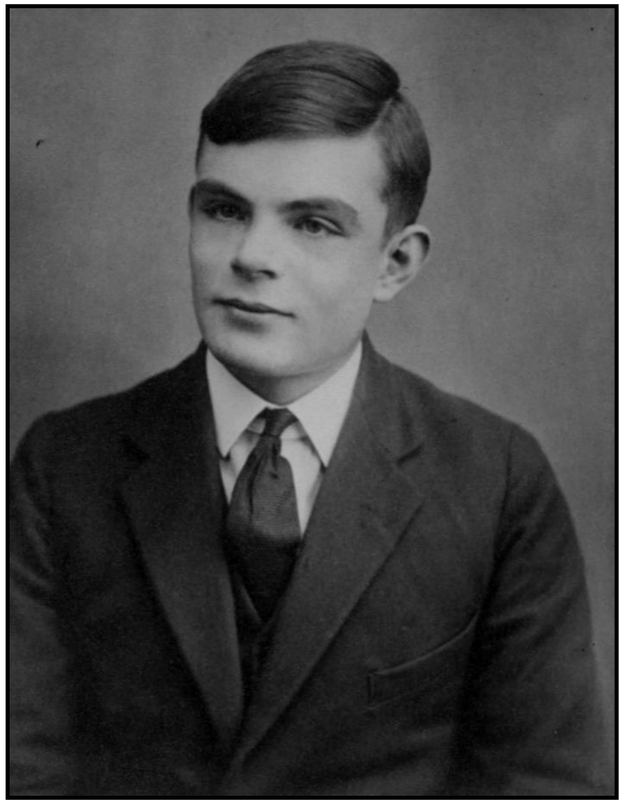
CS251

Great Ideas in *Theoretical* Computer Science



Limits of Computation 2: The Finite and Undecidable

Poll: Which ones are decidable?

$$\text{ACCEPTS}_{\text{TM}} = \{\langle M, x\rangle : M \text{ is a TM and } x \in \Sigma^* \text{ s.t. } x \in L(M)\}$$

$$\text{SELF-ACCEPTS}_{\text{TM}} = \{\langle M\rangle : M \text{ is a TM s.t. } \langle M\rangle \in L(M)\}$$

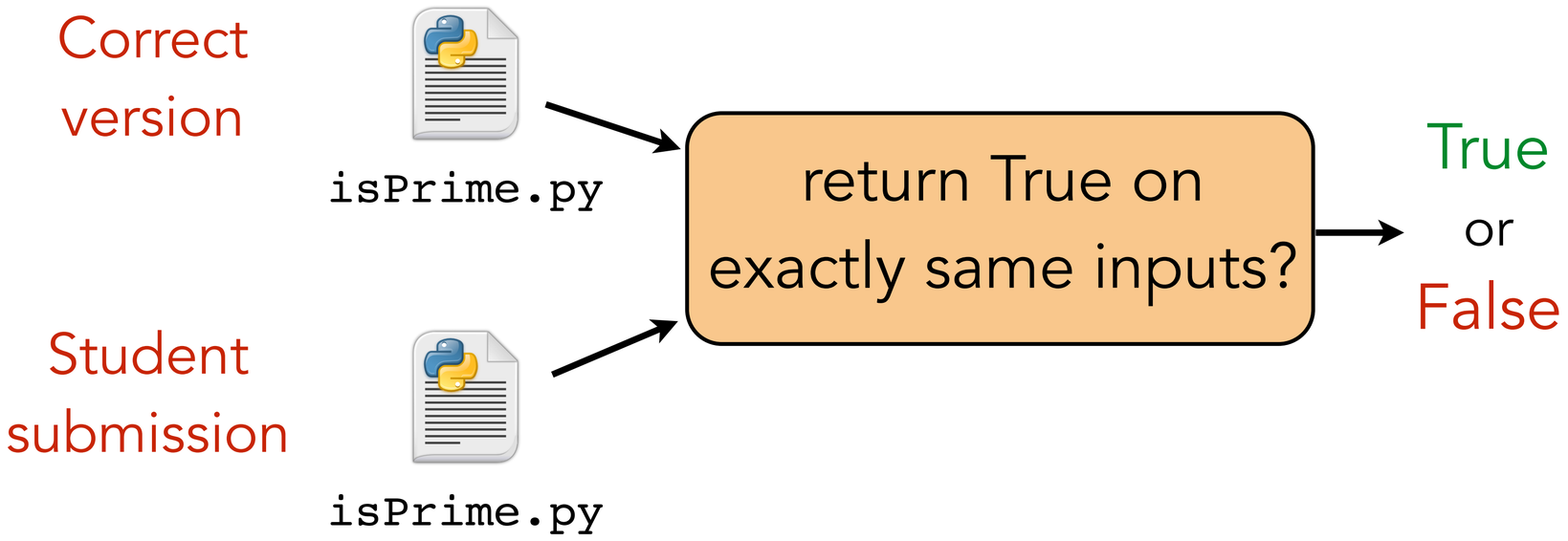
$$\text{HALTS}_{\text{TM}} = \{\langle M, x\rangle : M \text{ is a TM and } x \in \Sigma^* \text{ s.t. } M(x) \text{ halts}\}$$

$$\text{SAT}_{\text{TM}} = \{\langle M\rangle : M \text{ is a TM s.t. } M \text{ accepts some string}\}$$

$$\text{NEQ}_{\text{TM}} = \{\langle M_1, M_2\rangle : M_1, M_2 \text{ are TMs s.t. } L(M_1) \neq L(M_2)\}$$

Can we write an autograder?

$$\text{NEQ}_{\text{TM}} = \{\langle M_1, M_2 \rangle : M_1, M_2 \text{ are TMs s.t. } L(M_1) \neq L(M_2)\}$$

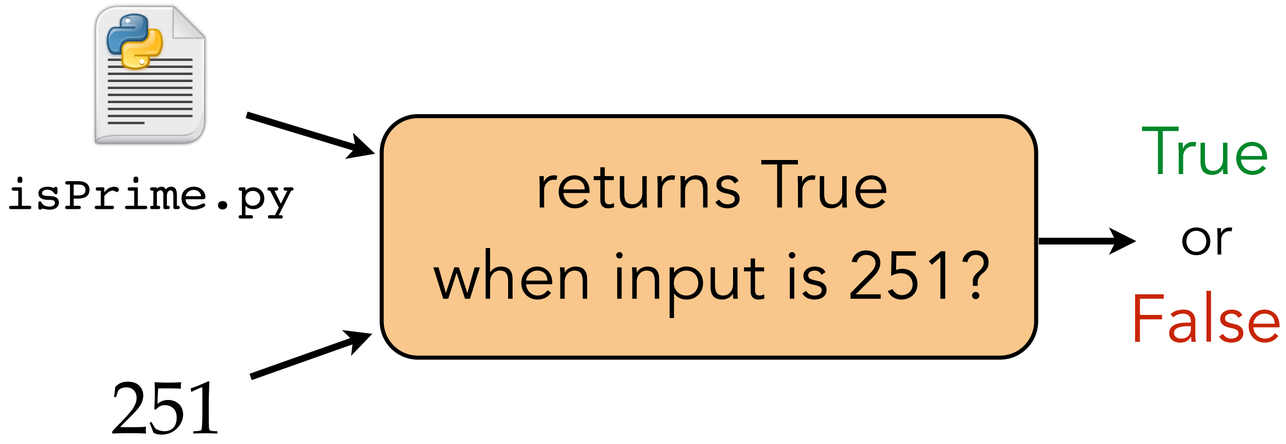


Is there an algorithm (TM) that solves NEQ?

