15-251: Great Theoretical Ideas in Computer Science

Fall 2016 Lecture 6

September 15, 2016

Turing & the Uncomputable

3-slide review of last lecture

Comparing the cardinality of sets

$|A| \leq |B|$
if there is an injection (one-to-one map) from A to B

$|A| \geq |B|$
if there is a surjection (onto map) from A to B

$|A| = |B|$
if there is a bijection from $A\ to\ B$

$|A| > |B|$
if there is no surjection from B to A
(or equivalently, there is no injection from A to B)

Countable and uncountable sets

- Set A is countable if $|A| \leq |\mathbb{N}|$
- Set A is countably infinite if it is countable and infinite, i.e., $|A| = |\mathbb{N}|$ (there's a bijection from A to $\mathbb{N}$)
- Set A is uncountable if it is not countable, i.e., $|A| > |\mathbb{N}|$

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 surjection from $\mathbb{N}$ to A
- Show that $|A| \leq |B|$ where

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

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

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

Proving sets countable using computation

For example, f(n) = 'the n^{th} prime'.

You could write a program (Turing machine) to compute f.

So this is a well-defined rule.

Or: f(n) = the n^{th} rational in our listing of $\mathbb{Q}$.

(List $\mathbb{Z}^2$ via the spiral, omit the terms p/0, omit rationals seen before…)

You could write a program to compute this f.

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

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 $x \in \Sigma^*$ is an input to M

Uncountable to uncomputable

The real number 1/7 is "computable".
You could write a (non-halting) program
(in your favorite language)
which printed out all its digits:

.142857142857142857…

The same is true of $\sqrt{2}$, π, e,
" the first prime larger than $2^{43,112,609}$ ", etc.;
indeed, any real number "you can think of".

Uncountable to uncomputable

However, the set of all programs
(in your favorite language)
is just Σ^*, for some finite alphabet Σ.

Hence the set of all programs is countable.

Hence the set of all
"computable reals" is countable.

But $\mathbb{R}$ is uncountable.

Therefore there exist "uncomputable reals".

Recap: Turing Machines

Rules of computation:

Tape initialized with input $x \in \Sigma^*$ placed starting at square 0, preceded & followed by infinite ⊔'s.

Control starts in state q_0, head starts in square 0.

If the current state is q and head is reading symbol $s \in \Gamma$, the machine transitions according to $\delta(q,s)$, which gives:

- the next state,
- what tape symbol to overwrite the current square with,
- and whether the head moves Left or Right.

Continues until either the accept state or reject state reached.

When accept/reject state is reached, M **halts**.

M might also never halt, in which case we say it **loops**.

Formal definition of Turing Machines

A Turing Machine is a 7-tuple

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$:

Q is a finite set of **states**,

Σ is a finite **input alphabet** (with $\sqcup \notin \Sigma$),

Γ is a finite **tape alphabet** (with $\sqcup \in \Gamma$, $\Sigma \subseteq \Gamma$)

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ is **transition function**,

$q_0 \in Q$ is the **start state**,

$q_{accept} \in Q$ is the **accept state**,

$q_{reject} \in Q$ is the **reject state**, $q_{reject} \neq q_{accept}$.

Decidable languages

Definition:

A language $L \subseteq \Sigma^*$ is **decidable** if there is a Turing Machine M which:

1. **Halts on every input** $x \in \Sigma^*$.
2. Accepts inputs $x \in L$ and rejects inputs $x \notin L$.

Such a Turing Machine is called a **decider**.

It 'decides' the language L.

We like deciders. We don't like TM's that sometimes loop.

Computable functions

An equivalence between

languages and (Boolean-valued) functions:

function $f : \{0,1\}^* \to \{0,1\} \equiv$ subset $L \subseteq \{0,1\}^*$

$$L = \{x \in \{0,1\}^* : f(x) = 1\}$$

$$f(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$$

If L is **decidable** we call f **computable**,

and vice versa.

Decidable languages

Examples:

Hopefully you're convinced that $\{0^n1^n : n\in\mathbb{N}\}$
is decidable. (Recall it's not "regular".)

The language $\{0^{2^n} : n\in\mathbb{N}\} \subseteq \{0\}^*$,
i.e. {0, 00, 0000, 00000000, …},
is decidable.

