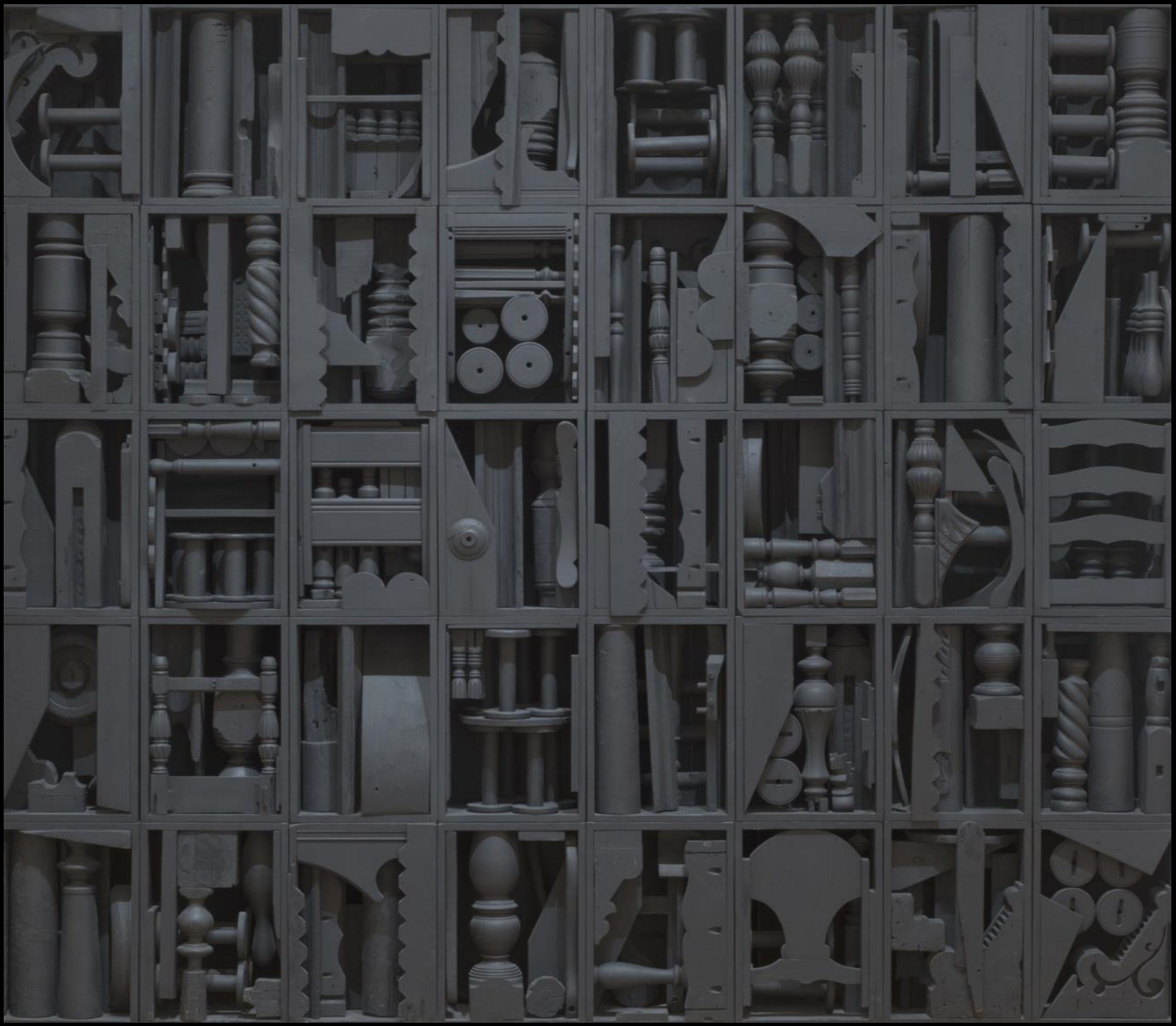


Decidability

Contents

- Decidable Languages
- Undecidability

Decidability

• We previously studied abstract models for general-purpose computing, including the *Turing Machine* (TM). The *Church-Turing Thesis* encapsulates the equivalence of intuitive, procedural processes (i.e. algorithms) and algorithms run on TMs.