(is NEQ decidable?)

Can we write an autograder?

$$\text{ACCEPTS}_{\text{TM}} = \{\langle M, x\rangle : M \text{ is a TM and } x \in \Sigma^* \text{ such that } x \in L(M)\}$$



Poll: Which ones are decidable?

$$\text{ACCEPTS} = \{\langle M, x\rangle : M \text{ is a TM and } x \in \Sigma^* \text{ s.t. } x \in L(M)\}$$

$$\text{SELF-ACCEPTS} = \{\langle M\rangle : M \text{ is a TM s.t. } \langle M\rangle \in L(M)\}$$

$$\text{HALTS} = \{\langle M, x\rangle : M \text{ is a TM and } x \in \Sigma^* \text{ s.t. } M(x) \text{ halts}\}$$

$$\text{SAT} = \{\langle M\rangle : M \text{ is a TM s.t. } M \text{ accepts some string}\}$$

$$\text{NEQ} = \{\langle M_1, M_2\rangle : M_1, M_2 \text{ are TMs s.t. } L(M_1) \neq L(M_2)\}$$

Last time in 251

Functions $\mathcal{F}$	Inputs X			
	x_1	x_2	x_3	x_4
f_1	0	0	1	0
f_2	1	1	1	0
f_3	1	0	0	0
f_4	1	0	1	1
f_D	**1**	**0**	**1**	**0**

Given:

A set $\mathcal{F}$ of functions $f : X \to \{0,1\}$.

Goal:

Construct a function f_D different from each $f \in \mathcal{F}$.

How:

$\forall f \in \mathcal{F}$,
pick a unique input $x \in X$,
and make $f_D(x) \neq f(x)$.

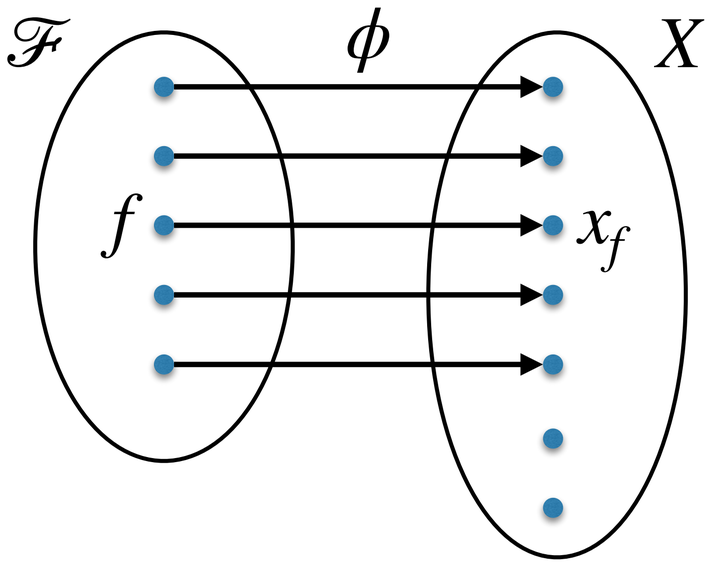
Condition needed:

$|X| \geq |\mathcal{F}|$

Diagonalization Lemma:

Let X be any set and let $\mathcal{F}$ be any set of functions $f : X \to \{0,1\}$.

If $|X| \geq |\mathcal{F}|$, we can construct $f_D : X \to \{0,1\}$ not in $\mathcal{F}$.



	$\phi(f_1)$	$\phi(f_2)$	$\phi(f_3)$	$\phi(f_4)$	⋯
f_1	0	0	1	0	⋯
f_2	1	1	1	0	⋯
f_3	1	0	0	0	⋯
f_4	1	0	1	1	⋯
⋮	⋮	⋮	⋮	⋮	
f_D	1	0	1	0	⋯

Diagonalization Lemma:

Let X be any set and let $\mathscr{F}$ be any set of functions $f : X \to \{0,1\}$.

If $|X| \geq |\mathscr{F}|$, we can construct $f_D : X \to \{0,1\}$ not in $\mathscr{F}$.

This is called "diagonalization against $\mathscr{F}$".

Diagonalization produces an explicit f_D outside $\mathscr{F}$.

You can pretty much view anything as a function.

The range need not be $\{0,1\}$.

Diagonalization Lemma:

Let X be any set and let $\mathscr{F}$ be any set of functions $f : X \to Y$, where $|Y| \geq 2$.

If $|X| \geq |\mathscr{F}|$, we can construct $f_D : X \to Y$ not in $\mathscr{F}$.

This is called "diagonalization against $\mathscr{F}$".

Diagonalization produces an explicit f_D outside $\mathscr{F}$.

You can pretty much view anything as a function.

The range need not be $\{0,1\}$.

Diagonalization Lemma:

Let X be any set and let $\mathscr{F}$ be any set of functions $f : X \to Y$, where $|Y| \geq 2$.

If $|X| \geq |\mathscr{F}|$, we can construct $f_D : X \to Y$ not in $\mathscr{F}$.

So $|X| \geq |\mathscr{F}| \implies \exists f_D : X \to Y$ not in $\mathscr{F}$.