Proof: You can describe decider TMs for these…

Describing Turing Machines

Low Level:

Explicitly describing all states and transitions.

Medium Level:

Carefully describing in English how the TM operates. Should be ‘obvious’ how to translate into a Low Level description.

High Level:

Skips ‘standard’ details, just highlights ‘tricky’ details. **For experts only!**

$\{0^{2^n} : n \in \mathbb{N}\}$ is decidable

Medium Level description:

1. Sweep from left to right across the tape, overwriting a # over top of every *other* 0.
2. If you saw one 0 on the sweep, **accept**.
3. If you saw an odd number of 0's, **reject**.
4. Move back to the leftmost square.
   (Say you write a marker on the leftmost square at the very beginning so that you can recognize it later.)
5. Go back to step 1.

TM programming exercises & tricks

- Convert input $x_1x_2x_3\cdots x_n$ to $x_1\sqcup x_2\sqcup x_3\sqcup\cdots\sqcup x_n$.
- Simulate a big Γ by just $\{0,1,\sqcup\}$. (Or just $\{0,\sqcup\}$!)
- Increment/decrement a number in binary.
- Copy sections of tape from one spot to another.
- Simulate having 2 tapes, with separate heads.
- Create a Turing Machine U whose input is

  $\langle M\rangle$, the **encoding** of a TM M,

  x, a string

  and which **simulates** the execution of M on x.

Universal Turing Machine

If you get stuck on the last exercise, you can look up the answer in Turing’s 1936 paper!

Such a simulating TM is called a **universal Turing Machine**.

TM's: good definition of computation?

After playing with them for a while,
you'll become convinced you can program
TM's to compute anything you could compute
using Python, Java, ML, C++, etc.

(and using arbitrarily much memory!)

You were probably already convinced that
Python, Java, ML, C++, etc. can all
simulate each other.

Church–Turing Thesis:

“Any natural / reasonable notion of computation can be simulated by a TM.”

Describing Turing Machines

Low Level:

Medium Level:

High Level:

Super-high Level:

Just describe an algorithm / pseudocode.

Assuming the Church–Turing Thesis

(which everybody does)

there exists a TM which executes that algorithm.

Question:

Is every language in $\{0,1\}^*$ decidable?

$\Leftrightarrow$ Is every function $f : \{0,1\}^* \rightarrow \{0,1\}$ computable?

Answer: No!

Every TM is encodable by a finite string.

Therefore the set of all TM's is countable.

So the subset of all *decider* TM's is countable.

Thus the set of all decidable languages is countable.

But the set of **all** languages is uncountable.

(from last lecture, $|P(\{0,1\}^*)| > |\{0,1\}^*|$)

Question:

Is it just weirdo languages that no one would care about which are *undecidable*?

Answer (due to Turing, 1936):

Sadly, no.

There are some very reasonable languages we'd like to compute which are undecidable.

Some uncomputable functions

Given two TM descriptions, $\langle M_1 \rangle$ and $\langle M_2 \rangle$, do they act the same (accept/reject/loop) on all inputs?

Given the description of an algorithm, $\langle M \rangle$, does it print out "HELLO WORLD"?

```
main(t,_,a ) char * a; { return!  0<t? t<3?  main(-79,-13,a+ main(-87,1-_, main(-86, 0, a+1 )
+a)):  1, t<_? main( t+1, _, a ) :3,  main ( -94, -27+t, a ) &&t == 2 ?_ <13 ?  main ( 2, _+1, "%s
%d %d\n" )  :9:16: t<0? t<-72? main( _, t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#;#q#n+,/+k#;*+,/'r
:'d*'3,}{w+K w'K:'+}e#';dq#'l
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# ){nl]!/n{n#'; r{#w'r
nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' iwk{KK{nl]!/w{%'l##w#' i;
:{nl]'/*{q#'ld;r'}{nlwb!/*de}'c ;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n'
')# }'+}##(!!/") : t<-50? _==*a ? putchar(31[a]):  main(-65,_,a+1) : main((*a == '/') + t, _, a + 1 ) :
0<t?  main ( 2, 2 , "%s") :*a=='/'||  main(0,  main(-61,*a, "!ek;dc i@bK'(q)-
[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry")  ,a+1);}
```

This C program prints out all the lyrics of *The Twelve Days Of Christmas*.