• Despite their inherent computational power and potential for generalizability, there exist problems (note the use of plural) that TMs – and moreover *any* procedural algorithms – **cannot solve**. This means, somewhat surprisingly, that some problems cannot be solved *algorithmically*.

• In fact, as we shall see, **there are uncountably many undecidable problems** (and only countably many decidable problems).

Decidability

Why study undecidability and unsolvable problems?

(1) Knowing/recognizing that a problem is undecidable/unsolvable is useful, as it informs us that a simplification or approximation is necessary.

(2) Understanding unsolvable problems gives us an important, "high-level" perspective about computation and algorithms more generally; oftentimes this broader understanding informs our ability to solve practical computational and algorithmic problems.

Decidable Languages

- We define the **acceptance problem for DFA** (deterministic finite automata) as testing whether a particular DFA accepts a given string.

- The acceptance problem for DFA gives rise to a language; this language contains the <u>encodings of all DFAs together with strings that the DFA accepts</u>. Define:

Decidable Languages

- We define the **acceptance problem for DFA** (deterministic finite automata) as testing whether a particular DFA accepts a given string.

- The acceptance problem for DFA **gives rise to a language**; this language contains the <u>encodings of all DFAs together with strings that the DFA accepts</u>. Define:

$$\boxed{A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}}$$

- Notice that the problem of testing whether a DFA B accepts an input w is the same as the problem of testing whether $\langle B, w\rangle$ is a member of the language.

Decidable Languages

• We define the **acceptance problem for DFA** (deterministic finite automata) as testing whether a particular DFA accepts a given string.

• The acceptance problem for DFA gives rise to a language; this language contains the encodings of all DFAs together with strings that the DFA accepts. Define:

$$\boxed{A_{DFA} = \{\langle B, w\rangle | B \text{ is a } DFA \text{ that accepts input string } w\}}$$

• Notice that the problem of testing whether a DFA B accepts an input w is the same as the problem of testing whether $\langle B, w\rangle$ is a member of the language.

(*) **In general, showing that a language is decidable is the same as showing the computational problem is decidable**.

Decidable Languages

Theorem. A_{DFA} is a decidable language.

Decidable Languages

Theorem. A_{DFA} is a decidable language.

Proof. We simply construct a TM *M* that decides A_{DFA}.

Define *M*: On input $\langle B, w\rangle$, where B is a DFA and w is a string:

(1) Simulate B on input w

(2) If the simulation ends in an accept state, *accept*. If it ends in a non-accepting state, *reject*.

Decidable Languages

Theorem. A_{DFA} is a decidable language.

Proof. We simply construct a TM *M* that decides A_{DFA}.

Define *M*: On input $\langle B, w\rangle$, where B is a DFA and w is a string:

(1) Simulate B on input w

(2) If the simulation ends in an accept state, *accept*. If it ends in a non-accepting state, *reject*.

- A few comments: The input $\langle B, w\rangle$ is a representation of a DFA B together with a string w. One reasonable representation of B is through the formalization with respect to components: Q, Σ, δ, q_0, and F. When *M* receives its input, *M* first determines whether it properly represents a DFA B and a string w; if not, M rejects.

M carries out the simulation directly; when it finishes processing the last symbol of w, *M* accepts the input if B is in an accepting state.

Decidable Languages

Define:

$$A_{NFA} = \{\langle B, w\rangle | B \text{ is an NFA that accepts input string } w\}$$

Theorem. A_{NFA} is a decidable language.

Decidable Languages

Define:

$$A_{NFA} = \{\langle B, w\rangle | B \text{ is an } NFA \text{ that accepts input string } w\}$$

Theorem. A_{NFA} is a decidable language.

