

Intro to Analysis of Algorithms

Computational Foundations

Section 9.5

Chapter 9

Michael Soltys

CSU Channel Islands

[Ed: 4th, last updated: October 2, 2025]

Part I

Turing machines

Finite control and an infinite tape.

Initially the input is placed on the tape, the head of the tape is reading the first symbol of the input, and the state is q_0 .

The other squares contain blanks.

Formally, a *Turing machine* is a tuple $(Q, \Sigma, \Gamma, \delta)$

where Q is a finite set of *states* (always including the three special states q_{init} , q_{accept} and q_{reject})

Σ is a finite *input alphabet*

Γ is a finite *tape alphabet*, and it is always the case that $\Sigma \subseteq \Gamma$ (it is convenient to have symbols on the tape which are never part of the input),

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{\text{Left}, \text{Right}\}$$

is the *transition function*



Alan Turing

A **configuration** is a tuple (q, w, u) where $q \in Q$ is a state, and where $w, u \in \Gamma^*$, the cursor is on the last symbol of w , and u is the string to the right of w .

A configuration (q, w, u) **yields** (q', w', u') in one step, denoted as $(q, w, u) \xrightarrow{M} (q', w', u')$ if one step of M on (q, w, u) results in (q', w', u') .

Analogously, we define $\xrightarrow{M^k}$, yields in k steps, and $\xrightarrow{M^*}$, yields in any number of steps, including zero steps.

The initial configuration, C_{init} , is $(q_{\text{init}}, \triangleright, x)$ where q_{init} is the initial state, x is the input, and \triangleright is the left-most tape symbol, which is always there to indicate the left-end of the tape.

Given a string w as input, we “turn on” the TM in the initial configuration C_{init} , and the machine moves from configuration to configuration.

The computation ends when either the state q_{accept} is entered, in which case we say that the TM *accepts* w , or the state q_{reject} is entered, in which case we say that the TM *rejects* w . It is possible for the TM to never enter q_{accept} or q_{reject} , in which case the computation does not halt.

Given a TM M we define $L(M)$ to be the set of strings accepted by M , i.e., $L(M) = \{x \mid M \text{ accepts } x\}$, or, put another way, $L(M)$ is the set of precisely those strings x for which $(q_{\text{init}}, \triangleright, x)$ yields an accepting configuration.

Alan Turing showed the existence of a so called *Universal Turing machine* (UTM); a UTM is capable of simulating any TM from its description.

A UTM is what we mean by a *computer*, capable of running any algorithm. The proof is not difficult, but it requires care in defining a consistent way of presenting TMs and inputs.

Every Computer Scientist should at some point write a UTM in their favorite programming language ...

This exercise really means: designing your own programming language (how you present descriptions of TMs); designing your own compiler (how your machine interprets those “descriptions”); etc.

NTM

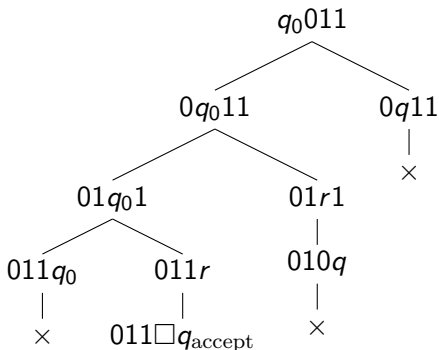
N s.t. $L(N) = \{w \in \{0,1\}^* \mid \text{last symbol of } w \text{ is } 1\}$.

$$\delta(q_0, 0) = \{(q_0, 0, \rightarrow), (q, 0, \rightarrow)\}$$

$$\delta(q_0, 1) = \{(q_0, 1, \rightarrow), (r, 1, \rightarrow)\}$$

$$\delta(r, \square) = \{(q_{\text{accept}}, \square, \rightarrow)\}$$

$$\delta(r, 0/1) = \{(q, 0, \rightarrow)\}$$



Different variants of TMs are equivalent (*robustness*): tape infinite in only one direction, or several tapes.

TM = NTM: D maintains a sequence of config's on tape 1:

...	config ₁	config ₂	config ₃ *	...
-----	---------------------	---------------------	-----------------------	-----

and uses a second tape for scratch work.

The marked config (*) is the current config. D copies it to the second tape, and examines it to see if it is accepting. If it is, it accepts.