Does the following program (written in Maple) print out “HELLO WORLD” ?

```
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
print(“HELLO  WORLD”)
```

It does so if and only if “Goldbach’s Conjecture” is false.

Some uncomputable functions

Given two TM descriptions, $\langle M_1 \rangle$ and $\langle M_2 \rangle$, do they act the same (accept/reject/loop) on all inputs?

Given the description of an algorithm, $\langle M \rangle$, does it print out "HELLO WORLD"?

Given a TM description $\langle M \rangle$ and an input x, does M halt on input x?

Given a TM description $\langle M \rangle$, does M halt when the input is a blank tape?

Some uncomputable functions

This one is called
The Halting Problem.

Given a TM description ⟨M⟩ and an input x,
does M halt on input x?

Turing's Theorem:
The Halting Problem is undecidable.

The Halting Problem is Undecidable

Theorem:

Let HALTS $\subseteq \{0,1\}^*$ be the language

$\{ \langle M,x \rangle$: M is a TM which halts on input x $\}$.

Then HALTS is undecidable.

Proof:

Assume for the sake of contradiction that

M_{HALTS} is a decider TM which decides HALTS.

The Halting Problem is Undecidable

Here is the (super-high level) description of another TM called D, which uses M_{HALTS} as a subroutine:

D:

Given as input $\langle M \rangle$, the encoding of a TM M:

D executes $M_{HALTS}(\langle M, \langle M \rangle \rangle)$.

If this call accepts, D enters an infinite loop.

If this call rejects, D halts (say, it accepts).

In other words… $D(\langle M \rangle)$ loops if $M(\langle M \rangle)$ halts,

halts if $M(\langle M \rangle)$ loops.

The Halting Problem is Undecidable

Assume M_{HALTS} is a decider TM which decides HALTS.

We can use it to construct a machine D such that

$$D(\langle M\rangle) \quad \text{loops} \quad \text{if } M(\langle M\rangle) \text{ halts,}$$
$$\text{halts} \quad \text{if } M(\langle M\rangle) \text{ loops.}$$

Time for the contradiction:

Does D(⟨D⟩) loop or halt?

By definition, if it loops it halts and if it halts it loops.

Contradiction. $\square$

BTW: last part of proof basically the same as Cantor's Diagonal Argument.

$D(\langle M \rangle)$ loops if $M(\langle M \rangle)$ halts, halts if $M(\langle M \rangle)$ loops

The set of all TM's is countable, so list it:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	…
M_1	halts	halts	loops	halts	loops	
M_2	loops	loops	loops	loops	loops	
M_3	halts	loops	halts	halts	halts	
M_4	halts	halts	halts	halts	loops	
M_5	halts	loops	loops	halts	loops	
□						

How could D be on this list?

What would the diagonal entry be??

D(⟨M⟩) loops if M(⟨M⟩) halts, halts if M(⟨M⟩) loops

The set of all TM's is countable, so list it:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\cdots$
M_1	halts	halts	loops	halts	loops	
M_2	loops	loops	loops	loops	loops	
M_3	halts	loops	halts	halts	halts	
M_4	halts	halts	halts	halts	loops	
M_5	halts	loops	loops	halts	loops	
$\vdots$						

Given some code,
determine if it terminates.

It’s not: “we don’t know how to solve it efficiently”.

It’s not: “we don’t know if it’s a solvable problem”.

We know that it is ***unsolvable by any algorithm****.*

In our proof that HALTS is undecidable,
we used a *hypothetical* TM deciding HALTS to derive a contradiction

Having established the undecidability of HALTS, we can show further problems to be undecidable using the powerful tool of

REDUCTIONS

Reductions

*Using one problem as a **subroutine***
to solve another problem.

Informally, a reduction from A to B
gives a way to solve problem A using a
subroutine that can solve B

Calculating the area of a rectangle reduces to calculating its length and height.