Proof. We present a TM N that decides A_{NFA}.

N: On input $\langle B, w\rangle$, where B is an NFA and w a string,

(1) Convert NFA B to an equivalent DFA C
(2) Run TM M from the previous Theorem on input $\langle C, w\rangle$
(3) If M accepts, **accept**; otherwise **reject**.

Decidable Languages

Define:

$$A_{NFA} = \{\langle B, w\rangle | B \text{ is an NFA that accepts input string } w\}$$

Theorem. A_{NFA} is a decidable language.

Proof. We present a TM N that decides A_{NFA}.

N: On input $\langle B, w\rangle$, where B is an NFA and w a string,

(1) Convert NFA B to an equivalent DFA C

(2) Run TM M from the previous Theorem on input $\langle C, w\rangle$

(3) If M accepts, **accept**; otherwise **reject**.

*Note that one can similarly show that the following language is decidable.

$$A_{REX} = \{\langle R, w\rangle | \ R \text{ is an regular expression generates string } w\}$$

Decidable Languages

Define:

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a } DFA \text{ and } L(A) = \emptyset\}$$

Theorem. E_{DFA} is a decidable language.

Decidable Languages

Define:

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a } DFA \text{ and } L(A) = \emptyset\}$$

Theorem. E_{DFA} is a decidable language.

Proof. A DFA accepts some string *iff* reaching an accept state from the start state by traveling along the arrows of the DFA is possible.

We simply design a TM T using a "marking algorithm" (just see if any directed paths lead from the start state to an accept states), as follows:

T: On input $\langle A \rangle$, where A is a DFA:

(1) Mark the start state of A.

(2) Repeat until no new states get marked…mark any state that has a transition coming into it from any state that is already marked.

(3) If no accept state is marked, **accept**; otherwise **reject**.

Decidable Languages

Define:

$$EQ_{DFA} = \{\langle A, B\rangle \mid A \textit{ and } B \textit{ are } DFA \textit{ and } L(A) = L(B)\}$$

Theorem. EQ_{DFA} is a decidable language.

Decidable Languages

Define:

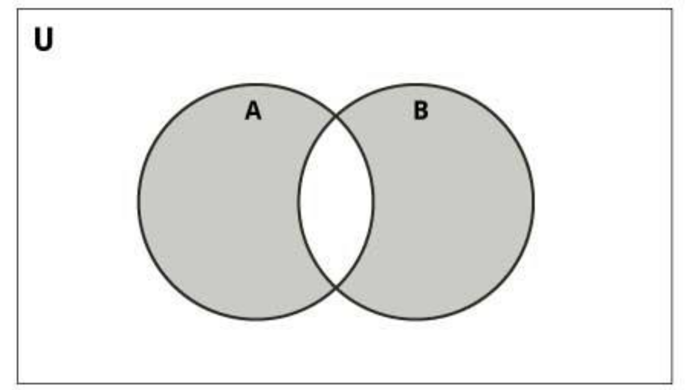
$$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are } DFA \text{ and } L(A) = L(B)\}$$

Theorem. EQ_{DFA} is a decidable language.

- We will prove this result next. First, however, a quick aside:

- The **symmetric difference** of two sets A and B is defined as the set of elements that are in A or B but not both (think of the analogue with the XOR operation):

$$\boxed{A \,\Delta\, B = (A \setminus B) \cup (B \setminus A)}$$



Notice that: $A \,\Delta\, B = (A \cap \bar{B}) \cup (\bar{A} \cap B)$

*Note that the symmetric difference of two languages is defined equivalently.

Decidable Languages

Define:

$$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are } DFA \text{ and } L(A) = L(B)\}$$

Theorem. EQ_{DFA} is a decidable language.

