

Decidability

We investigate the power of algorithms to solve problems. We demonstrate that certain problems can be solved algorithmically and others cannot. Our objective is to explore the limits of algorithmic solvability.

Coding decision problems as languages

Decision problems are problems that expect a Yes/No answer. E.g.

1. PRIME: Given a number, is it prime?
2. CYCLIC: Given a graph, is it cyclic?
3. DFA ACCEPTANCE: Given a DFA B and an input w , does B accept w ?

We represent decision problems as languages.

E.g. the language representing CYCLIC is

$$\{ \langle G \rangle : G \text{ is a cyclic graph} \}$$

where $\langle G \rangle$ denotes an encoding of G as a string over

$\Sigma = \{ 0, 1, (,), \# \}$ (say). The graph $(\{ 1, 2, 3 \}, \{ (1, 2), (2, 3), (3, 1) \})$ can be encoded as

$$(1\#10\#11)\#\left((1\#10)\#\left(10\#11\right)\#\left(11\#1\right)\right).$$

We say that a decision problem is *decidable* if the corresponding language is (Turing-machine) decidable.

DFA Acceptance Problem

DFA Acceptance Problem: Given a DFA B and an input w , does B accept w ?

The corresponding language is

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

Lemma. A_{DFA} is a decidable language.

Proof. We construct a TM M that decides A_{DFA} : On input $\langle B, w \rangle$ where B is a DFA and w an input

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*.

Convince yourself that it follows that A_{NFA} and A_{regex} (acceptance problems for NFA and regular expressions respectively) are also decidable.

Emptiness Problem for DFA: Given a DFA A , is $L(A)$ empty?

The corresponding language is

$$E_{\text{DFA}} = \{ \langle A \rangle : A \text{ is a DFA such that } L(A) = \emptyset \}$$

Lemma. E_{DFA} is a decidable language.

Proof. We design a TM T that uses a marking algorithm. 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 marked state.
3. If no accept state is marked, *accept*; otherwise *reject*.

Equivalence Problem for DFA: Given two DFAs, are they equivalent?

The corresponding language:

$$EQ_{\text{DFA}} = \{ \langle A, B \rangle : A \text{ and } B \text{ are DFA s.t. } L(A) = L(B) \}.$$

Lemma. EQ_{DFA} is a decidable language.

Proof. Key idea: For sets P and Q , $P \subseteq Q$ iff $P \cap \overline{Q} = \emptyset$. We use T , the algorithm for E_{DFA} .

On input $\langle A, B \rangle$, where A and B are DFAs:

1. Construct $C = (A \cap \overline{B}) \cup (B \cap \overline{A})$. (DFAs are closed under union, intersection and complementation.)
2. Run T on input $\langle C \rangle$.
3. If T accepts, accept; if T rejects, reject.

Acceptance problem for CFG

Consider the language:

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

Lemma. A_{CFG} is a decidable language.

We use a fact: If G is in Chomsky normal form, any derivation of w has $2n - 1$ steps, where n is the length of w .

Proof. The TM S for A_{CFG} is: On input $\langle G, w \rangle$, where G is a CFG and w is a string:

1. Convert G to an equivalent Chomsky normal form.
2. List all derivations with $2n - 1$ steps, where n is the length of w .
3. If any of these derivations generate w , accept; otherwise, reject.

This is of course extremely inefficient, and will not be used in practice.

Emptiness Problem for CFG: Given a CFG G , is $L(G)$ empty?

Lemma. E_{CFG} is a decidable language.

Proof. The TM R for E_{CFG} is: On input $\langle G \rangle$, where $G = (V, \Sigma, R, S)$ is a CFG:

1. Mark all terminal symbols in G .
2. Repeat until no new variables get marked:
Mark any variable A where G contains a rule $A \rightarrow U_1 \cdots U_k$ and each symbol $U_i \in \Sigma \cup V$ has already been marked.
3. If the start symbol is not marked, accept; otherwise, reject.

Equiv. Problem for CFG: Given two CFGs, are they equivalent?

Obvious attempt: Use the strategy for deciding EQ_{DFA} .

Unfortunately CFGs are neither closed under intersection nor complementation!

In fact EQ_{DFA} is *undecidable*.