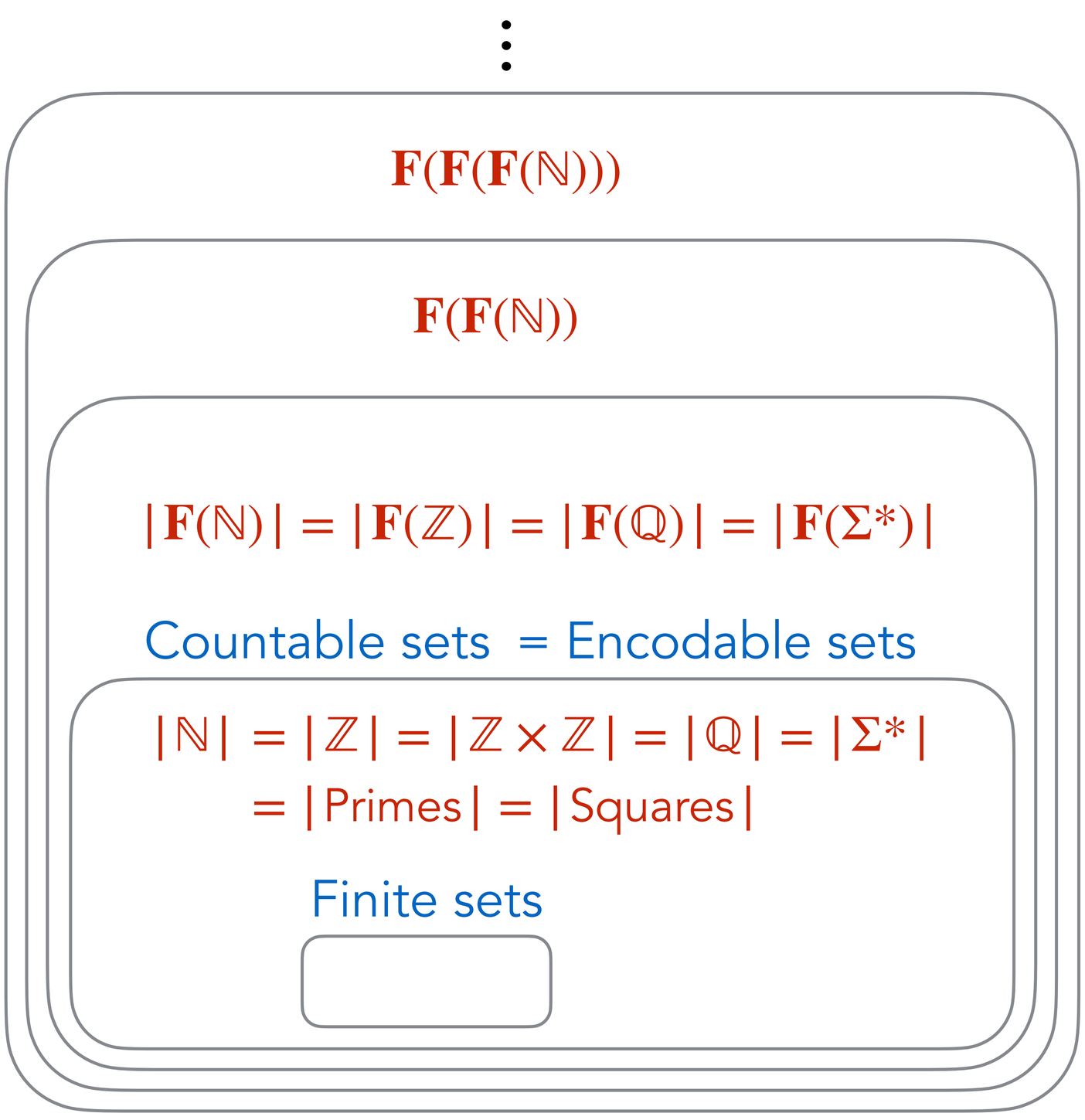
i.e. $\nexists f_D : X \to Y$ not in $\mathscr{F} \implies |X| < |\mathscr{F}|$.

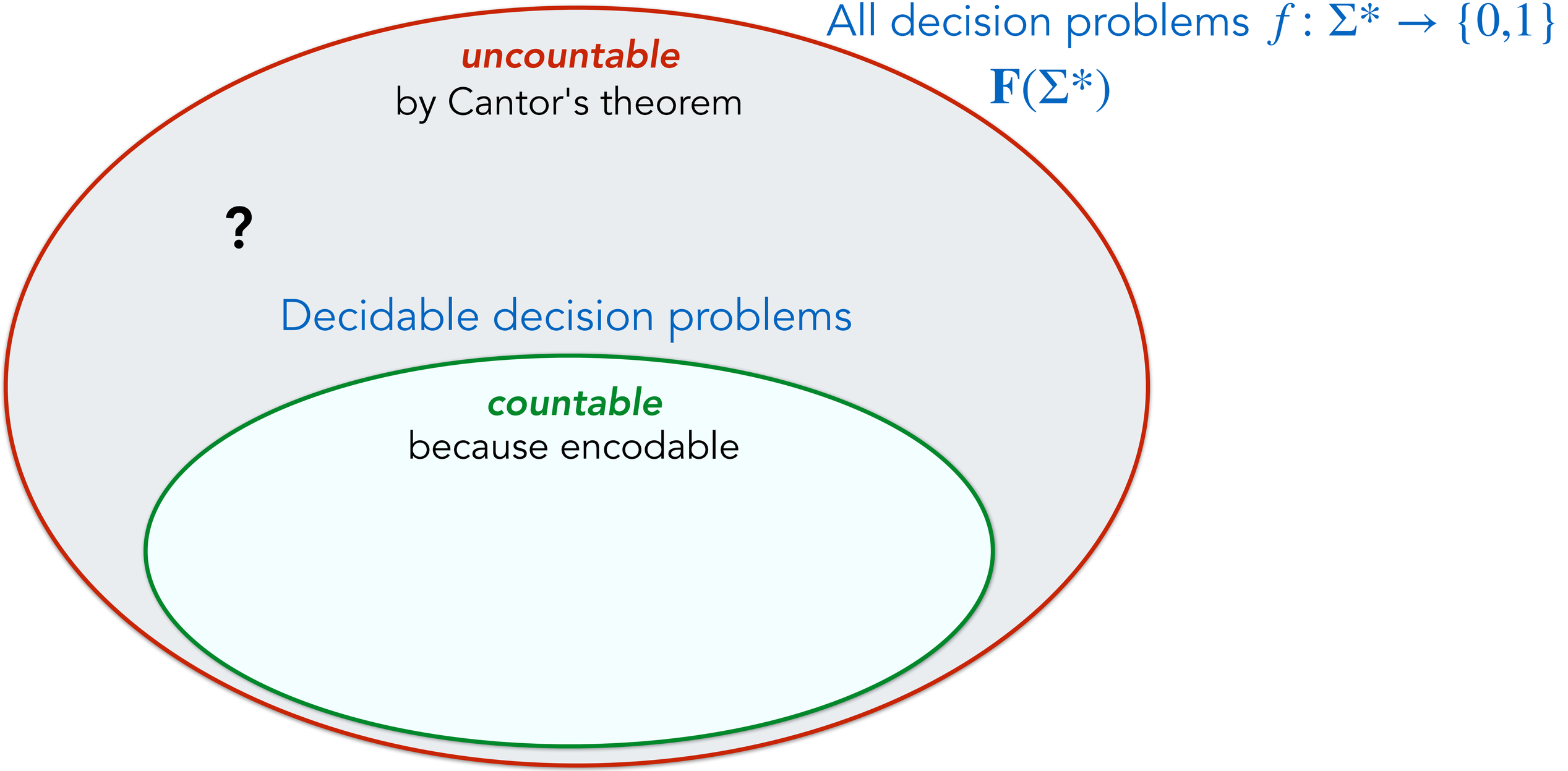
Definition: $\mathbf{F}(X)$ = set of **all** functions $f : X \to \{0,1\}$.

Corollary (Cantor's Theorem): For every set X, $|X| < |\mathbf{F}(X)|$.

Corollary: $|\mathbb{N}| < |\mathbf{F}(\mathbb{N})|$, so $\mathbf{F}(\mathbb{N})$ is uncountable.

Corollary: $|\Sigma^*| < |\mathbf{F}(\Sigma^*)|$, so $\mathbf{F}(\Sigma^*)$ is uncountable.


