CS244: Theory of Computation

Fu Song
ShanghaiTech University

Fall 2020

Outline

Decidability

Outline

Decidable Languages

A problem is decidable if

the language corresponding to the problem is recursive.

Otherwise, the problem is undecidable.

Decidable problems concerning regular languages

Membership problem of regular language

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

That is, for every $w \in \Sigma^*$ and DFA B,

$$w \in L(B) \iff \langle B, x\rangle \in A_{\text{DFA}}.$$

Theorem

A_{DFA} is a decidable language.

M on $\langle B, w\rangle$:

1. Simulate B on input w.
2. If the simulation ends in an accepting state, then accept. If it ends in a nonaccepting state, then reject.

Proof

Some implementation details:

- The representation of B is a list of 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.
- Then M carries out the simulation directly.
  1. It keeps track of B's current state and position in w by writing this information down on its tape.
  2. Initially, B's current state is q_0 and current input position is the leftmost symbol of w.
  3. The states and position are updated according to the specified transition function δ.
  4. When M finishes processing the last symbol of w, M accepts the input if B is in an accepting state; M rejects the input if B is in a nonaccepting state.

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

That is, for every $w \in \Sigma^*$ and NFA B,

$$w \in L(B) \iff \langle B, x\rangle \in A_{\text{NFA}}.$$

Theorem

A_{NFA} is a decidable language.

Proof

The simplest proof is to simulate an NFA using nondeterministic Turing machine, as we used the (deterministic) Turing machine M to simulate a DFA.

Instead we design a (deterministic) Turing machine N which uses M as a subroutine.

N on $\langle B, w \rangle$:

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

$$A_{\text{REX}} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates } w\}.$$

Theorem

A_{REX} is a decidable language.

P on $\langle R, w\rangle$:

1. Convert R to an equivalent NFA A.
2. Run TM N from the previous Theorem on input $\langle A, w\rangle$.
3. If N accepts, then accept; otherwise reject.

Emptiness Problem of Regular Language

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

Theorem

E_{DFA} is a decidable language.

A DFA accepts some string if and only if reaching an accept state from the start state by traveling along the arrows of the DFA is possible.

T on $\langle A \rangle$:

1. Mark the start state of A.
2. Repeat until no new states get marked:
3. Mark any state that has a transition coming into it from any state that is already marked.
4. If no accept state is marked, then accept; otherwise, reject.

Can be generalized to test emptiness of regular expressions and NFAs.

Testing Equality of two Regular Languages

$$EQ_{\text{DFA}} = \left\{ \langle A, B \rangle \;\middle|\; A \text{ are } B \text{ are DFAs and } L(A) = L(B) \right\}.$$

Theorem

EQ_{DFA} is a decidable language.

Proof (1)

From A and B we construct a DFA C such that

$$L(C) = \Big(L(A) \cap \overline{L(B)}\Big) \cup \Big(\overline{L(A)} \cap L(B)\Big),$$

i.e., the symmetric difference between $L(A)$ and $L(B)$.

Then

$$L(A) = L(B) \quad \Longleftrightarrow \quad L(C) = \emptyset.$$

F on $\langle A, B \rangle$:

1. Construct DFA C from A and B.
2. Run TM T from the previous Theorem on input $\langle C \rangle$.
3. If T accepts, then accept; otherwise reject.

Reading: Chen Fu, Yuxin Deng, David N. Jansen, Lijun Zhang: On Equivalence Checking of Nondeterministic Finite Automata. SETTA 2017

Discussion

Language inclusion problem

$$A = \left\{ \langle B_1, B_2 \rangle \;\middle|\; B_1 \text{ and } B_2 \text{ are NFA} \wedge L(B_1) \subseteq L(B_2) \right\}.$$