If it is not, and N has k possible moves, D copies the k new config's resulting from these moves at the end of tape 1, and marks the next config as current.

If max nr of choices of N is m , and N makes n moves, D examines $1 + m + m^2 + m^3 + \dots + m^n \approx nm^n$ many configs.

Undecidability

We can encode every Turing machine with a string over $\{0, 1\}$. For example, if M is a TM:

$$(\{q_1, q_2\}, \{0, 1\}, \delta, \dots)$$

and $\delta(q_1, 1) = (q_2, 0, \rightarrow)$ is one of the transitions, then it could be encoded as:

$$\underbrace{0}_{q_1} 1 \underbrace{00}_1 1 \underbrace{00}_{q_2} 1 \underbrace{0}_0 1 \underbrace{0}_{\rightarrow} 11 \underbrace{\dots}_{\text{encoding of other transitions}}$$

Not every string is going to be a valid encoding of a TM (for example the string 1 does not encode anything in our convention).

Let all “bad strings” encode a default TM M_{default} which has one state, and halts immediately, so $L(M_{\text{default}}) = \emptyset$.

The intuitive notion of algorithm is captured by the formal definition of a TM.

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

called the *universal language*

Theorem 9.6: A_{TM} is undecidable.

Suppose that it is decidable, and that H decides it. Then, $L(H) = A_{TM}$, and H always halts (observe that $L(H) = L(U)$, but U , as we already mentioned, is not guaranteed to be a decider). Define a new machine D (here D stands for “diagonal,” since this argument follows Cantor’s “diagonal argument”):

$$D(\langle M \rangle) := \begin{cases} \text{accept} & \text{if } H(\langle M, \langle M \rangle \rangle) = \text{reject} \\ \text{reject} & \text{if } H(\langle M, \langle M \rangle \rangle) = \text{accept} \end{cases}$$

that is, D does the “opposite.” Then we can see that $D(\langle D \rangle)$ accepts iff it rejects. Contradiction; so A_{TM} cannot be decidable.

It turns out that all nontrivial properties of RE languages are undecidable, in the sense that the language consisting of codes of TMs having this property is not recursive.

E.g., the language consisting of codes of TMs whose languages are empty (i.e., L_e) is not recursive.

A *property* of RE languages is simply a subset of RE. A property is *trivial* if it is empty or if it is everything.

If \mathcal{P} is a property of RE languages, the language $L_{\mathcal{P}}$ is the set of codes for TMs M_i s.t. $L(M_i) \in \mathcal{P}$.

When we talk about the decidability of \mathcal{P} , we formally mean the decidability of $L_{\mathcal{P}}$.

Rice's Theorem: Every nontrivial property of RE languages is undecidable.

Proof: Suppose \mathcal{P} is nontrivial. Assume $\emptyset \notin \mathcal{P}$ (if it is, consider $\overline{\mathcal{P}}$ which is also nontrivial).

Since \mathcal{P} is nontrivial, some $L \in \mathcal{P}$, $L \neq \emptyset$.

Let M_L be the TM accepting L .

For a fixed pair (M, w) consider the TM M' : on input x , it first simulates $M(w)$, and if it accepts, it simulates $M_L(x)$, and if that accepts, M' accepts.

$\therefore L(M') = \emptyset \notin \mathcal{P}$ if M does not accept w , and $L(M') = L \in \mathcal{P}$ if M accepts w .

Thus, $L(M') \in \mathcal{P} \iff (M, w) \in A_{\text{TM}}$, $\therefore \mathcal{P}$ is undecidable.

Post's Correspondence Problem (PCP)

An instance of *PCP* consists of two finite lists of strings over some alphabet Σ . The two lists must be of equal length:

$$A = w_1, w_2, \dots, w_k$$

$$B = x_1, x_2, \dots, x_k$$

For each i , the pair (w_i, x_i) is said to be a *corresponding pair*. We say that this instance of PCP has a solution if there is a sequence of one or more indices:

$$i_1, i_2, \dots, i_m \quad m \geq 1$$

such that:

$$w_{i_1} w_{i_2} \dots w_{i_m} = x_{i_1} x_{i_2} \dots x_{i_m}$$

The PCP is: given (A, B) , tell whether there is a solution.