$\mathbf{F}(\mathbf{F}(\mathbf{F}(\mathbb{N})))$
$\mathbf{F}(\mathbf{F}(\mathbb{N}))$
$|\mathbf{F}(\mathbb{N})| = |\mathbf{F}(\mathbb{Z})| = |\mathbf{F}(\mathbb{Q})| = |\mathbf{F}(\Sigma^*)|$
Countable sets = Encodable sets
$|\mathbb{N}| = |\mathbb{Z}| = |\mathbb{Z} \times \mathbb{Z}| = |\mathbb{Q}| = |\Sigma^*|$
$= |\text{Primes}| = |\text{Squares}|$
Finite sets



Encoding of a **decidable** decision problem f: $\langle M \rangle$ (where TM M solves f)

All decision problems $f : \Sigma^* \to \{0,1\}$

$\mathbf{F}(\Sigma^*)$

Decidable

All decision problems $f : \Sigma^* \to \{0,1\}$

$\mathbf{F}(\Sigma^*)$

Too many problems/languages!

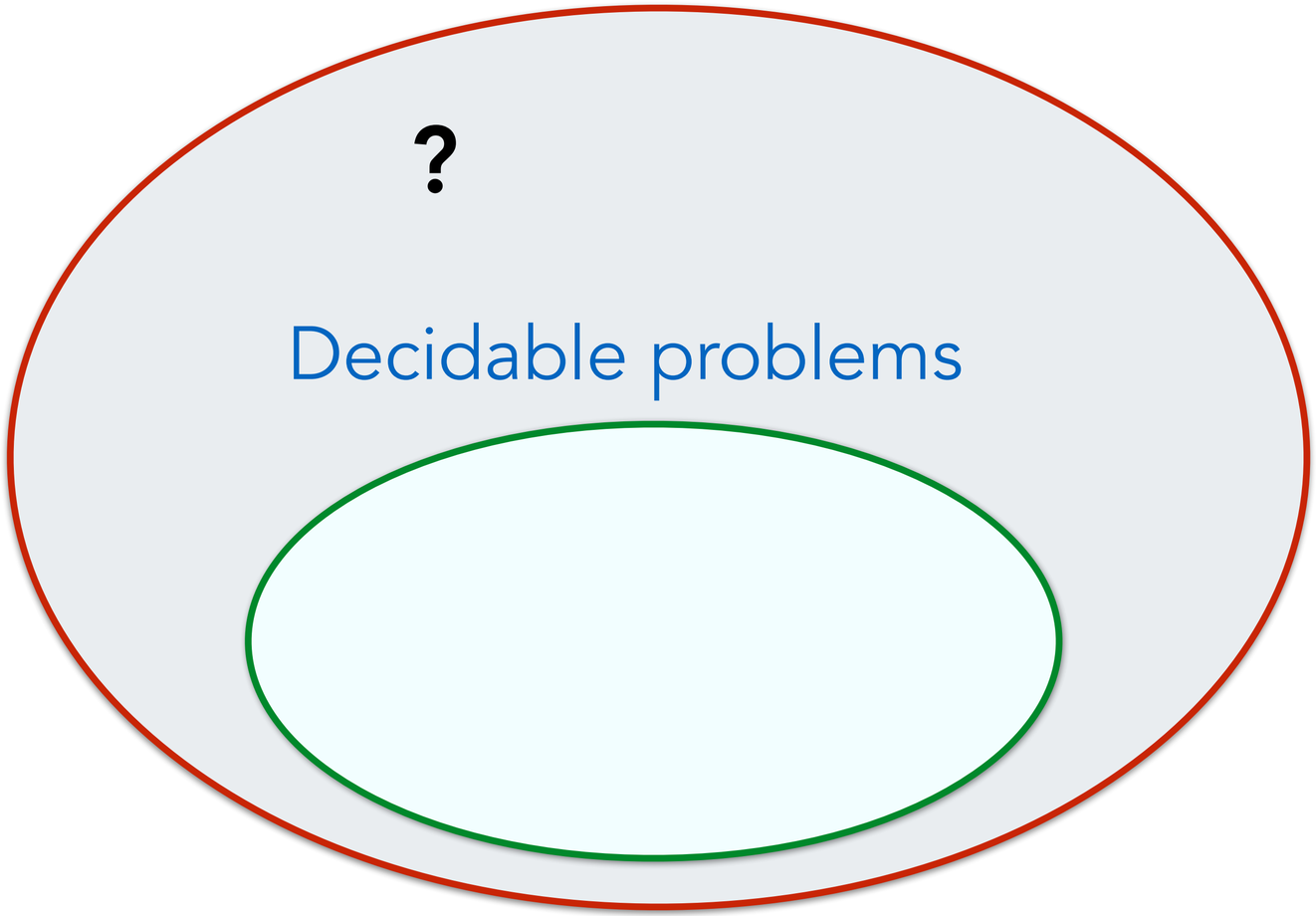
Most cannot be even communicated!

(beyond mathematical analysis)

Decidable

(can be mathematically communicated/analysed)

Finitely describable problems



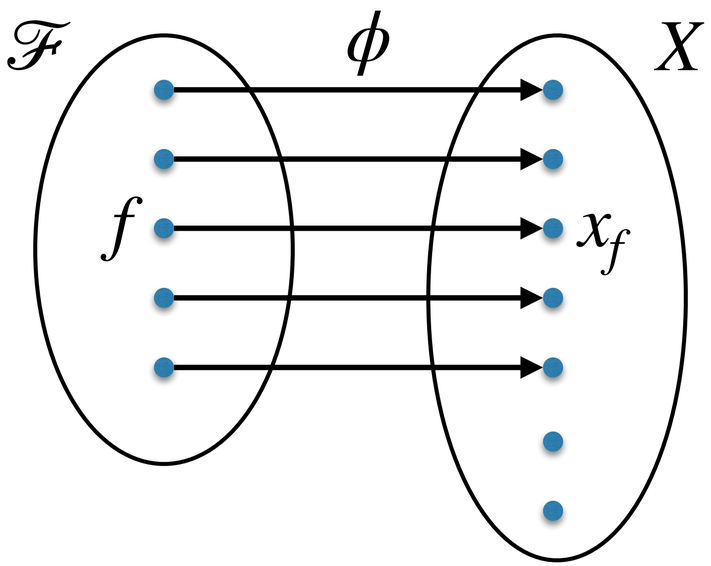
? Is there an **explicit** undecidable problem?

Limits of Computation:
The Finite and Undecidable

Great Idea:

Diagonalizing against a set produces an explicit object not in that set.

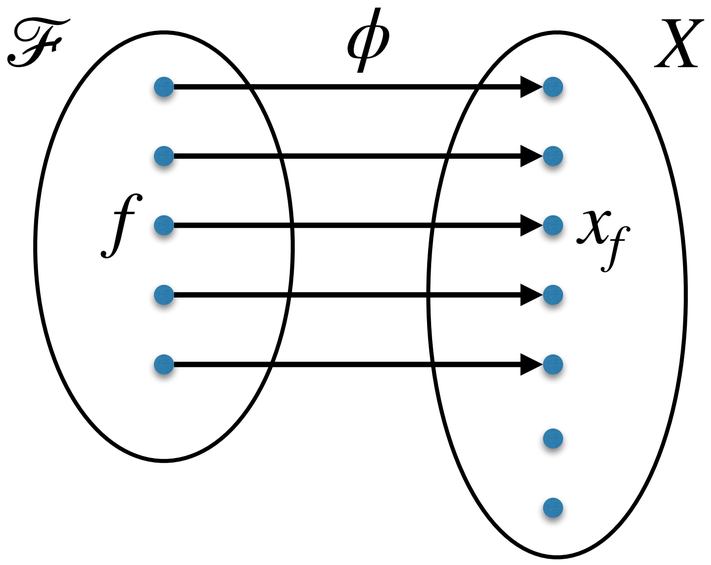
Diagonalization against the set of all TMs:



	$\phi(f_1)$	$\phi(f_2)$	$\phi(f_3)$	$\phi(f_4)$	⋯
f_1	0	0	1	0	⋯
f_2	1	1	1	0	⋯
f_3	1	0	0	0	⋯
f_4	1	0	1	1	⋯
⋮	⋮	⋮	⋮	⋮	
f_D	1	0	1	0	⋯

Diagonalization against the set of all TMs:

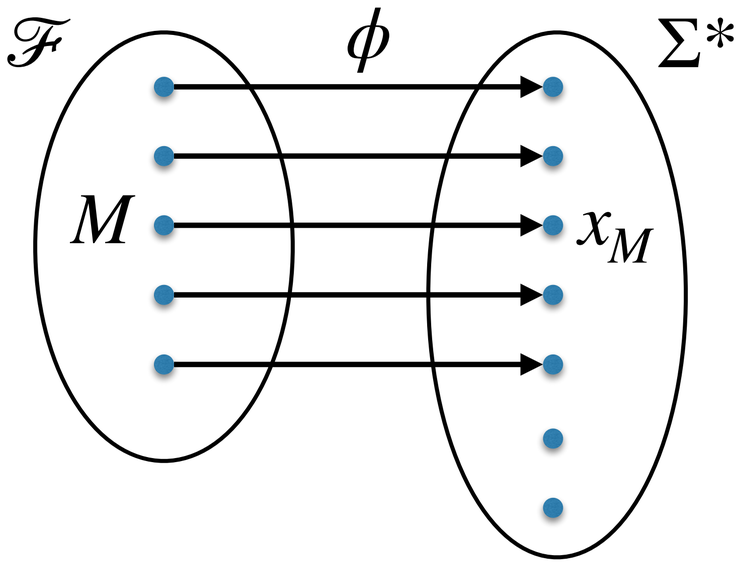
- $\mathcal{F}$ = set of all Turing machines M (mapping Σ^* to $\{0,1,\infty\}$).
- Need: $|\Sigma^*| \geq |\mathcal{F}|$ ✓



	$\phi(f_1)$	$\phi(f_2)$	$\phi(f_3)$	$\phi(f_4)$	⋯
f_1	**0**	0	1	0	⋯
f_2	1	**1**	1	0	⋯
f_3	1	0	**0**	0	⋯
f_4	1	0	1	**1**	⋯
⋮	⋮	⋮	⋮	⋮	
f_D	1	0	1	0	⋯

Diagonalization against the set of all TMs:

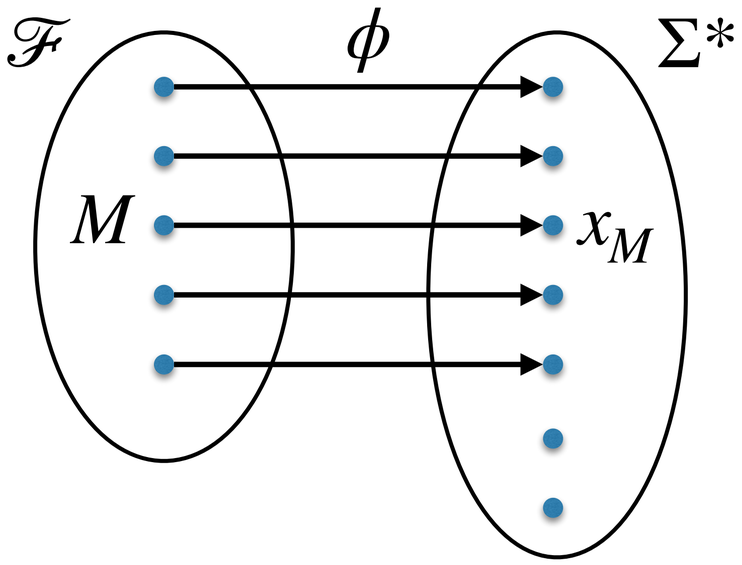
- $\mathcal{F}$ = set of all Turing machines M (mapping Σ^* to $\{0,1,\infty\}$).
- Need: $|\Sigma^*| \geq |\mathcal{F}|$ ✓



	$\phi(f_1)$	$\phi(f_2)$	$\phi(f_3)$	$\phi(f_4)$	⋯
f_1	**0**	0	1	0	⋯
f_2	1	**1**	1	0	⋯
f_3	1	0	**0**	0	⋯
f_4	1	0	1	**1**	⋯
⋮	⋮	⋮	⋮	⋮	
f_D	1	0	1	0	⋯

Diagonalization against the set of all TMs:

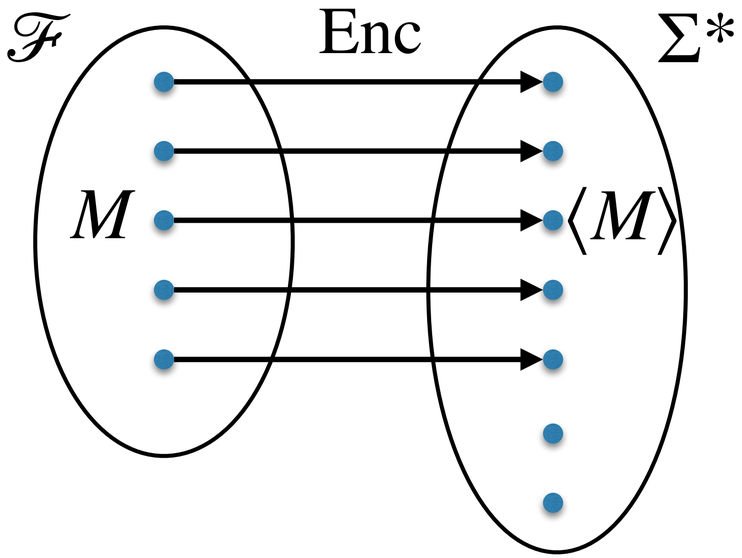
- $\mathcal{F}$ = set of all Turing machines M (mapping Σ^* to $\{0,1,\infty\}$).
- Need: $|\Sigma^*| \geq |\mathcal{F}|$ ✓