Universal Turing Machines

Acceptance Problem for TM

$$A_{\text{TM}} = \{ \langle M, w \rangle : M \text{ is a TM that accepts input } w \}$$

Theorem. A_{TM} is r.e.

Proof. We define a TM U that accepts A_{TM} : On input $\langle M, w \rangle$ where M is a TM and w is a string

1. *Simulate M on input w .*
2. If M ever enters its accept state, accept; if M ever enters its reject state, reject.

Note that U loops on $\langle M, w \rangle$ if M loops on w . This is why U is not a decider.

If the algorithm had some way to determine that M was not halting on w , it could reject.

Universal Turing Machines: Encoding Turing machines

A *universal* TM is a TM U that can simulate the actions of any Turing machine.

I.e. for any TM M and any input x of M , U accepts (respectively rejects or loops on) $\langle M, x \rangle$, if and only if, M accepts (respectively rejects or loops on) x .

This is equivalent to an interpreter of C written in in C.

An encoding scheme over $\{0, 1\}^*$: $\langle M, x \rangle$.

E.g. if the string begins with the prefix

$$0^n 1 0^m 1 0^k 1 0^s 1 0^t 1 0^r 1 0^u 1 0^v$$

this might indicate that the machine has n states $(0, 1, \dots, n - 1)$; it has m tapes symbols $(0, 1, \dots, m - 1)$, of which the first k are input symbols; the start, accept and reject states are s, t and r respectively, and the endmarker and blank symbols are u and v respectively. The remainder of the string can consist of a sequence of substrings specifying the transitions in δ .

E.g. $0^p 1 0^a 1 0^q 1 0^b 1 0$ might indicate that δ contains the transition

$$((p, a), (q, b, L)).$$

The exact details are unimportant.

Simulation. The tape of U is partitioned into 3 tracks:

- The top track holds the transition function δ_M of the input TM M .
- The middle track will be used to hold the simulated contents of M 's tape.
- The bottom track will hold the current state of M , and the current position of M 's tape head.

U simulates M on input x one step at a time, shuttling back and forth between the three tracks. In each step, it updates M 's state and simulated tape contents and head position as dictated by δ_M . If ever M halts and accepts or halts and rejects, then U does the same.

Note: M 's input alphabet and U 's may be different.

There are languages that are not r.e.

There are only countably many TMs.

Observe that Σ^* is countable, for any alphabet Σ . Since $\langle M \rangle \in \Sigma^*$ for each TM M , we can enumerate the set of TMs by simply omitting those strings that are not encodings of TMs.

The set of languages over Σ , $\mathcal{P}(\Sigma^*)$, is uncountable.

Suppose there were an enumeration of $\mathcal{P}(\Sigma^*)$, namely, L_1, L_2, L_3, \dots . Let x_1, x_2, x_3, \dots be an enumeration of Σ^* . Define a new language L by: for $i \geq 1$

$$x_i \in L \iff x_i \notin L_i$$

Now $L \in \mathcal{P}(\Sigma^*)$, but L is not equal to any of the L_i s.

Thus $\mathcal{P}(\Sigma^*)$ is not in 1-1 correspondence with the set of Turing machines (there are more languages than TMs). It follows that there is some language that is not accepted by any TM.

A_{TM} : Acceptance Problem for TMs

Theorem. A_{TM} is undecidable.

Proof. Suppose, for a contradiction, TM H is a decider for A_{TM} . I.e.

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

We construct a new TM D with H as a subroutine. D : On input $\langle M \rangle$, where M is a TM

1. Run H on input $\langle M, \langle M \rangle \rangle$
2. Output the opposite of H . I.e. if H accepts, *reject*, and if H rejects, *accept*.

Thus

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

Now we run D on input $\langle D \rangle$. We have

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

I.e. D accepts $\langle D \rangle$ iff D rejects $\langle D \rangle$, which is a contradiction. Thus neither D nor H can exist. □

To summarize, A_{TM} is (r.e. but) undecidable.

A language that is not r.e.

We say that L is *co-r.e.* if \bar{L} is r.e.

Theorem. A language is decidable iff it is both r.e. and co-r.e.

Proof. Suppose L and \bar{L} are acceptable by M and M' respectively. Define a TM N : On input x

1. Write x on the input tapes of M and M' .
2. Run M and M' in parallel (e.g. by dovetailing the two computation).
3. If M accepts, *accept*; if M' accepts, *reject*.