From B_1 and B_2 we construct a NFA B such that

$$L(B) = L(B_1) \cap \overline{L(B_2)},$$

Then

$$L(B_1) \subseteq L(B_2) \quad \Longleftrightarrow \quad L(B) = \emptyset.$$

Discussion

Universality problem

$$A = \left\{ \langle B \rangle \;\middle|\; B \text{ is a NFA} \wedge L(B) = \Sigma^* \right\}.$$

From B we construct a DFA $\overline{B}$ such that

$$L(\overline{B}) = \Sigma^* \setminus L(B),$$

Then

$$L(B) = \Sigma^* \quad \Longleftrightarrow \quad L(\overline{B}) = \emptyset.$$

Quiz

$A = \{\langle B_1, B_2, w\rangle \mid B_1 \text{ and } B_2 \text{ are NFA that both accept input string } w\}$.

Theorem

A is a decidable language.

Hint: you may use a TM N as a subroutine to check membership of NFA.

Decidable problems concerning context-free languages

Membership Problem of CFL

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

Theorem

A_{CFG} is a decidable language.

Proof (1)

For CFG G and string w, we want to determine whether G generates w.

- One idea is to use G to go through all derivations to determine whether any is a derivation of w.
- Then if G does not generate w, this algorithm would never halt.
  Any problem?
- It gives a Turing machine that is a recognizer, but not a decider.

Recall:

Definition

A context-free grammar is in Chomsky normal form if every rule is of the form

$$A \to BC \quad \text{and} \quad A \to a$$

where a is a terminal, and A, B, and C are non-terminal – except that B and C may be not the start non-terminal.
In addition, we permit the rule $S \to \varepsilon$, where S is the start non-terminal.

Theorem

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Theorem

Let G be CFG in Chomsky normal form, and G generates w with $w \neq \varepsilon$. Then any derivation of w has $2|w| - 1$ steps.

Proof

Theorem

Let G be CFG in Chomsky normal form ($A \to BC$ and $A \to a$), and G generates w with $w \neq \varepsilon$. Then any derivation of w has $2|w| - 1$ steps.

Applying induction on the length $|w| \geq 1$ of w.

- Base case: $|w| = 1$, then $S \to w$, where S is the start symbol.
- Inductive step: $|w| = k + 1$ for some $k > 0$. Let $w = w_0 w_1 \cdots w_k$. There necessary exists a production rule $S \to BC$ such that
  - $B \to^* w_0 \cdots w_i$ and
  - $C \to^* w_{i+1} \cdots w_k$

  for some $0 \leq i \leq k - 1$.

  By applying induction hypothesis:
  - $B \to^* w_0 \cdots w_i$ has $2(i + 1) - 1$ steps
  - $C \to^* w_{i+1} \cdots w_k$ has $2(k - i) - 1$ steps

  Thus, $S \to^* w$ has $2(i + 1) - 1 + 2(k - i) - 1 + 1 = 2(k + 1) - 1$ steps.

Proof (2)

Theorem

A_{CFG} is a decidable language.

S on $\langle G, w \rangle$:

1. Convert G to an equivalent grammar in Chomsky normal form.
2. List all derivations with $2|w| - 1$ steps; except if $|w| = 0$, then instead check whether there is a rule $S \to \varepsilon$.
3. If any of these derivations generates w, then accept; otherwise reject.

Testing the emptiness

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

Theorem

E_{CFG} is a decidable language.

Proof (1)

To determine whether $L(G) = \emptyset$, the algorithm might try going through all possible w's, one by one. But there are infinitely many w's to try, so this method could end up running forever.

Instead, the algorithm solves a more general problem: determine for each non-terminal whether that non-terminal is capable of generating a string of terminals.

- First, the algorithm marks all the terminal symbols in the grammar.
- It scans all the rules of the grammar. If it finds a rule that permits some non-terminal to be replaced by some string of symbols, all of which are already marked, then it marks this non-terminal.
- Repeat the second step until no more non-terminal can be marked or the stack symbol is marked

Proof (2)

R on $\langle G \rangle$:

1. Mark all terminal symbols in G.
2. Repeat until no new non-terminals get marked or the start symbol is marked:
   - Mark any non-terminal A where G contains a rule $A \to U_1 \cdots U_k$ and all U_i's have already been marked.
3. If the start symbol is not marked, then accept; otherwise, reject.