Emil Leon Post

Aside: To express **PCP** as a language, we let L_{PCP} be the language:

$$\{\langle A, B \rangle \mid (A, B) \text{ instance of PCP with solution}\}$$

Example: Consider (A, B) given by:

$$A = 1, 10111, 10$$

$$B = 111, 10, 0$$

Then 2, 1, 1, 3 is a solution as:

$$\underbrace{10111}_{w_2} \underbrace{1}_{w_1} \underbrace{1}_{w_1} \underbrace{10}_{w_3} = \underbrace{10}_{x_2} \underbrace{111}_{x_1} \underbrace{111}_{x_1} \underbrace{0}_{x_3}$$

Note that 2, 1, 1, 3, 2, 1, 1, 3 is another solution.

On the other hand, you can check that: $A = 10, 011, 101$ & $B = 101, 11, 011$ Does not have a solution.

The **MPCP** has an additional requirement that the first pair in the solution must be the first pair of (A, B) .

So i_1, i_2, \dots, i_m , $m \geq 0$, is a solution to the (A, B) instance of **MPCP** if:

$$w_1 w_{i_1} w_{i_2} \dots w_{i_m} = x_1 x_{i_1} x_{i_2} \dots x_{i_m}$$

We say that i_1, i_2, \dots, i_r is a *partial solution* of PCP if one of the following is the prefix of the other:

$$w_{i_1} w_{i_2} \dots w_{i_r} \quad x_{i_1} x_{i_2} \dots x_{i_r}$$

Same def holds for MPCP, but w_1, x_1 must be at the beginning.

We now show:

1. If **PCP** is decidable, then so is **MPCP**.
2. If **MPCP** is decidable, then so is A_{TM} .
3. Since A_{TM} is *not* decidable, neither is **(M)PCP**.

PCP decidable \implies **MPCP** decidable

We show that given an instance (A, B) of MPCP, we can construct an instance (A', B') of PCP such that:

$$(A, B) \text{ has solution} \iff (A', B') \text{ has solution}$$

Let (A, B) be an instance of MPCP over the alphabet Σ . Then (A', B') is an instance of PCP over the alphabet $\Sigma' = \Sigma \cup \{*, \$\}$.

If $A = w_1, w_2, w_3, \dots, w_k$, then
 $A' = *w_1*, w_1*, w_2*, w_3*, \dots, w_k*, \$$.

If $B = x_1, x_2, x_3, \dots, x_k$, then $B' = *x_1, *x_1, *x_2, *x_3, \dots, *x_k, *\$$.

where if $x = a_1 a_2 a_3 \dots a_n \in \Sigma^*$, then $\mathbf{x} = a_1 * a_2 * a_3 * \dots * a_n$.

For example: If (A, B) is an instance of MPCP given as:

$$A = 1, 10111, 10$$

$$B = 111, 10, 0$$

Then (A', B') is an instance of PCP given as follows:

$$A' = *1*, 1*, 1*0*1*1*1*, 1*0*, \$$$

$$B' = *1*1*1, *1*1*1, *1*0, *0, *\$$$

MPCP decidable $\implies A_{\text{TM}}$ decidable

Given a pair (M, w) we construct an instance (A, B) of MPCP such that:

TM M accepts $w \iff (A, B)$ has a solution.

Idea: The MPCP instance (A, B) simulates, in its partial solutions, the computation of M on w .

That is, partial solutions will be of the form:

$$\# \alpha_1 \# \alpha_2 \# \alpha_3 \# \dots$$

where α_1 is the initial config of M on w , and for all i , $\alpha_i \rightarrow \alpha_{i+1}$.

The string from the B list will always be one config ahead of the A list; the A list will be allowed to “catch-up” only when M accepts w .

To simplify things, we may assume that our TM M :

1. Never prints a blank.
2. Never moves left from its initial head position.

The configs of M will always be of the form $\alpha q \beta$, where α, β are non-blank tape symbols and q is a state.

