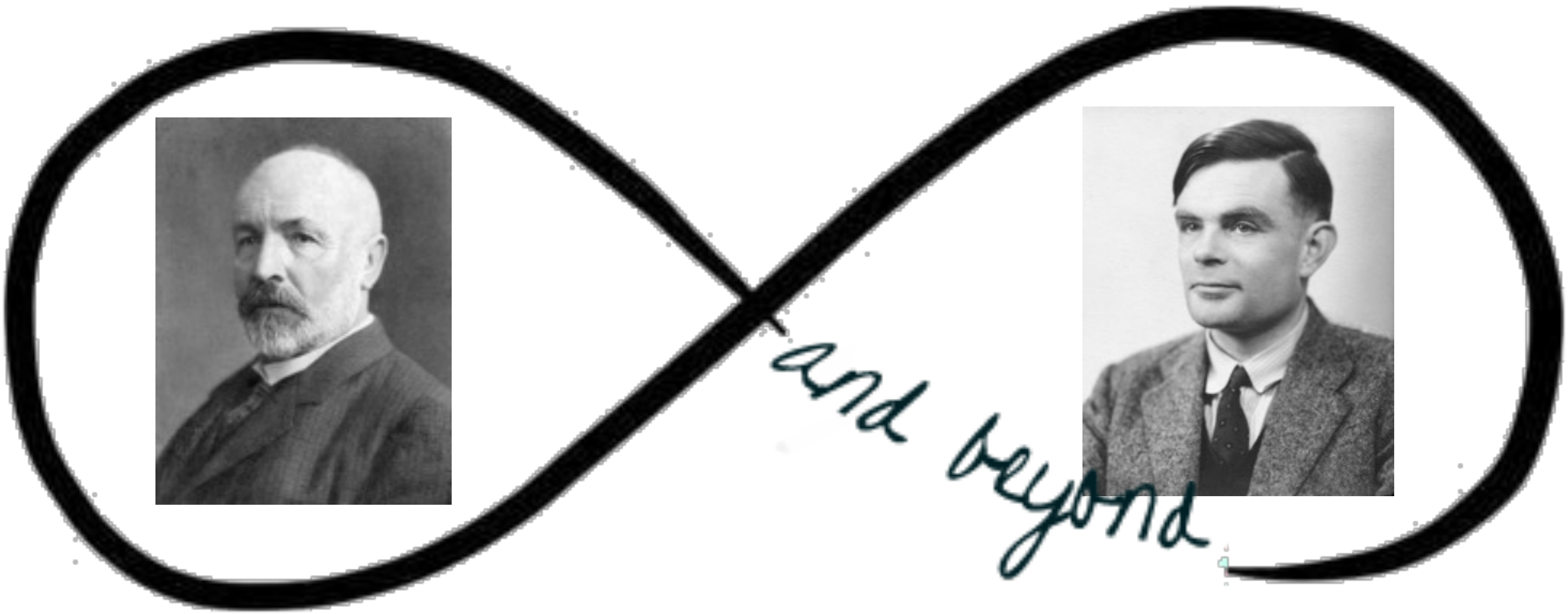
Proved in 2007 by Halava, Harju, Hirvensalo.

Most problems are undecidable.

Some very interesting problems undecidable.

But most interesting problems are decidable.

So what next?


and beyond