Testing equality

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

Recall

From A and B we construct a DFA C such that

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right),$$

Then

$$L(A) = L(B) \iff L(C) = \emptyset.$$

But

Theorem

EQ_{CFG} is not decidable.

Theorem

Every context-free language is decidable.

Recall using Chomsky normal form, we have shown:

Theorem

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

M_G on $\langle G\rangle$:

1. Run the TM $M_{G,w}$ on input $\langle G, w\rangle$.
2. If $M_{G,w}$ accepts, then accept; otherwise reject

Discussion

The Universality Problem

$$A = \{\langle G \rangle \mid G \text{ is a CFG} \wedge L(G) = \Sigma^*\}.$$

Theorem

A is not decidable.

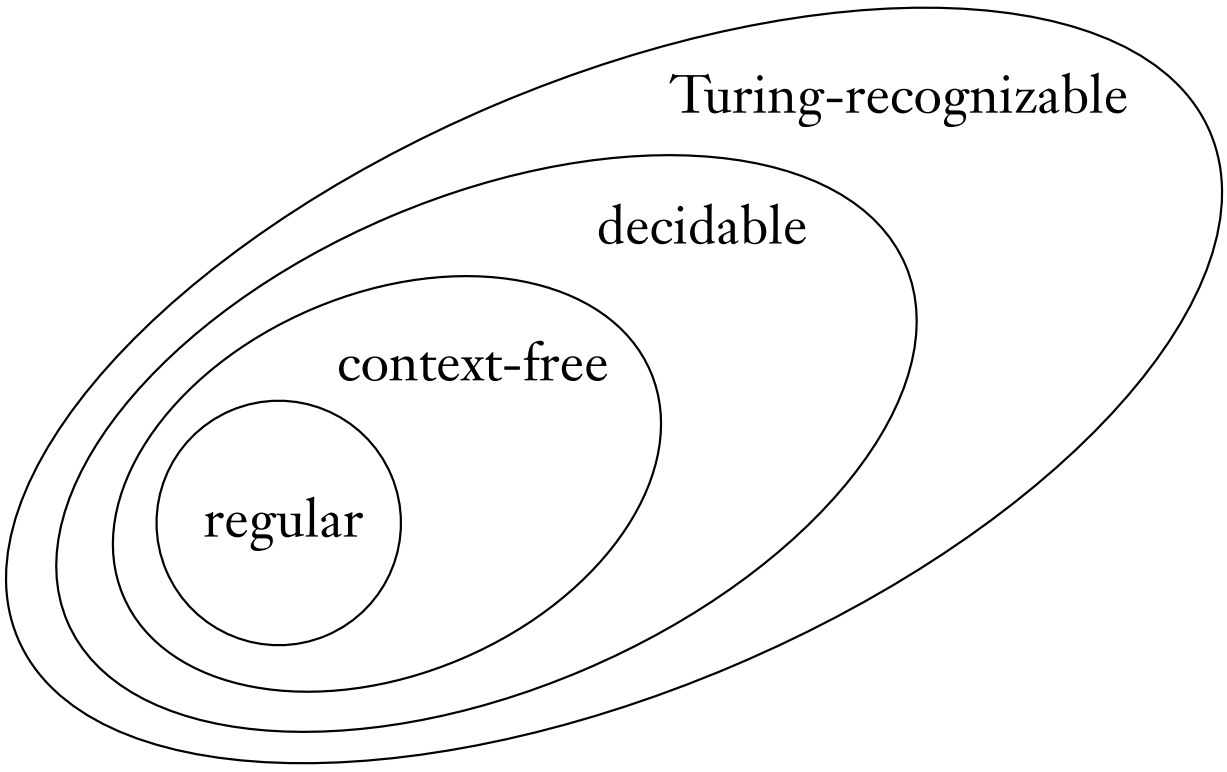
The Inclusion Problem

$$B = \{\langle G_1, G_2 \rangle \mid G_1 \text{ and } G_2 \text{ are CFGs} \wedge L(G_1) \subseteq L(G_2)\}.$$

Theorem

B is not decidable.