N is a decider for L because for any input x , either $x \in L$ or $x \in \bar{L}$: if the former then N must accept because M must halt and accept at some point, if the latter, then N must reject because M' must halt and reject at some point.

Corollary. $\overline{A_{\text{TM}}}$ is not r.e. □

The Halting Problem: “Given M and x , does M halt on x ?”

$$HALT_{TM} = \{ \langle M, w \rangle : M \text{ is a TM and } M \text{ halts on input } w \}$$

Theorem. The Halting Problem, $HALT_{TM}$, is undecidable.

Proof. Suppose, for a contradiction, H is a decider for $HALT_{TM}$. We use H to construct a TM S to decide A_{TM} as follows: On input $\langle M, w \rangle$ where M is a TM w is an input

1. Run TM H on input $\langle M, w \rangle$
2. If H rejects, *reject*.
3. If H accepts, run M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*.

Since A_{TM} is undecidable, H cannot be a decider for $HALT_{TM}$. □

The Emptiness Problem for TM is undecidable

$$E_{\text{TM}} = \{ \langle M \rangle : M \text{ is a TM and } L(M) = \emptyset \}$$

Theorem. The Emptiness Problem for TM is undecidable.

Proof. Suppose, for a contradiction, E_{TM} is decidable by a TM E . We shall use E to construct a TM S that decides A_{TM} .

Definition of S : On input $\langle M, x \rangle$ where M is a TM and x an input

1. Construct a TM M_x defined as: On input y
 - (a) If $y \neq x$ then *reject*.
 - (b) Run M (input is x) and *accept* if M does.
2. Run E on input $\langle M_x \rangle$. If E accepts, *reject*, if E rejects, *accept*.

Now we have

$$\begin{aligned} M \text{ accepts } x &\Rightarrow L(M_x) = \{ x \} && \text{i.e. } S \text{ accepts } \langle M, x \rangle \\ M \text{ does not accept } x &\Rightarrow L(M_x) = \emptyset && \text{i.e. } S \text{ rejects } \langle M, x \rangle \end{aligned}$$

Thus S decides A_{TM} , which is a contradiction. □

The Equivalence Problem for TM is undecidable

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

Theorem. The Equivalence Problem for TM is undecidable.

Proof. Suppose, for a contradiction, EQ_{TM} is decidable by Q . We define a TM that decides E_{TM} by: On input M

1. Run Q on input $\langle M, M_1 \rangle$ where M_1 is a TM that accepts nothing i.e. $L(M_1) = \emptyset$.
2. If Q accepts, *accept*; if Q rejects, *reject*.

This gives a contradiction. □

PCP: Post Correspondence Problem

A puzzle: Given a collection of dominos such as e.g.

$$P = \left\{ \left[\frac{b}{ca} \right], \left[\frac{a}{ab} \right], \left[\frac{ca}{a} \right], \left[\frac{abc}{c} \right] \right\}$$

find a list of dominos from P (repetitions permitted) so that the string we get by reading off the symbols on the top is the same as the string of symbols on the bottom. This list is called a *match*.

E.g. the following is a match

$$\left\{ \left[\frac{a}{ab} \right], \left[\frac{b}{ca} \right], \left[\frac{ca}{a} \right], \left[\frac{a}{ab} \right], \left[\frac{abc}{c} \right], \right\}$$

(Reading off the top string we get *abcaabc*.)

A formal statement of the PCP

Instance: A collection P of *dominos*:

$$P = \left\{ \left[\begin{array}{c} t_1 \\ b_1 \end{array} \right], \left[\begin{array}{c} t_2 \\ b_2 \end{array} \right], \dots, \left[\begin{array}{c} t_k \\ b_k \end{array} \right] \right\}$$

where $k \geq 1$, and each t_i and b_j are strings over Σ .

Question: Is there a *match* for P ? I.e. a non-empty sequence i_1, \dots, i_l where each $1 \leq i_j \leq k$ and

$$t_{i_1} t_{i_2} \dots t_{i_l} = b_{i_1} b_{i_2} \dots b_{i_l}$$

Theorem. PCP is undecidable.