Proof. We construct a new DFA C, where $\mathrm{L(C)} = L(A)\,\Delta\, L(B)$, which is to say C accepts the symmetric difference of the languages of A and B. Notice, importantly that $\mathrm{L(C)} = \emptyset$ *iff* $\mathrm{L}(A) = L(B)$.

How to proceed?

Decidable Languages

Define:

$$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are } DFA \text{ and } L(A) = L(B)\}$$

Theorem. EQ_{DFA} is a decidable language.

Proof. We construct a new DFA C, where $\mathrm{L(C)} = L(A)\,\Delta\,L(B)$, which is to say C accepts the symmetric difference of the languages of *A* and *B*. Notice, importantly that $\mathrm{L(C)} = \emptyset$ *iff* $\mathrm{L}(A) = L(B)$.

Because the symmetric difference involves complement, union and intersection operations (see previous slides) – and in addition, **regular languages are closed under these operations** – we can simply construct at TM *M* that runs *C* as input.

From the previous Theorem (E_{DFA} is decidable), if *M* accepts, **accept**; otherwise **reject**.

Decidable Languages

Define:

$$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a } CFG \text{ that generates } w\}$$

$\boxed{\textbf{Theorem}.\ A_{CFG} \text{ is a decidable language.}}$

Decidable Languages

Define:

$$A_{CFG} = \{\langle G, w\rangle \mid G \text{ } is\ a\ CFG\ that\ generates\ w\}$$

Theorem. A_{CFG} is a decidable language.

- One idea (though incorrect) is to try to work through all derivations in *G* to see whether any produce *w*. This won't work, naturally, because infinitely-many derivations may need to be attempted (and thus the algorithm won't halt).

Decidable Languages

Define:

$$A_{CFG} = \{\langle G, w \rangle \mid G \text{ is a CFG that generates } w\}$$

Theorem. A_{CFG} is a decidable language.

- (Proof sketch) To make this TM a **decider**, we need to ensure that the algorithm tries only a finite number of derivations. It can be shown (we omit proof for brevity) that when G is in *Chomsky Normal Form* (CNF), any derivations of $|w| = n$ has $2n - 1$ steps.

So the TM S to decide A_{CFG}, when given input $\langle G, w \rangle$, converts G to CNF, lists all derivations using $2n - 1$ steps and checks to see if any generate w.

Decidable Languages

Define:

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a } CFG \text{ and } L(G) = \emptyset\}$$

Theorem. E_{CFG} is a decidable language.

Decidable Languages

Define:

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a } CFG \text{ and } L(G) = \emptyset\}$$

Theorem. E_{CFG} is a decidable language.

- A tempting – but again, ultimately incorrect – approach is to enumerate all strings w and rely on the decidability of:

$$A_{CFG} = \{\langle G, w \rangle \mid G \text{ is a } CFG \text{ that generates } w\}$$

to determine whether $L(G) = \emptyset$. Of course, this is an unsound approach, why?

Decidable Languages

Define:

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a } CFG \text{ and } L(G) = \emptyset\}$$

Theorem. E_{CFG} is a decidable language.

• Instead, we approach the problem is a fashion similar to the technique employed for the proof the decidability of E_{DFA}.

Proof idea: First, we "mark" all terminal symbols in G; next we recursively mark any variable A where G has a rule A $\rightarrow$ $U_1 U_2 \dots U_k$ and each symbol $U_1 U_2 \dots U_k$ has already been marked.

If the start variable is not marked, **accept**; otherwise **reject**.

Decidable Languages

Define:

$$EQ_{CFG} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are } CFGs \text{ and } L(G) = L(H)\}$$

Theorem. EQ_{CFG} is a NOT a decidable language.

- Previously we proved that the comparable problem for DFA, EQ_{DFA} is decidable. What was the key insight for this proof?

Decidable Languages

Define:

$$EQ_{CFG} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$$

Theorem. EQ_{CFG} is <u>NOT</u> a decidable language.

- Previously we proved that the comparable problem for DFA, EQ_{DFA} is decidable. What was the key insight for this proof?