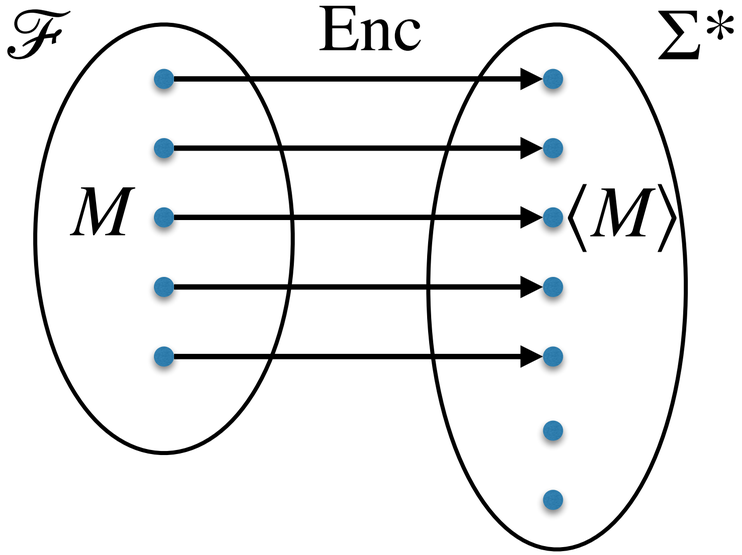
	$\phi(M_1)$	$\phi(M_2)$	$\phi(M_3)$	$\phi(M_4)$	⋯
M_1	0	∞	1	0	⋯
M_2	1	1	1	∞	⋯
M_3	1	0	∞	0	⋯
M_4	1	0	1	∞	⋯
⋮	⋮	⋮	⋮	⋮	
f_D	1	0	1	1	⋯

Diagonalization against the set of all TMs:

- $\mathcal{F}$ = set of all Turing machines M (mapping Σ^* to $\{0,1,\infty\}$).
- Need: $|\Sigma^*| \geq |\mathcal{F}|$ ✓



	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	⋯
M_1	0	∞	1	0	⋯
M_2	1	1	1	∞	⋯
M_3	1	0	∞	0	⋯
M_4	1	0	1	∞	⋯
⋮	⋮	⋮	⋮	⋮	
f_D	1	0	1	1	⋯



	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	⋯
M_1	0	∞	1	0	⋯
M_2	1	1	1	∞	⋯
M_3	1	0	∞	0	⋯
M_4	1	0	1	∞	⋯
⋮	⋮	⋮	⋮	⋮	
f_D	1	0	1	1	⋯

Conclusions:

- For every TM M_i, $f_D(\langle M_i \rangle) \neq M_i(\langle M_i \rangle)$. f_D is undecidable!
- f_D corresponds to $L = \{ \langle M \rangle : M(\langle M \rangle) \in \{0, \infty\}\} = \{\langle M \rangle : \langle M \rangle \notin L(M)\}$

Theorem (1st Explicit Undecidable Language)

Theorem: $\overline{\text{SELF-ACCEPTS}}$ is undecidable.

Theorem 2: SELF-ACCEPTS is undecidable

$$\text{SELF-ACCEPTS} = \{\langle M \rangle : M \text{ is a TM s.t. } \langle M \rangle \in L(M)\}$$

Proof: AFSOC SELF-ACCEPTS is decidable.

So ∃ decider M_{SA} that decides SELF-ACCEPTS.

Then we can construct $M_{\overline{\text{SA}}}$ deciding $\overline{\text{SELF-ACCEPTS}}$:

```
def M_SA'(⟨M⟩):
    return not M_SA(⟨M⟩)
```

But $\overline{\text{SELF-ACCEPTS}}$ is undecidable. Contradiction.

Theorem 3: ACCEPTS is undecidable

$$\text{ACCEPTS} = \{\langle M, x\rangle : M \text{ is a TM and } x \in \Sigma^* \text{ s.t. } x \in L(M)\}$$

Proof: AFSOC ACCEPTS is decidable.

So ∃ decider M_{A} that decides ACCEPTS.

Then we can construct M_{SA} deciding SELF-ACCEPTS:

```
def M_SA(⟨M⟩):
    return M_A(⟨M, ⟨M⟩⟩)
```

But SELF-ACCEPTS is undecidable. Contradiction.

Theorem 4 (Turing): HALTS is undecidable

$\text{HALTS} = \{\langle M, x\rangle : M \text{ is a TM and } x \in \Sigma^* \text{ s.t. } M(x) \text{ halts}\}$

Proof: AFSOC HALTS is decidable. So ∃ decider M_{H} that decides HALTS.

Then we can construct M_{A} deciding ACCEPTS:

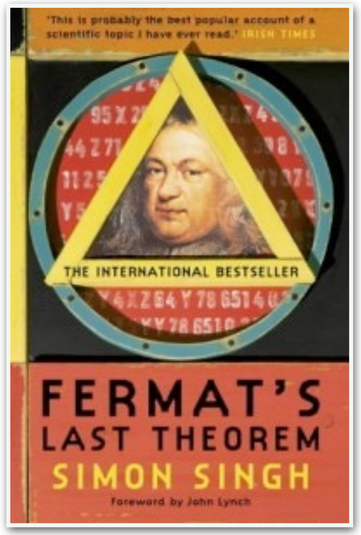
```
def M_A(⟨M, x⟩):
    run M_H(⟨M, x⟩)
    if it rejects: reject
    else:
        run M(x)
        if it accepts: accept
        if it rejects: reject
```

Some consequences

- Program verification is hard!

- No guaranteed autograder program.

- Consider the following program:




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

Does this program halt?

Some consequences

- 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;
        end if
    end for
end do
```

Goldbach Conjecture

Does this program halt?

Some consequences

- **Reductions:** show new problems are undecidable.

  e.g. Entscheidungsproblem, Hilbert's 10th problem

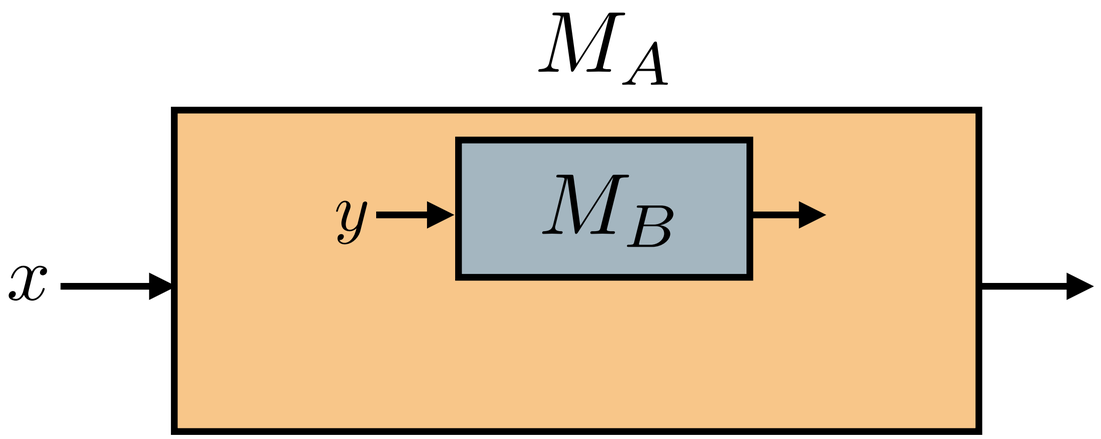
- By Physical Church-Turing Thesis

  we are proving the computational limits of our universe.

Revisiting Reductions

We write $A \leq B$ if you can do the following:

- assume $\exists\ M_B$ solving B,
- construct M_A solving A (using M_B as a subroutine).



Revisiting Reductions

We write $A \leq B$ if you can do the following:

- assume $\exists\ M_B$ solving B,
- construct M_A solving A (using M_B as a subroutine).

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

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

helper function

Revisiting Reductions

We write $A \leq B$ if you can do the following:

- assume $\exists\ M_B$ solving B,
- construct M_A solving A (using M_B as a subroutine).

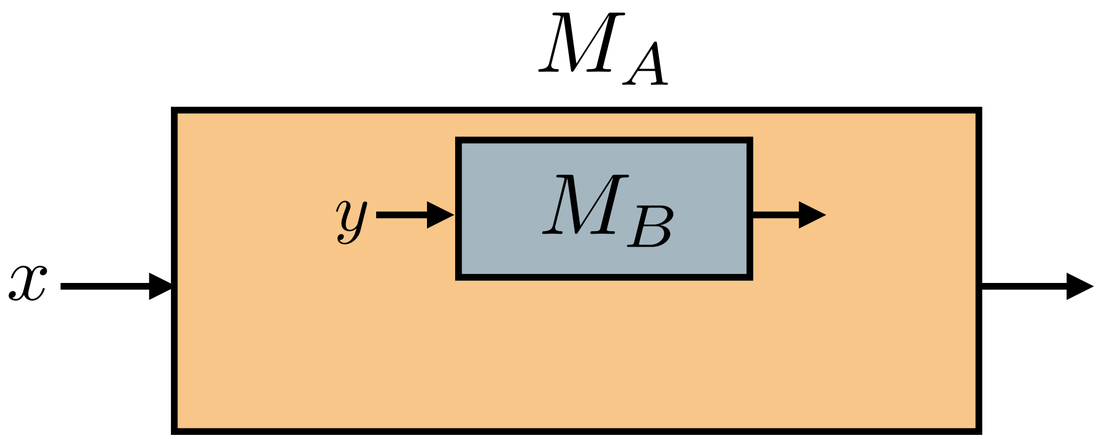
```
from God import fooB