Let M be a TM and $w \in \Sigma^*$. We construct an instance (A, B) of MPCP as follows:

1. A : #
 B : $\#q_0w\#$
2. A : $X_1, X_2, \dots, X_n, \#$
 B : $X_1, X_2, \dots, X_n, \#$
 where the X_i are all the tape symbols.
3. To simulate a move of M , for all non-accepting $q \in Q$:

list A	list B	
qX	Yp	if $\delta(q, X) = (p, Y, \rightarrow)$
ZqX	pZY	if $\delta(q, X) = (p, Y, \leftarrow)$
$q\#$	$Yp\#$	if $\delta(q, B) = (p, Y, \rightarrow)$
$Zq\#$	$pZY\#$	if $\delta(q, B) = (p, Y, \leftarrow)$

4. If the config at the end of B has an accepting state, then we need to allow A to catch up with B . So we need for all accepting states q , and all symbols X, Y :

list A	list B
XqY	q
Xq	q
qY	q

5. Finally, after using 4 and 3 above, we end up with $x\#$ and $x\#q\#$, where x is a long string. Thus we need $q\#\#$ in A and $\#$ in B to complete the catching up.

Ex. $\delta(q_1, 0) = (q_2, 1, \rightarrow)$, $\delta(q_1, 1) = (q_2, 0, \leftarrow)$, $\delta(q_1, B) = (q_2, 1, \leftarrow)$
 $\delta(q_2, 0) = (q_3, 0, \leftarrow)$, $\delta(q_2, 1) = (q_1, 0, \rightarrow)$, $\delta(q_2, B) = (q_2, 0, \rightarrow)$

Rule	list A	list B	Source
1	#	#q ₁ 01#	
2	0 1 #	0 1 #	
3	q ₁ 0 0q ₁ 1 1q ₁ 1 0q ₁ # 1q ₁ # 0q ₂ 0 1q ₂ 0 q ₂ 1 q ₂ #	1q ₂ q ₂ 00 q ₂ 10 q ₂ 01# q ₂ 11# q ₃ 00# q ₃ 10# 0q ₁ 0q ₂ #	$\delta(q_1, 0) = (q_2, 1, \rightarrow)$ $\delta(q_1, 1) = (q_2, 0, \leftarrow)$ $\delta(q_1, 1) = (q_2, 0, \leftarrow)$ $\delta(q_1, B) = (q_2, 1, \leftarrow)$ $\delta(q_1, B) = (q_2, 1, \leftarrow)$ $\delta(q_2, 0) = (q_3, 0, \leftarrow)$ $\delta(q_2, 0) = (q_3, 0, \leftarrow)$ $\delta(q_2, 1) = (q_1, 0, \rightarrow)$ $\delta(q_2, B) = (q_2, 0, \rightarrow)$

4	0q ₃ 0 0q ₃ 1 1q ₃ 0 1q ₃ 1 0q ₃ 1q ₃ q ₃ 0 q ₃ 1	q ₃ q ₃ q ₃ q ₃ q ₃ q ₃ q ₃ q ₃	
5	q ₃ ##	#	

The TM M accepts the input 01 by the sequence of moves:

$$q_1 01 \rightarrow 1q_2 1 \rightarrow 10q_1 \rightarrow 1q_2 01 \rightarrow q_3 101$$

We examine the sequence of partial solutions that mimics this computation of M and eventually leads to a solution.

We must start with the first pair (MPCP):

A : #

B : # q_1 01#

The only way to extend this partial solution is with the corresponding pair $(q_1 0, 1q_2)$, so we obtain:

A : # q_1 0

B : # q_1 01#1 q_2

Now using copying pairs we obtain:

A: $\#q_101\#1$

B: $\#q_101\#1q_21\#1$

Next corresponding pair is $(q_21, 0q_1)$:

A: $\#q_101\#1q_21$

B: $\#q_101\#1q_21\#10q_1$

Now careful! We only copy the next two symbols to obtain:

A: $\#q_101\#1q_21\#1$

B: $\#q_101\#1q_21\#10q_1\#1$

because we need the $0q_1$ as the head now moves left, and use the next appropriate corresponding pair which is $(0q_1\#, q_201\#)$ and obtain:

A: $\#q_101\#1q_21\#10q_1\#$

B: $\#q_101\#1q_21\#10q_1\#1q_201\#$

We can now use another corresponding pair $(1q_20, q_310)$ right away to obtain:

$A: \#q_101\#1q_21\#10q_1\#1q_20$

$B: \#q_101\#1q_21\#10q_1\#1q_201\#q_310$

and note that we have an accepting state! We use two copying pairs to get:

$A: \#q_101\#1q_21\#10q_1\#1q_201\#$

$B: \#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#$

and we can now start using the rules in 4. to make A catch up with B :