Solving a linear system $Ax = b$ reduces to computing the matrix inverse A^{-1}

Reductions

Language A *reduces to* language B means (informally):

"there is a method that could be used to solve A
if it has available to it a subroutine for solving B."

The *reduction* gives such a method.

Formally a **(Turing) reduction** from A to B is an *oracle* Turing machine that decides A when run with an oracle for B.

Notation for A reduces to B:

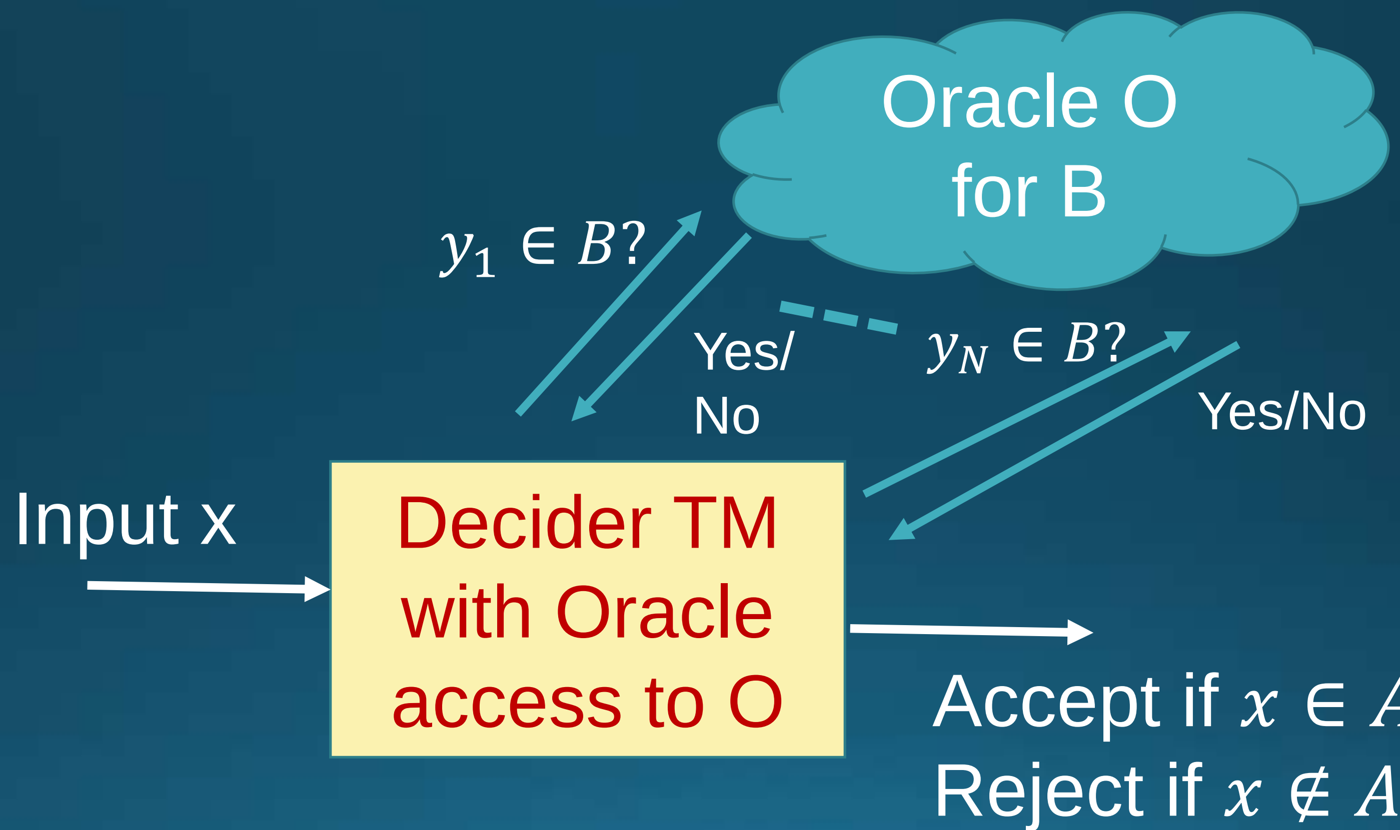
$A \leq_T B$ (T stands for Turing).