Relationship among classes of languages



Outline

Undecidability

- One of the most philosophically important theorems of the theory of computation:
  There is a specific problem that is algorithmically unsolvable.
- Computers appear to be so powerful that you may believe that all problems will eventually yield to them.
- The theorem presented here demonstrates that computers are limited in a fundamental way.

One type of unsolvable problem

- Given a computer program and a precise specification of what that program is supposed to do (e.g., sort a list of numbers).
- Verify that the program performs as specified (i.e., that it is correct).
- Because both the program and the specification are mathematically precise objects, you hope to automate the process of verification by feeding these objects into a suitably programmed computer.
- However, the general problem of software verification is not solvable by computer

Objectives

- Introduce and prove several computationally unsolvable problems
- Help you develop a feeling for the types of problems that are unsolvable and to learn techniques for proving unsolvability.

Testing membership

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

Theorem

A_{TM} is not decidable.

Theorem

A_{TM} is Turing-recognizable (i.e., more powerful than deciders).

Proof.

U on $\langle M, w \rangle$:

1. Simulate M on w.
2. If M enters its accept state, then accept; if it enters its reject state, reject.

□

U is a universal Turing machine first proposed by Alan Turing in 1936. This machine is called universal because it is capable of simulating any other Turing machine from the description of that machine.

The diagonalization method

The diagonalization method

- Diagonalization was discovered by mathematician Georg Cantor in 1873.
- Cantor was concerned with the problem of measuring the sizes of infinite sets.
- If we have two infinite sets, how can we tell whether one is larger than the other or whether they are of the same size?
- Cantor observed that two sets have the same size if the elements of one set can be paired with the elements of the other set.
- $|\{i \in \mathbb{N} \mid i \text{ is even } \}| = |\mathbb{N}|$

Functions

Definition

Let $f : A \to B$ be a function.

1. f is one-to-one, if $a \neq a'$, then $f(a) \neq f(a')$.
2. f is onto, if for every $b \in B$ there is an $a \in A$ with $f(a) = b$.

A and B are the same size if there is a one-to-one, onto function $d : A \to B$.

A function that is both one-to-one and onto is a correspondence.

injective	one-to-one
surjective	onto
bijective	one-to-one and onto

Cantor's Theorem

Definition

A is countable if it is either finite or has the same size as $\mathbb{N}$.