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

To show $A \leq B$: Give me the code for fooA.

Revisiting Reductions

We write $A \leq B$ if you can do the following:

- assume $\exists\, M_B$ solving B,
- construct M_A solving A (using M_B as a subroutine).



B decidable $\implies$ A decidable

A undecidable $\implies$ B undecidable

$A \leq B$: A **is no harder than** B (with respect to decidability).

Revisiting Reductions

Expand the landscape of undecidable languages:

ACCEPTS is undecidable.

If $\text{ACCEPTS} \leq B$, then B is undecidable.

Proved: $\text{ACCEPTS} \leq \text{HALTS}$.

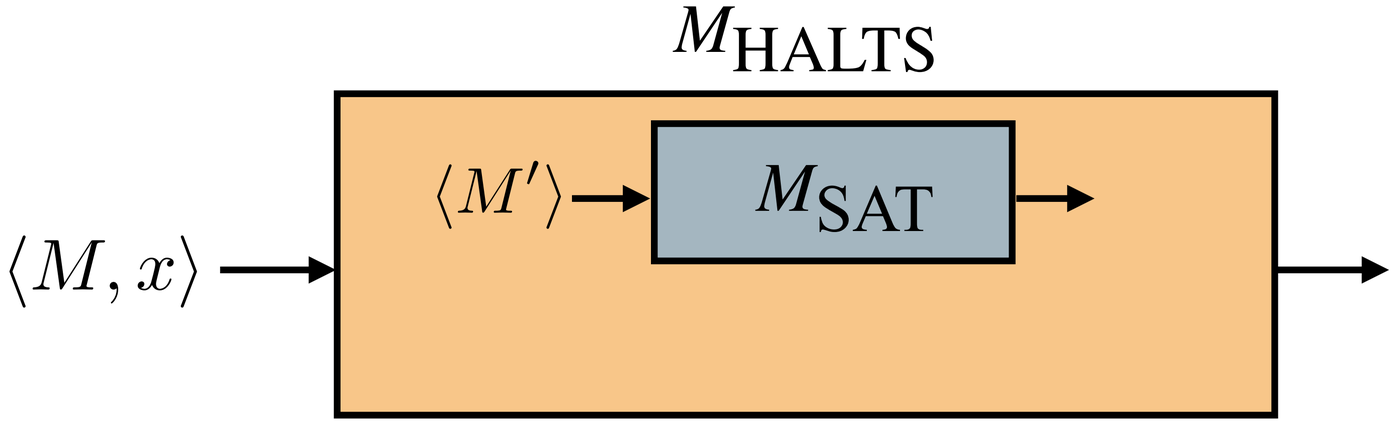
If $\text{HALTS} \leq B$, then B is undecidable.

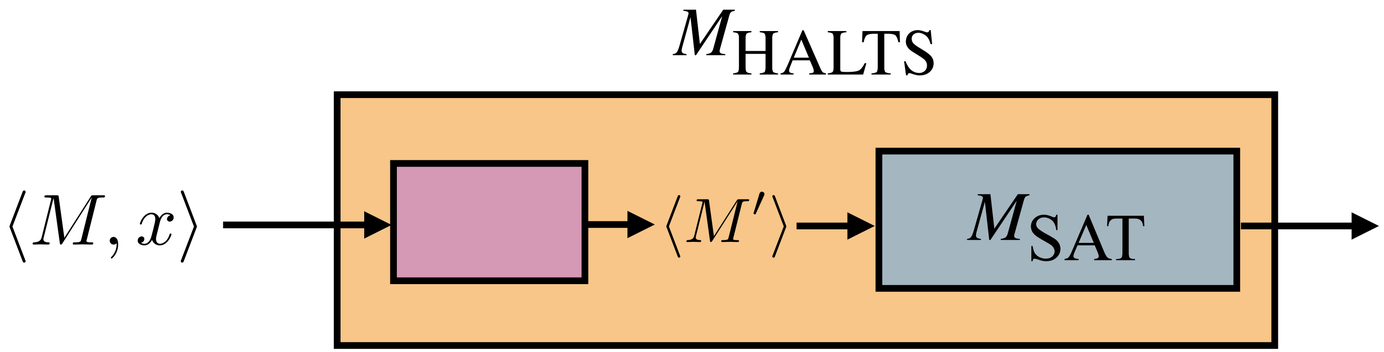
$\vdots$

Theorem 5: HALTS $\leq$ SAT

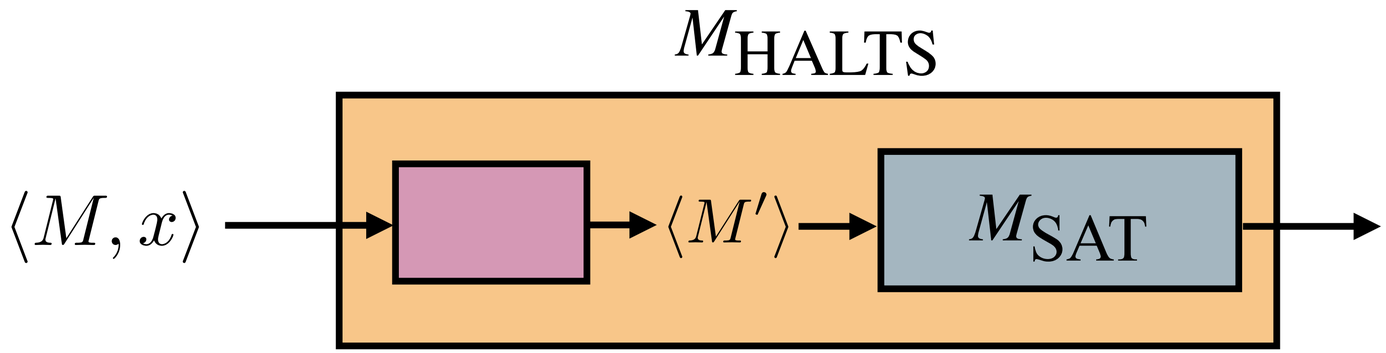
$\text{HALTS} = \{\langle M, x\rangle : M \text{ is a TM and } x \in \Sigma^* \text{ s.t. } M(x) \text{ halts}\}$