- We relied on the fact that E_{DFA} is decidable and that when $\mathrm{L(C)} = L(A) \,\Delta\, L(B)$ (i.e. the symmetric difference of the languages of A and B) $\mathrm{L(C)} = \emptyset$ *iff* $\mathrm{L}(A) = L(B)$. The proof follows immediately by construction.

- Notice, though, <u>**that we cannot use this approach for CFGs**</u>!

Decidable Languages

Define:

$$EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$$

Theorem. EQ_{CFG} is <u>NOT</u> a decidable language.

• Previously we proved that the comparable problem for DFA, EQ_{DFA} is decidable. What was the key insight for this proof?

• We relied on the fact that E_{DFA} is decidable and that when $\mathrm{L(C)} = L(A) \,\Delta\, L(B)$ (i.e. the symmetric difference of the languages of A and B) $\mathrm{L(C)} = \emptyset$ *iff* $\mathrm{L}(A) = L(B)$. The proof follows immediately by construction.

• Notice, though, <u>**that we cannot use this approach for CFGs**</u>!

• The problem is that we relied on the fact that DFA are closed under regular operations (in addition to complement) – and CFGs do not obey these closure properties. In particular, **CFGs are not closed under intersection or complement**. We show the full method of proof in the next chapter.

Decidable Languages

Let's summarize the aforementioned results regarding decidability:

All Decidable

$$A_{DFA} = \{\langle B, w\rangle | B \text{ is a DFA that accepts input string } w\}$$

$$A_{NFA} = \{\langle B, w\rangle | B \text{ is an NFA that accepts input string } w\}$$

$$E_{DFA} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

$$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFA and } L(A) = L(B)\}$$

$$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\}$$

$$E_{CFG} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Undecidable

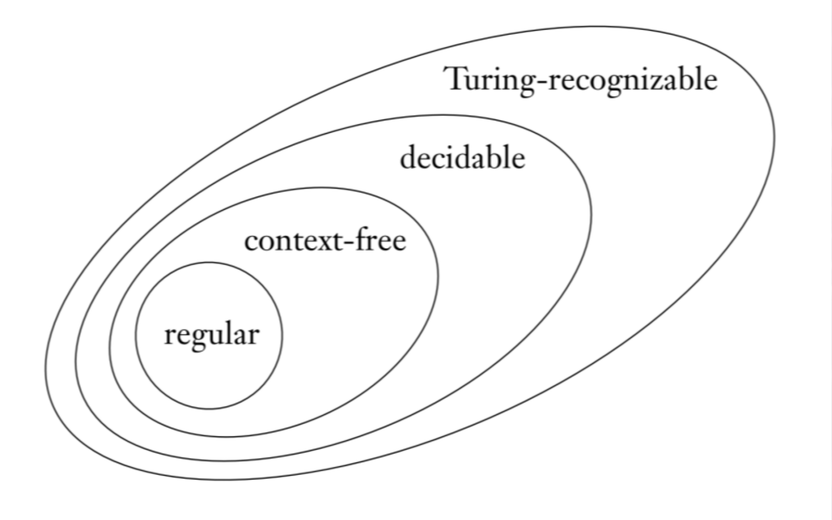
$$EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$$

Undecidability

• We now explore one of the most philosophically important ideas in the theory of computation, undecidability.

• On the surface, computers often appear to be so powerful that we may believe that all problems eventually yield to them. This is however far from the truth.

• We now show that even computers have a fundamental limitation insofar as there exist problems that are algorithmically unsolvable (i.e. undecidable).



Undecidability

Define:

$$A_{TM} = \{\langle M, w \rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w\}$$

Theorem. A_{TM} is Turing-Recognizable

Undecidability

Define:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w\}$$

Theorem. A_{TM} is Turing-Recognizable

Proof. Define *U* (a **universal Turing Machine**, i.e. it can simulate any other TM) that recognizes A_{TM}.