Think,

"A is no harder than B"

"A is at least as easy as B"

Reducing language A to B



Reductions

Fact: Suppose $A \leq_T B$; i.e., A reduces to B.

If B is decidable, then A is also decidable.

(can replace the assumed oracle for B with a decider for B, and the reduction can run this decider whenever it needs to ascertain membership of some string in B)

Contrapositive: ***if A is undecidable then so is B.***

Think: “B is at least as hard as A”

Reductions are ***the*** main technique for showing undecidability.

Reductions — examples

Theorem:

ACCEPTS = {⟨M, x⟩ : M is a TM which accepts x}

is undecidable.

Proof: We'll prove HALTS reduces to ACCEPTS.

Suppose $O_{ACCEPTS}$ is an oracle for language ACCEPTS.

Then here's a description of an oracle TM deciding HALTS:

"Given ⟨M, x⟩, run $O_{ACCEPTS}$(⟨M, x⟩). If it accepts, then accept.

Reverse the accept & reject states in ⟨M⟩, forming ⟨M′⟩.

Run $O_{ACCEPTS}$(⟨M′, x⟩). If it accepts (i.e., M rejects x), then accept.

Else reject."

Interesting observation

To prove a **negative** result about computation (that a certain language is undecidable),

you actual **construct an algorithm** –
namely, the reduction.

Reductions — another example

Theorem: EMPTY = {⟨M⟩ : M accepts no strings}
is undecidable.

Proof: Let's prove ACCEPTS $\leq_T$ EMPTY.

This suffices, since we just showed ACCEPTS is undecidable.

So suppose O_{EMPTY} is an oracle for language EMPTY.

Here's an oracle TM with oracle access to O_{EMPTY} deciding ACCEPTS:

"Given ⟨M, x⟩…

Write down the **description** ⟨N_x⟩ of a TM N_x which does the following:

"On input y, check if y=x.

If not, reject. If so, simulate M on y."

Then call upon the oracle O_{EMPTY} on input ⟨N_x⟩ and *do the opposite.*"

Correctness of reduction

Code for N_x :

"On input y,
check if y=x.
If not, reject.
If so, simulate M on y."