Theorem

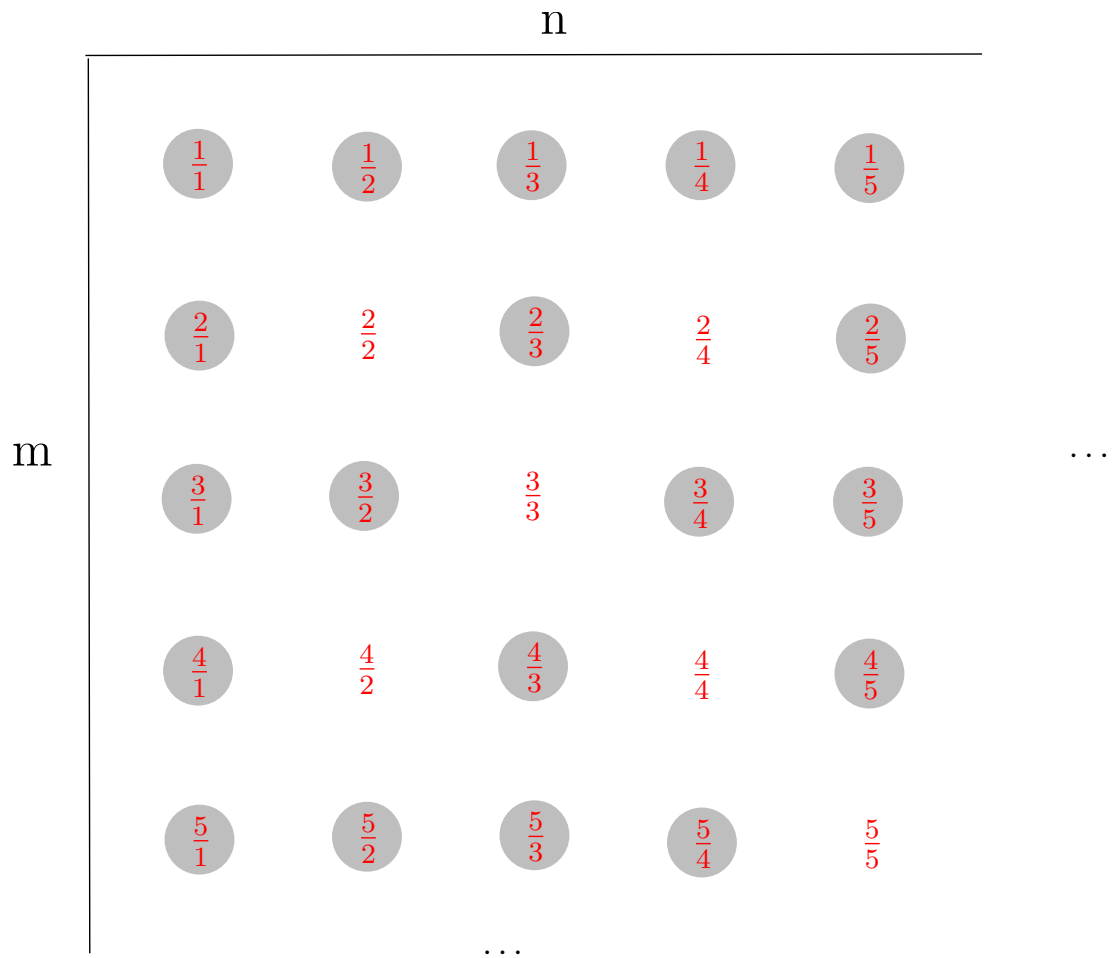
Positive rational number $\mathbb{G} = \{\frac{m}{n} \mid m, n \in \mathbb{N}\}$ *is countable.*

Proof

Theorem

Positive rational number $\mathbb{G} = \{\frac{m}{n} \mid m, n \in \mathbb{N}\}$ *is countable.*

- Give a correspondence with $\mathbb{N}$ to show that $\mathbb{Q}$ is countable. How?
- Represent $\mathbb{G}$ as an infinite matrix