Examples

1. Let $\Sigma = \{0, 1\}$, and

$$P = \left\{ \left[\begin{array}{c} 11 \\ \hline 111 \end{array} \right], \left[\begin{array}{c} 111 \\ \hline 11 \end{array} \right], \left[\begin{array}{c} 100 \\ \hline 001 \end{array} \right] \right\}.$$

Here is a match for P :

$$P = \left\{ \left[\begin{array}{c} 11 \\ \hline 111 \end{array} \right], \left[\begin{array}{c} 100 \\ \hline 001 \end{array} \right], \left[\begin{array}{c} 111 \\ \hline 11 \end{array} \right] \right\}.$$

2. For some collection of dominos, find a match may not be possible. E.g.

$$\left\{ \left[\begin{array}{c} abc \\ \hline ab \end{array} \right], \left[\begin{array}{c} ca \\ \hline a \end{array} \right], \left[\begin{array}{c} acc \\ \hline bc \end{array} \right] \right\}$$

(because every top string is longer than the bottom string)

Undecidability of PCP: outline proof

We modify *PCP* to

$$MPCP = \{P \text{ is an instance of PCP with a match} \\ \text{that starts with the first domino}\}$$

1. Suppose, for a contradiction, there is a TM R that decides the PCP. We use R to construct a TM S that decide A_{TM} . Let

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_{acc}, q_{rej})$$

Take an input w for M . We construct an instance P' of the *MPCP* (over the alphabet $\Gamma \cup \{ \# \}$) that simulates M on w i.e.

$$P' \in MPCP \iff \langle M, w \rangle \in A_{TM}.$$

Idea. Writing a configuration (u_i, q_i, v_i) as $\#u_i q_i v_i$, a match of P' gives a string; the first part

$$\#q_0 \vdash w \#u_1 q_1 v_1 \#u_2 q_2 v_2 \# \cdots \#u_i q_i v_i \# \cdots \#u_n q_{acc} v_n$$

corresponds exactly to an accepting configuration; the rest of the string

$$\# \cdots \# \cdots \cdots \#q_{acc} \#\#$$

consists of progressively shorter segments of the form $\# \cdots$ that finally ends in $\#\#$.

2. We convert P' to P , an instance of PCP which still simulates M on w .

Definition of P'

Part 1. Put $\left[\frac{\#}{\#q_0 \vdash w_1 \cdots w_n \#} \right]$ into P' as the first domino $\left[\frac{t_1}{b_1} \right]$

Part 2. For each $a, b \in \Gamma, q, r \in Q$, if $\delta(q, a) = (r, b, R)$, put $\left[\frac{qa}{br} \right]$ into P'

Part 3. For each $a, b \in \Gamma, q, r \in Q$, if $\delta(q, a) = (r, b, L)$, put $\left[\frac{cqa}{rcb} \right]$ into P'

Part 4. For every $a \in \Gamma$, put $\left[\frac{a}{a} \right]$ into P' .

Part 5. Put $\left[\frac{\#}{\#} \right]$ and $\left[\frac{\#}{\sqcup \#} \right]$ into P' .

Part 6. For every $a \in \Gamma$, put $\left[\frac{aq_{acc}}{q_{acc}} \right]$ and $\left[\frac{q_{acc}a}{q_{acc}} \right]$ into P' .

Part 7. Add $\left[\frac{q_{acc} \# \#}{\#} \right]$ to P' .

E.g. $\Sigma = \{0, 1, 2\}$ Input: 01

$$\delta : \begin{cases} (q_0, \vdash) \mapsto (q_0, \vdash, R) \\ (q_0, 0) \mapsto (q_3, 2, R) \\ (q_3, 1) \mapsto (q_4, 0, R) \\ (q_4, \sqcup) \mapsto (q_{acc}, 1, L) \end{cases}$$