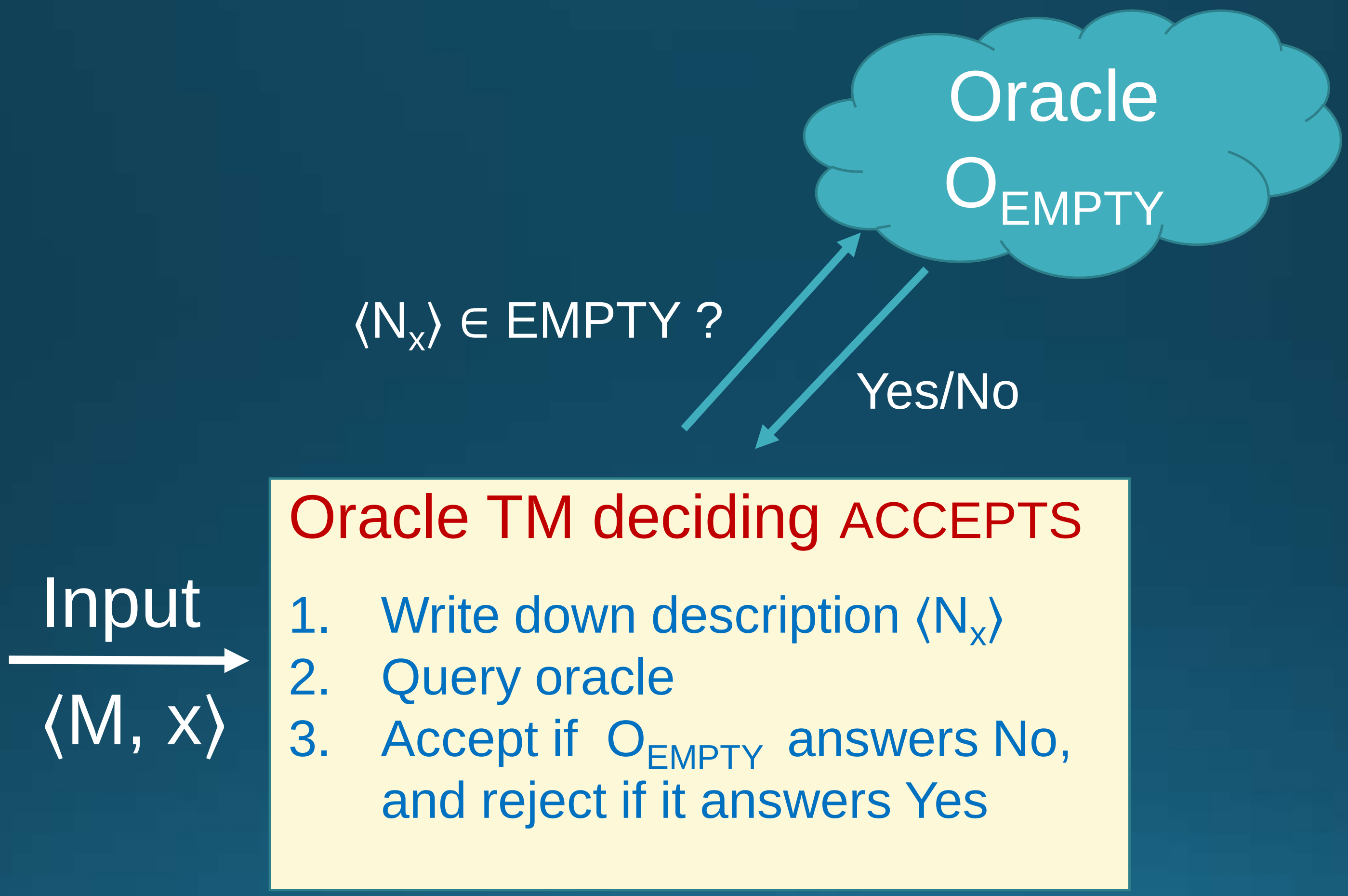
$L(N_x)$ is either {x} or Ø

And $L(N_x) = \{x\}$ precisely when M accepts x, i.e., $\langle M, x\rangle \in$ ACCEPTS

Important:

Reduction ***never*** runs N_x ;
it simply writes down the description $\langle N_x\rangle$ of N_x
and probes the oracle whether $\langle N_x\rangle \in$ EMPTY

Schematic of the reduction ACCEPTS $\leq_T$ EMPTY



Another example:

ACCEPTS $\leq_T$ INFINITE = {⟨M⟩ : M accepts infinitely many strings}

Oracle $O_{INFINITE}$

⟨I_x⟩ ∈ INFINITE ?

Yes/No

Input

⟨M, x⟩

Oracle TM deciding ACCEPTS

1. Write down description ⟨I_x⟩
2. Query oracle on ⟨I_x⟩
3. Accept if $O_{INFINITE}$ answers Yes, and reject if it answers No

Code ⟨I_x⟩:
"On input y:
- ignore y
- Run M on x

& accept if it does."

Remember:
we don't run I_x,
we only write down it's code

Note: Reduction is *particularly simple:*
a single oracle query, and
we just pass on answer to that query.
Called "mapping reduction"

Undecidability galore

Similar reductions can show undecidability of telling if, given an input TM $\langle M \rangle$, L(M) is:

Finite
Regular
Contains 15251 in binary
Decidable
Contains a string of length more than 15251
Etc etc

Essentially any non-trivial property of languages

Question:

Do all undecidable problems involve TM’s?

Answer:

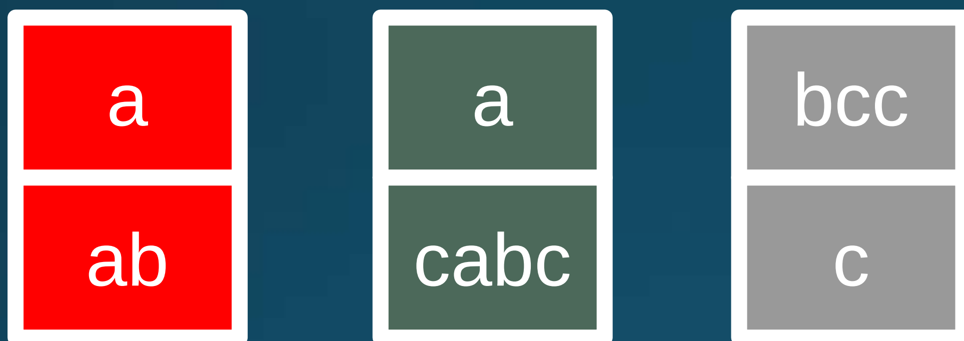
No!