$A: \dots\#q_31$

$B: \dots\#q_3101\#q_3$

and we copy three symbols:

$A: \dots\#q_3101\#$

$B: \dots\#q_3101\#q_301\#$

And again catch up a little:

A: ...# q_3 101# q_3 0

B: ...# q_3 101# q_3 01# q_3

Copy two symbols:

A: ...# q_3 101# q_3 01#

B: ...# q_3 101# q_3 01# q_3 1#

and catch up:

A: ...# q_3 101# q_3 01# q_3 1

B: ...# q_3 101# q_3 01# q_3 1# q_3

and copy:

A: ...# q_3 101# q_3 01# q_3 1#

B: ...# q_3 101# q_3 01# q_3 1# q_3 #

And now end it all with the corresponding pair $(q_3\#\#, \#)$ given by rule 5. to get matching strings:

A: ... $\#q_3101\#q_301\#q_31\#q_3\#\#$

B: ... $\#q_3101\#q_301\#q_31\#q_3\#\#$

THEREFORE: we reduced A_{TM} to the MPCP. Now, we can solve A_{TM} by producing a carefully crafted instance of MPCP (A, B) , and asking if it has a solution. If yes, then we know that M accepts w .

Since we have already shown that A_{TM} is undecidable, MPCP must also be undecidable. Thus, PCP is undecidable.

NEXT: We can now use the fact that PCP is undecidable to show that a number of questions about CFLs are undecidable.

Let $A = w_1, w_2, \dots, w_k$, let G_A be the related CFG given by:

$$A \longrightarrow w_1 A a_1 | w_2 A a_2 | \dots | w_k A a_k | w_1 a_1 | w_2 a_2 | \dots | w_k a_k$$

Let $L_A = L(G_A)$, the language of the list A , and a_1, a_2, \dots, a_k are distinct index symbols not in alphabet of A .

The terminal strings of G_A are of the form:

$$w_{i_1} w_{i_2} \dots w_{i_m} a_{i_m} \dots a_{i_2} a_{i_1}$$

Let G_{AB} be a CFG consisting of G_A, G_B , with $S \longrightarrow A|B$.

$\therefore G_{AB}$ is ambiguous \iff the PCP (A, B) has a solution.

Theorem: It is undecidable whether a CFG is ambiguous.

$\overline{L_A}$ is also a CFL; we show this by giving a PDA P .

$$\Gamma_P = \Sigma_A \cup \{a_1, a_2, \dots, a_k\}.$$

As long as P sees a symbol in Σ_A it stores it on the stack.

As soon as P sees a_i , it pops the stack to see if top of string is w_i^R . (i) if not, then accept no matter what comes next. (ii) if yes, there are two subcases:

(iia) if stack is not yet empty, continue.

(iib) if stack is empty, and the input is finished, reject.

If after an a_i , P sees a symbol in Σ_A , it accepts.

Theorem: G_1, G_2 are CFGs, and R is a reg. exp., then the following are undecidable problems:

1. $L(G_1) \cap L(G_2) \stackrel{?}{=} \emptyset$
2. $L(G_1) \stackrel{?}{=} L(G_2)$
3. $L(G_1) \stackrel{?}{=} L(R)$
4. $L(G_1) \stackrel{?}{=} T^*$
5. $L(G_1) \stackrel{?}{\subseteq} L(G_2)$
6. $L(R) \stackrel{?}{\subseteq} L(G_2)$

Proofs: 1. Let $L(G_1) = L_A$ and $L(G_2) = L_B$, then $L(G_1) \cap L(G_2) \neq \emptyset$ iff PCP (A, B) has a solution.

2. Let G_1 be the CFG for $\overline{L_A} \cup \overline{L_B}$ (CFGs are closed under union). Let G_2 be the CFG for the reg. lang. $(\Sigma \cup \{a_1, a_2, \dots, a_k\})^*$.

Note $L(G_1) = \overline{L_A} \cup \overline{L_B} = \overline{L_A \cap L_B}$ = everything but solutions to PCP (A, B) .

$\therefore L(G_1) = L(G_2)$ iff (A, B) has no solution.

3. Shown in 2.

4. Again, shown in 2.

5. Note that $A = B$ iff $A \subseteq B$ and $B \subseteq A$, so it follows from 2.

6. By 3. and 5.