$\text{SAT} = \{\langle M\rangle : M \text{ is a TM s.t. } L(M) \neq \varnothing\}$





Theorem 5: HALTS ≤ SAT



$M(x)$ halts ⟶ $L(M') \neq \emptyset$

$M(x)$ loops ⟶ $L(M') = \emptyset$

def $M_{\text{HALTS}}(\langle M, x \rangle)$:

    def $M'(y)$:

        run $M(x)$

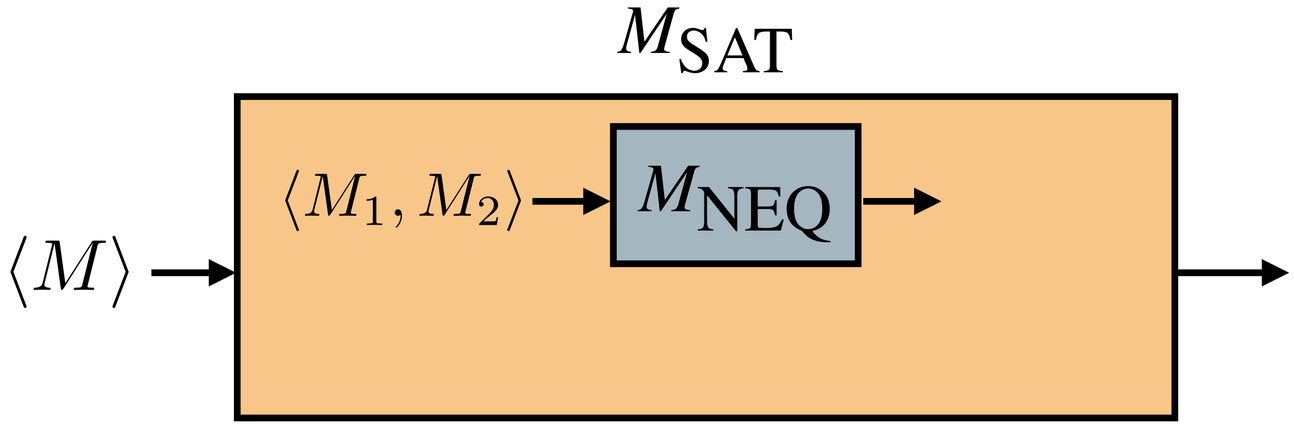
        accept

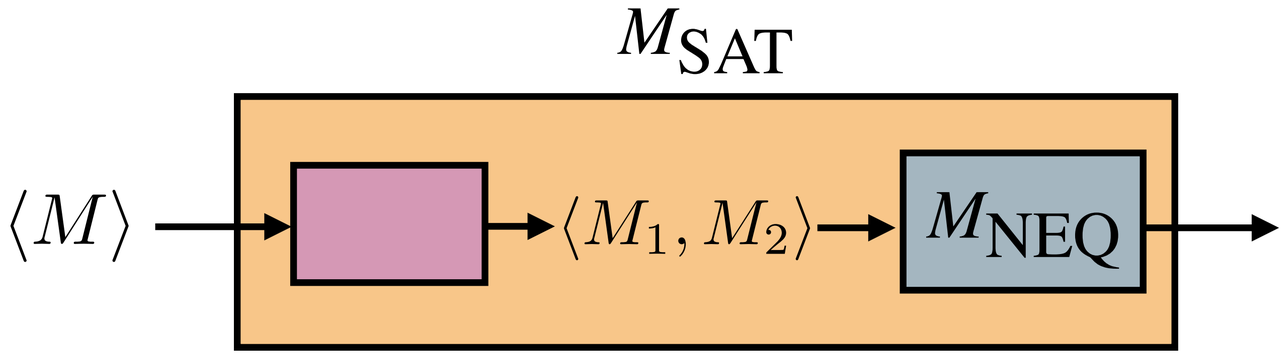
    return $M_{\text{SAT}}(\langle M' \rangle)$

Theorem 6: $\text{SAT} \leq \text{NEQ}$

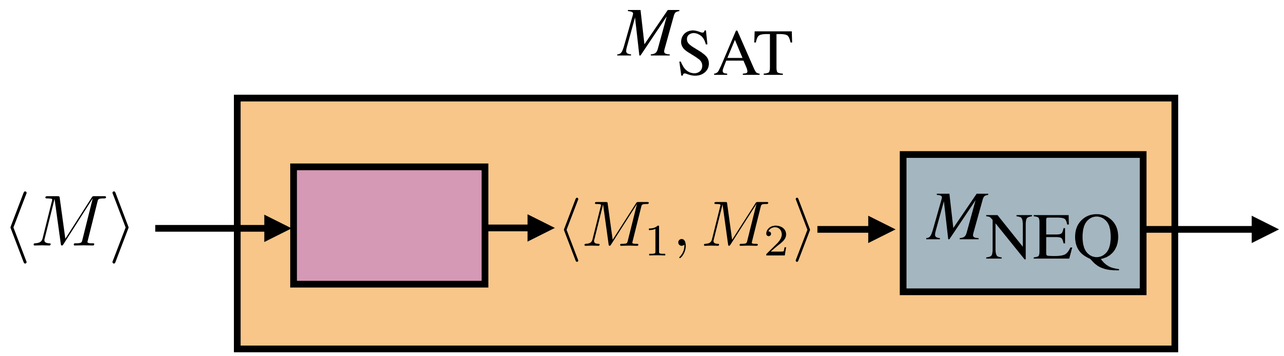
$\text{SAT} = \{\langle M \rangle : M \text{ is a TM s.t. } L(M) \neq \emptyset\}$

$\text{NEQ} = \{\langle M_1, M_2 \rangle : M_1, M_2 \text{ are TMs s.t. } L(M_1) \neq L(M_2)\}$





Theorem 6: $\text{SAT} \leq \text{NEQ}$



$$L(M) \neq \varnothing \longrightarrow L(M_1) \neq L(M_2)$$
$$L(M) = \varnothing \longrightarrow L(M_1) = L(M_2)$$

def $M_{\text{SAT}}(\langle M \rangle)$:

  def $M_2(x)$:

    reject

  return $M_{\text{NEQ}}(\langle M, M_2 \rangle)$

$L(M_2) = \varnothing$

Summary

Diagonalize against **the set of all decidable languages:**

$\overline{\text{SELF-ACCEPTS}}$ is undecidable.

$$\overline{\text{SELF-ACCEPTS}} \leq \text{SELF-ACCEPTS} \leq \text{ACCEPTS}$$

$$\leq \text{HALTS} \leq \text{SAT} \leq \text{NEQ}$$