U: On input $\langle M, w\rangle$, where M is a TM and w is a string,

(1) Simulate M on w

(2) If M ever enters its accept state, **accept**; if M ever enters its reject state, **reject**.

*Notice that it is entirely possible that *U* loops forever – but this issue doesn't directly affect the ability of U to recognize A_{TM}.

Undecidability

Recall from our previous discussions regarding Cantor's methods, the following fundamental definitions and concepts:

- Two sets are **equinumerous** if there exists a bijection between them; we say that a set is **countable** if it is finite or equinumerous with $\mathbb{N}$; we say that a set is **uncountable** if its cardinality is strictly greater than $\aleph_0$.

- If $\mathrm{A} \subsetneq B,\ and\ |A| < \infty$, then $|A| < |B|$.

- Conversely, if $\mathrm{A} \subsetneq B$ and $|A| = \infty$ then $|A| \leq |B|$.

- $|\mathbb{N}| = |\mathbb{Z}| = |\mathbb{Q}| = \aleph_0$

- However, using the *diagonalization technique*, it follows that:

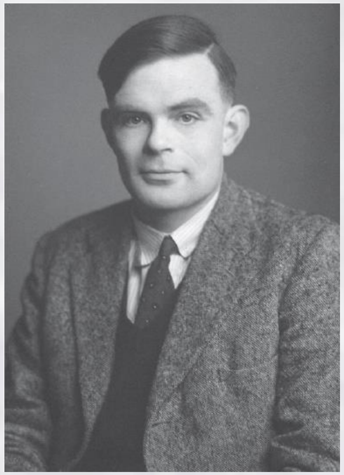
$$|\mathbb{Q}| < |\mathbb{R}|$$

- For any set A, $|A| < |P(A)|$.

Undecidability

Theorem. There are only countably-many TMs

Undecidability

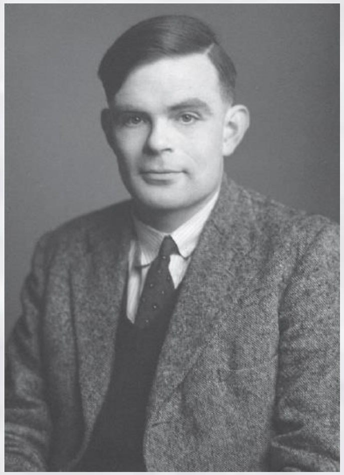


Theorem. There are only countably-many TMs

- This astounding result is due to Turing and was proven in his seminal 1936 paper.

How did Turing show this? Fundamentally, Turing melded two prior notions (Gödel encodings + diagonalization) with his own novel formalization of algorithms and computational machines (TMs).

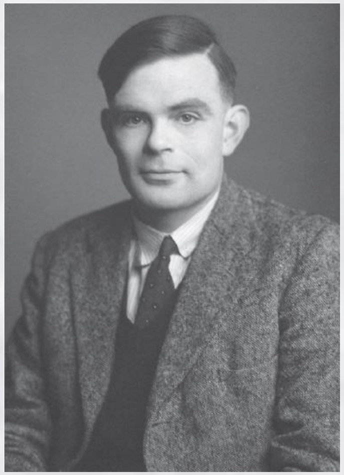
Undecidability



Theorem. There are only countably-many TMs

- Every TM M gives rise to a finite binary encoding (i.e. a binary string) which we denote $\langle M \rangle$. (Why is every $\langle M \rangle$ finite in length?)

Undecidability



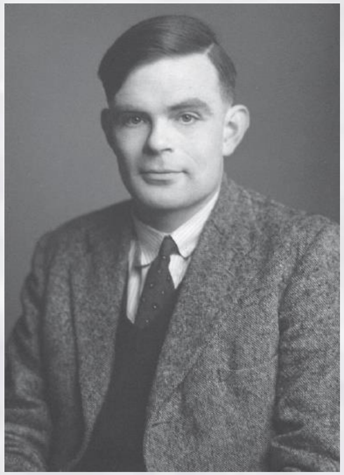
Theorem. There are only countably-many TMs

- Every TM M gives rise to a finite binary encoding (i.e. a binary string) which we denote $\langle M \rangle$.