Some very different problems are undecidable!

Post's Correspondence Problem

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

E.g.:

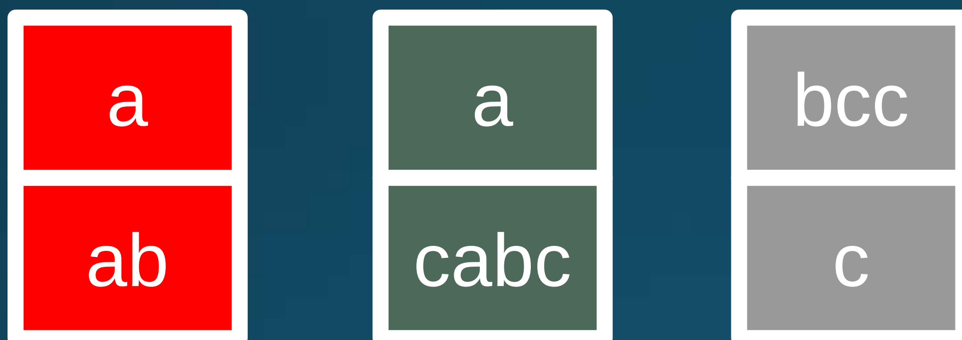


Definition: A match is a sequence of dominoes,
repetitions allowed, such that
top string = bottom string.

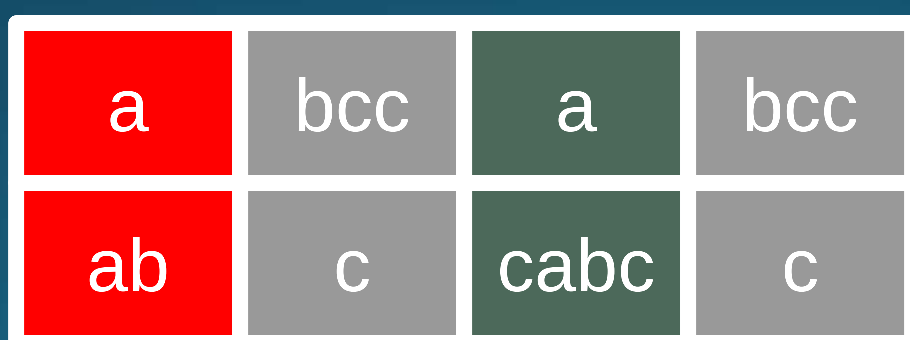
Post’s Correspondence Problem

Input: A finite collection of “dominoes”, having strings written on each half.

E.g.:



Match:



= abccabcc

= abccabcc

Post's Correspondence Problem

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

Task: Output YES if and only if there is a match.

Theorem (Post, 1946): **Undecidable**.
There is no algorithm solving this problem.

(More formally, PCP = {⟨Domino Set⟩ : there's a match}
is an undecidable language.)

Post's Correspondence Problem

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

Task: Output YES if and only if there is a match.

Theorem (Post, 1946): **Undecidable**. There is no algorithm solving this problem.

Two-second proof sketch:

Given a TM M, you can make a domino set such that the only matches are execution traces of M which end in the accepting state. Hence ACCEPTS $\leq_T$ PCP.

Wang Tiles

Input: Finite collection of “Wang Tiles” (squares) with colors on the edges. E.g.,

Task: Output YES if and only if it’s possible to make an infinite grid from copies of them, where touching sides must color-match.

Theorem (Berger, 1966): **Undecidable.**

Modular Systems

Input: Finite set of rules of the form
"from $ax+b$, can derive $cx+d$", where $a,b,c,d \in \mathbb{Z}$.
Also given is a starting integer u and a target v.

Task: Decide if v can be derived starting from u.