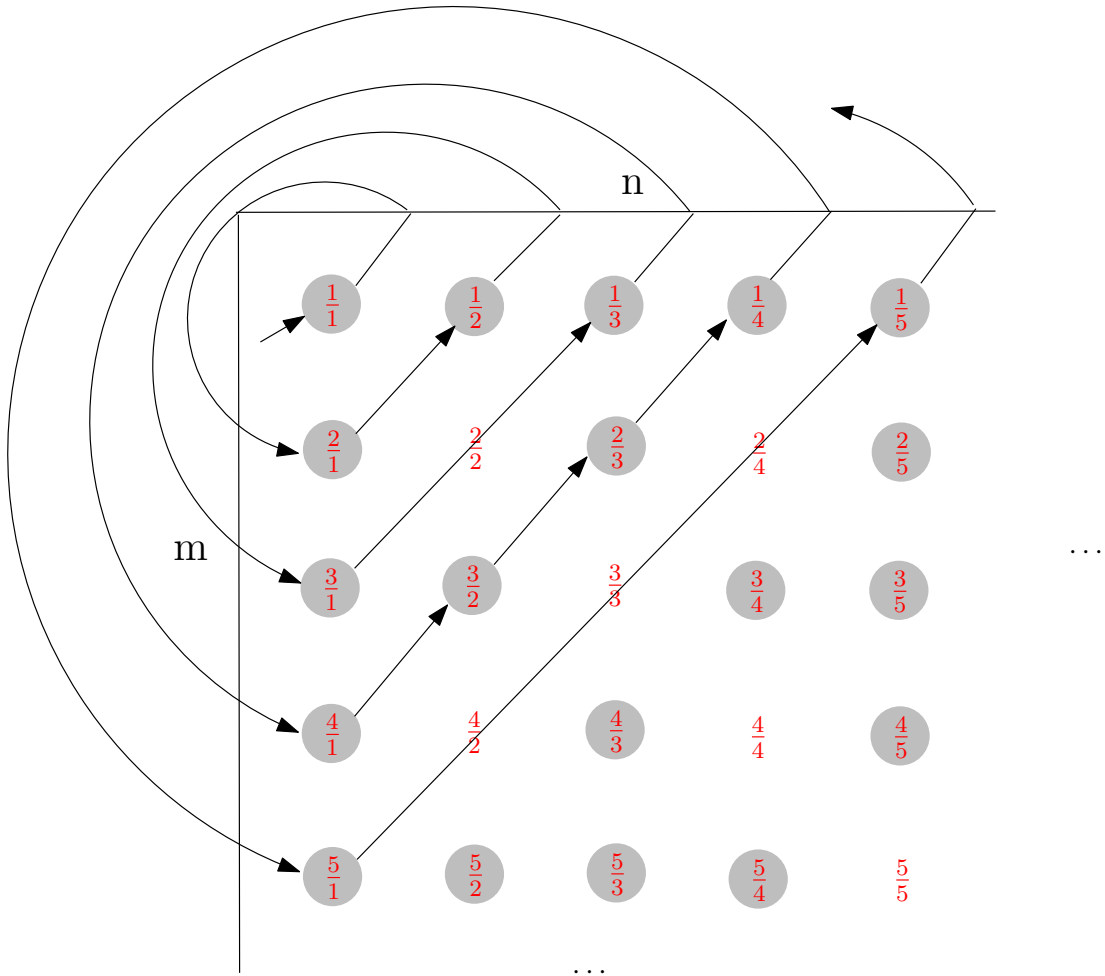


Proof

Theorem

Positive rational number $\mathbb{G} = \{\frac{m}{n} \mid m, n \in \mathbb{N}\}$ *is countable.*

- Give a correspondence with $\mathbb{N}$ to show that $\mathbb{Q}$ is countable. How?
- Represent $\mathbb{G}$ as an infinite matrix



Uncountable set

Theorem

Real number $\mathbb{R}$ is not countable.

- Proof by contradiction
- Assume that $\mathbb{R}$ is countable. Let $f : \mathbb{N} \to \mathbb{R}$ be a correspondence

n	f(n)
1	$x_{11}.x_{12}\ x_{13}\ x_{14}\cdots$
2	$x_{21}.x_{22}\ x_{23}\ x_{24}\cdots$
3	$x_{31}.x_{32}\ x_{33}\ x_{34}\cdots$
4	$x_{41}.x_{42}\ x_{43}\ x_{44}\cdots$

- Construct $x = 0.\overline{x_{12}}\ \overline{x_{23}}\ \overline{x_{34}}\cdots$, where $x_{ij} \neq \overline{x_{ij}} \notin \{0, 9\}$.
- Then, $x \neq f(n)$ for all $n \in \mathbb{N}$

Corollary

Some languages are not Turing-recognizable.

Proof

We fix an alphabet Σ.