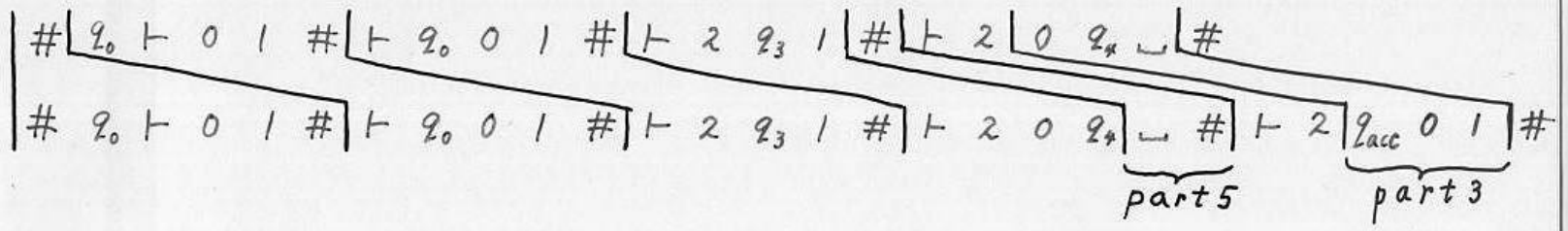
An accepting computation

$$\begin{aligned} (\varepsilon, q_0, \vdash 01) &\rightarrow (\vdash, q_0, 01) \\ &\rightarrow (\vdash 2, q_3, 1) \\ &\rightarrow (\vdash 20, q_4, \sqcup) \\ &\rightarrow (\vdash 2, q_{acc}, 01) \end{aligned}$$

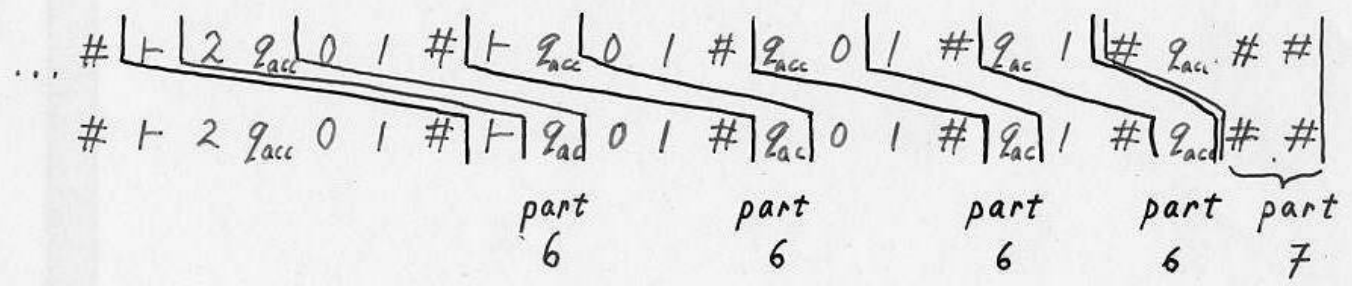
$\#$ |
 $\# q_0 \vdash 0 \mid \#$ |
 part 1

$\#$ | $q_0 \vdash$ |
 $\# q_0 \vdash 0 \mid \# \vdash q_0$ |
 part 3

$\# q_0 \vdash$ | $0 \mid \#$ |
 $\# q_0 \vdash 0 \mid \# \vdash q_0$ | $0 \mid \#$ |
 part 4 part 5



Tidying up



Converting a MPCP instance to a PCP instance

Finally we show how to convert P' to an instance of PCP which still simulates M on w . The idea is to build the requirement of starting with the first domino directly into the problem.

Notation. Take a string $u = u_1 \cdots u_n$. Define

$$\star u = \star u_1 \star u_2 \star \cdots \star u_n$$

$$u \star = u_1 \star u_2 \star \cdots \star u_n \star$$

$$\star u \star = \star u_1 \star u_2 \star \cdots \star u_n \star$$

Suppose P' is the MPCP instance (over alphabet Θ)

$$\left\{ \left[\frac{t_1}{b_1} \right], \left[\frac{t_2}{b_2} \right], \left[\frac{t_3}{b_3} \right], \dots, \left[\frac{t_k}{b_k} \right] \right\}$$

Now define P to be the PCP instance (over alphabet $\Theta \cup \{*, \diamond\}$)

$$\left\{ \left[\frac{*t_1}{*b_1*} \right], \left[\frac{*t_2}{b_2*} \right], \left[\frac{*t_3}{b_3*} \right], \dots, \left[\frac{*t_k}{b_k*} \right], \left[\frac{* \diamond}{\diamond} \right] \right\}$$

Note that the only domino that can start a match is the first one, because it is the only one where both the top and bottom start with the same symbol, namely, $*$.

The original symbols in Θ now occur in even positions of the match.

The domino $\left[\frac{* \diamond}{\diamond} \right]$ is to allow the top to add the extra $*$ at the end of the match.