- The set of all finite binary strings is countable. Why? (note that this different from the set of all countably infinite binary strings which results in an uncountable set! You proved this result in a homework problem)

- Thus there are only countably-many TMs.

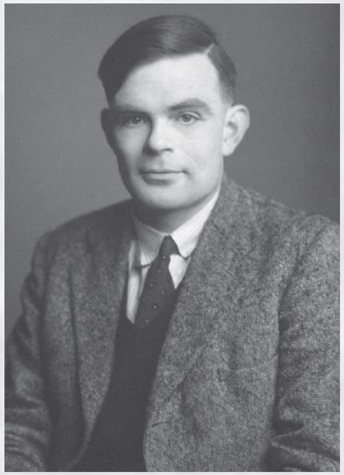
Undecidability



Theorem. The set of all language is uncountable, i.e. some languages are not Turing-recongizable).

Proof. As mentioned, the set of all infinite binary strings B is uncountable. We now derive a bijection between L, the set of all languages over alphabet Σ, and B.

Undecidability



Theorem. The set of all language is uncountable, i.e. some languages are not Turing-recongizable).

Proof. As mentioned, the set of all infinite binary strings B is uncountable. We now derive a bijection between L, the set of all languages over alphabet Σ, and B.

Let $\Sigma^* = \{s_1, s_2, s_3, \ldots.\}$. We show that each language $\mathrm{A} \in L$ has a unique sequence in B, yielding the necessary bijection. Define the characteristic sequence of A (χ_A) as follows: the *i*th bit of χ_A is 1 if $s_i \in A$ and 0 if $s_i \notin A$:

$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$$
$$A = \{\,0,00, 01, 000, 001, \ldots\}$$
$$\chi_A = \{\,0,1, 0,1,1,0,0,1,1, \ldots\}$$

This demonstrates $|L| = |B|$ and the result follows.

Undecidability

Define:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w\}$$

Theorem. A_{TM} is undecidable.

Undecidability

Define:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ } is \text{ } a \text{ } TM \text{ } and \text{ } M \text{ } accepts \text{ } w\}$$

Theorem. A_{TM} is undecidable.

Proof. Suppose not, and we assume on the contrary that A_{TM} is decidable; let H be a decider for A_{TM}.

So H is defined:

$$H(\langle M, w\rangle) = \begin{cases} accept \text{ } if \text{ } M \text{ } accepts \text{ } w \\ reject \text{ } if \text{ } M \text{ } does \text{ } not \text{ } accept \text{ } w \end{cases}$$

Undecidability

Define:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ } is \text{ } a \text{ } TM \text{ } and \text{ } M \text{ } accepts \text{ } w\}$$

Theorem. A_{TM} is undecidable.

Proof. Now we construct a new TM D with H as a subroutine:

D($\langle M\rangle$):

(1) Run H on input $\langle M, \langle M\rangle\rangle$

(2) Output the opposite of what H outputs. That is, if H accepts, **reject**; else **accept**.

Undecidability

Define:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w\}$$

Theorem. A_{TM} is undecidable.

Proof. Now we construct a new TM D with H as a subroutine:

D($\langle M\rangle$):

(1) Run H on input $\langle M, \langle M\rangle\rangle$

(2) Output the opposite of what H outputs. That is, if H accepts, **reject**; else **accept**.

*Key observation: By construction, neither D nor H can exist!

Undecidability

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w\}$$

Theorem. A_{TM} is undecidable.

Proof. *Key observation: By construction, neither D nor H can exist!