1. Σ^* is countable, a list of Σ^* by writing down all strings of length 0, length 1, length 2, and so on
2. The set of all TMs is countable, as every M can be identified with a finite string $\langle M \rangle$.
3. The set of all infinite binary sequences $\mathbb{B}$ is uncountable (can be proved similar to $\mathbb{R}$)
4. We show that the set of all languages over Σ is uncountable, i.e., $\mathcal{L} = \{L \subseteq \Sigma^*\}$ is uncountable by giving a correspondence $f : \mathcal{L} \to \mathbb{B}$ with the uncountable language $\mathbb{B}$
5. Suppose the countable language $\Sigma^* = \{s_1, s_2, s_3, \cdots\}$, for every $L \in \mathcal{L}$, let $f(L)$ be an infinite binary sequence $w = w_1 w_1 \cdots$ such that $w_i = 1$ iff $s_i \in L$ (as well as $w_i = 0$ iff $s_i \notin L$). Obviously, f is one-to-one and onto function. (Note that L is countable.) Hence, $\mathcal{L}$ is uncountable.

An undecidable language

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

Theorem

A_{TM} is undecidable.

Proof (1)

Proof by contradiction. Assume A_{TM} is decidable.

Let H be a decider for A_{TM}. That is

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

Proof (2)

Construct a new TM D using the decider H of A_{TM} as a subroutine to determine what M does when the input to M is its string encoding $\langle M \rangle$

D on $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$.
2. Output the opposite of what H outputs. That is, if H accepts, then reject; and if H rejects, then accept.

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle. \end{cases}$$

Run D on its own string encoding $\langle D \rangle$,

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D \rangle. \end{cases}$$

Proof (3)

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\ldots$
M_1	accept		accept		
M_2	accept	accept	accept	accept	
M_3					$\ldots$
M_4	accept	accept			
$\vdots$			$\vdots$		

Run M_i on $\langle M_j \rangle$, entry i, j is accept if M_i accepts $\langle M_j \rangle$.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\ldots$
M_1	accept	reject	accept	reject	
M_2	accept	accept	accept	accept	
M_3	reject	reject	reject	reject	$\ldots$
M_4	accept	accept	reject	reject	
$\vdots$			$\vdots$		

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

Proof (4)

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\ldots$	$\langle D \rangle$	$\ldots$
M_1	accept	reject	accept	reject		accept	
M_2	accept	accept	accept	accept	$\ldots$	accept	
M_3	reject	reject	reject	reject		reject	
M_4	accept	accept	reject	reject		accept	
$\vdots$			$\vdots$				
D	reject	reject	accept	accept		?	
$\vdots$			$\vdots$				

D is a TM, it should be in the figure, then a contradiction occurs at "?"

D rejects $\langle D \rangle$ exactly when D accepts $\langle D \rangle$.

co-Turing-recognizable

Definition

A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language.

Theorem

A language is decidable if and only if it is Turing recognizable and co-Turing-cognizable.

Proof

($\Rightarrow$)

- A is decidable$\rightarrow$ A is Turing-recognizable
- A is decidable$\rightarrow$ $\bar{A}$ is decidable$\rightarrow$ $\bar{A}$ is Turing-recognizable
- $\bar{A}$ is Turing-recognizable $\rightarrow$ A is co-Turing-recognizable

($\Leftarrow$) Assume A is Turing recognizable and co-Turing-cognizable, then both A and $\bar{A}$ are Turing recognizable by M_1 and M_2 respectively.

The TM M on input w:

1. Run M_1 and M_2 on input w in parallel.
2. If M_1 accepts, then accept; and if M_2 accepts, then reject.

Clearly, M recognizes A. Since either $w \in A$ or $w \in \bar{A}$, then either M_1 or M_2 halts on w. Hence, M always halts, we proved that M is a decider.

Corollary

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

Proof.

A_{TM} is Turing-recognizable but undecidable.

$\overline{A_{\mathsf{TM}}}$ is Turing-recognizable $\rightarrow$ A_{TM} is co-Turing-recognizable $\rightarrow$ A_{TM} is decidable.

$\square$