E.g.: "from $2x$ derive x", "from $2x+1$ derive $6x+4$",
target $v = 1$. Starting from u, this is equivalent
to asking if the "3n+1 problem" halts on u.

Theorem (Börger, 1989): **Undecidable.**

Mortal Matrices

Input: Two 15×15 matrices of integers, A & B.

Question: Is it possible to multiply A and B together (multiple times in any order) to get the 0 matrix?

Theorem (Cassaigne, Halava, Harju, Nicolas, 2014):

Undecidable.

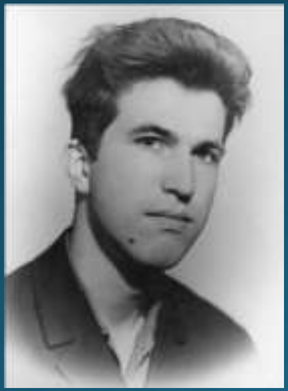
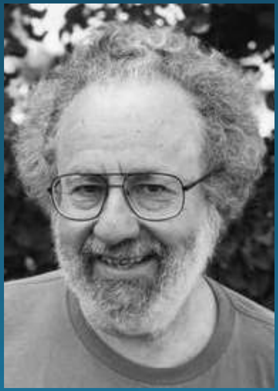
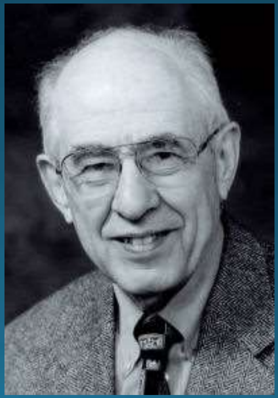
Hilbert's 10^{th} problem

Input: Multivariate polynomial w/ integer coeffs.

Question: Does it have an integer root?

Theorem (1970): **Undecidable.**

Matiyasevich Robinson Davis Putnam

Hilbert's 10^{th} problem

Input: Multivariate polynomial w/ integer coeffs.

Question: Does it have an integer root?

Undecidable.

Question: Does it have a real root?

Decidable.

Tarski, 1951.

Question: Does it have a rational root?

Not known if it's decidable or not.

Entscheidungsproblem

Input: A sentence in first-order logic.

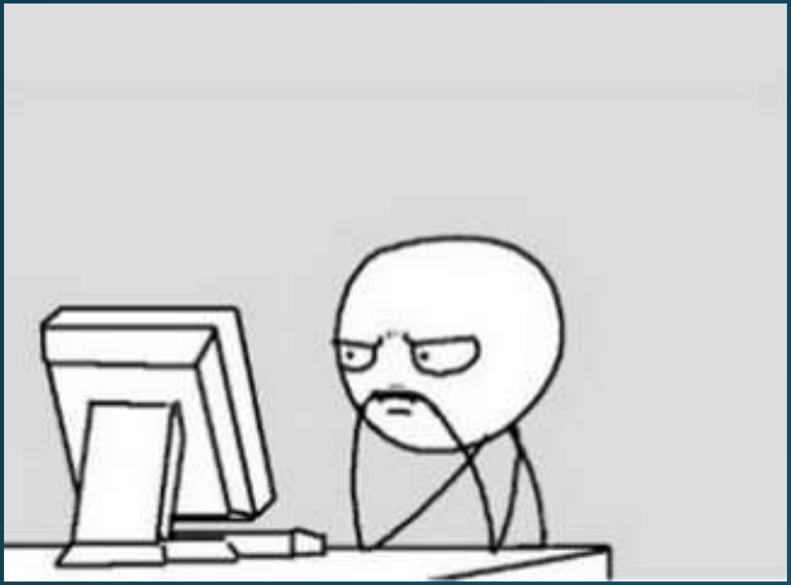
$$\neg \exists n, x, y, z \in N: (n \geq 3) \wedge (x^n + y^n = z^n)$$

Question: Is it provable?

This is undecidable.

We'll come back to this in the
lecture on Gödel's Incompleteness Theorem

Possible discussion of
Post Correspondence Problem
undecidability



Study Guide

Definitions:

Decidable languages/ computable functions

Undecidable languages

Halting Problem

Theorems/proofs:

Halting Problem is undecidable

Undecidability proofs via reductions

Practice:

Reductions