Recap: We assume TM H decides A_{TM}. We then use H to construct TM D that takes $\langle M\rangle$ as input, where D accepts its input $\langle M\rangle$ exactly when M does not accept its input $\langle M\rangle$. Finally, run D on itself:

- H accepts $\langle M, w\rangle$ exactly when M accepts w
- D rejects $\langle M\rangle$ exactly when M accepts $\langle M\rangle$
- D rejects $\langle D\rangle$ exactly when D accepts $\langle D\rangle$

Where does diagonalization come into play in this proof?

Undecidability

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w\}$$

Theorem. A_{TM} is undecidable.

Proof.

- H accepts $\langle M, w\rangle$ exactly when M accepts w
- D rejects $\langle M\rangle$ exactly when M accepts $\langle M\rangle$
- D rejects $\langle D\rangle$ exactly when D accepts $\langle D\rangle$

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\cdots$
M_1	*accept*		*accept*		
M_2	*accept*	*accept*	*accept*	*accept*	
M_3					$\cdots$
M_4	*accept*	*accept*			
$\vdots$		$\vdots$			

Entry i, j is *accept* if M_i accepts $\langle M_j\rangle$

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	
M_4	*accept*	*accept*	*reject*	*reject*	
$\vdots$			$\vdots$		

Entry i, j is the value of H on input $\langle M_i, \langle M_j\rangle\rangle$

Undecidability

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a } TM \text{ and } M \text{ accepts } w\}$$

Theorem. A_{TM} is undecidable.

Proof.

- H accepts $\langle M, w\rangle$ exactly when M accepts w
- D rejects $\langle M\rangle$ exactly when M accepts $\langle M\rangle$
- D rejects $\langle D\rangle$ exactly when D accepts $\langle D\rangle$

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\cdots$	$\langle D\rangle$	$\cdots$
M_1	<u>*accept*</u>	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	<u>*accept*</u>	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	<u>*reject*</u>	*reject*		*reject*	
M_4	*accept*	*accept*	*reject*	<u>*reject*</u>		*accept*	
$\vdots$		$\vdots$			$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		<u>?</u>	
$\vdots$		$\vdots$					$\ddots$

Undecidability

- We just exhibited a language, A_{TM}, that is undecidable. Now we explore a language that <u>isn't even Turing-recognizable</u>.

- Recall that the complement of a language is the language consisting of all strings that are not in the language. We say that a language is **co-Turing-recognizable** if it is the complement of a Turing-recognizable language.

Undecidability

- We say that a language is **co-Turing-recognizable** if it is the complement of a Turing-recognizable language.

Theorem. A language is decidable *iff* it is Turing-recognizable and co-Turing-recognizable.

Proof. (→)Suppose that language A is decidable. It follows that both A and $\bar{A}$ are Turing-recognizable.

This follows because <u>any decidable language is automatically Turing-recognizable</u>; furthermore, <u>the complement of a decidable language is also decidable</u> – why?

Undecidability

Theorem. A language is decidable *iff* it is Turing-recognizable and co-Turing-recognizable.

Proof. (←) Suppose that language A Turing-recognizable and co-Turing-recognizable. Let M_1 be the recognizer for A and M_2 the recognizer for $\bar{A}$. Then the following TM M decides A :

M on input w:

(1) Run both M_1 and M_2 on input w in parallel (e.g. run on two tapes)

(2) If M_1 accepts, **accept**; if M_2 accepts, **reject**.

Note that M decides A; every string w is either in A or $\bar{A}$. Thus either M_1 or M_2 accepts w; <u>M always halts</u> and so it is a decider for A, as was to be shown.

Undecidability

Corollary. $\overline{A_{TM}}$ is not Turing-recognizable.

Proof. We know that A_{TM} is Turing-recognizable. If $\overline{A_{TM}}$ also were Turing-recognizable, then, by the previous Theorem A_{TM} would be decidable.

But we have previously demonstrated that A_{TM} is not decidable, hence, $\overline{A_{TM}}$ is not Turing-recognizable, as was to be shown.

Fin