Efficient Turing Machines

CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN

Fall 2021

Chinese University of Hong Kong

Undecidability of PCP (optional)

Undecidability of PCP

$\text{PCP} = \{\langle C \rangle \mid C$ is a finite collection of tiles containing a top-bottom match$\}$

A top-bottom match is a finite sequence of tiles from C (possibly repeated) such that the top string equals the bottom string

The language PCP is undecidable

We will show that

If PCP can be decided, so can A_{TM}

We will only discuss the main idea, omitting details

Undecidability of PCP

$\langle M, w \rangle \quad \longmapsto \quad C$ (collection of tiles)

M accepts $w \quad \Longleftrightarrow \quad C$ contains a match

Idea: Matches represent accepting history

#$q_0$ab%ab#xq_1b%ab#...#xx%xq_ax#

#$q_0$ab%ab#xq_1b%ab#...#xx%xq_ax#

ε	#q_0a	b	a	%	a	b	#	xq_1%	...
#$q_0$ab%ab	#xq_1	b	a	%	a	b	#	x%q_2	

Undecidability of PCP

$$\langle M, w\rangle \longmapsto C \text{ (collection of tiles)}$$
$$M \text{ accepts } w \iff C \text{ contains a match}$$

We will assume that the following tile is forced to be the starting tile:

(S)
ε
$\#\,q_0$ab%ab

On input $\langle M, w\rangle$, we construct these tiles for PCP

(S)							
ε	$x_1\, q_i x_2$	$\#\, q_i x_1$	$\#$	x	$x q_a$	$q_a\, x$	$q_a\#\#$
$\#\, q_0\, w$	$x_3\, x_4 x_5$	$\square\# x_2 x_3$	$\square\#$	x	q_a	q_a	$\#$

(The brace over the tiles with x, xq_a and q_ax is labelled:) for all x in $\Gamma \cup \{\#\}$

(The brace under the second and third tiles is labelled:) for each valid window with state q_i in top middle

Undecidability of PCP

tile type	purpose
(S) $\left[\frac{\varepsilon}{\# q_0 w}\right]$	represents initial configuration
$\left[\frac{x_1 q_i x_2}{x_3 x_4 x_5}\right]$ $\left[\frac{x}{x}\right]$	represents valid transitions between configurations
$\left[\frac{\# q_i x_1}{\square \# x_2 x_3}\right]$ $\left[\frac{\#}{\square \#}\right]$	adds blank spaces before # if necessary
$\left[\frac{x q_a}{q_a}\right]$ $\left[\frac{q_a x}{q_a}\right]$ $\left[\frac{q_a \#\#}{\#}\right]$	matching completes if computation accepts

Undecidability of PCP

Once the accepting state symbol occurs, the last two tiles can “eat up” the rest of the symbols

#xx%x q_ax#xx%x q_a#...# q_a##

#xx%x q_ax#xx%x q_a#...# q_a##

x	xq_a	$q_a x$	q_a##
x	q_a	q_a	#

If M rejects on input w, then q_{rej} appears on the bottom at some point, but it cannot be matched on top

If M loops on w, then matching goes on forever

Getting rid of the starting tile

We assumed that one tile is marked as the starting tile
(the only tile that can start a match)

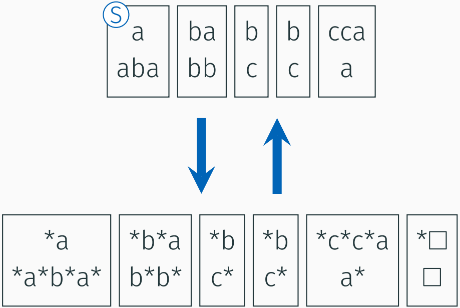
S			
a	ba	b	cca
aba	bb	c	a

We can simulate this assumption by changing tiles a bit

*a	*a	*b*a	*b	*c*c*a	*□
*a*b*a*	a*b*a*	b*b*	c*	a*	□

"starting tile" begins with * — "middle tiles" — "ending tiles"

Getting rid of the starting tile

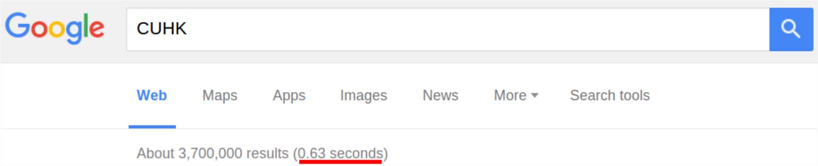


only possible
starting tile
(top and bottom strings
begin with the same symbol)

only possible
ending tile
(top and bottom strings
end with the same symbol)

Polynomial time

Running time



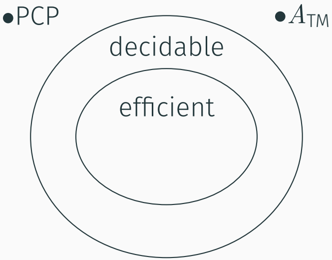
We don't want to just solve a problem, we want to solve it quickly

Efficiency

Undecidable problems:
We cannot find solutions in any finite amount of time

Decidable problems:
We can solve them, but it may take a very long time

Efficiency



The running time depends on the input

For longer inputs, we should allow more time

Efficiency is measured as a function of input size

Running time

The running time of a Turing machine M is the function $t_M(n)$:

$$t_M(n) = \text{maximum number of steps that } M \text{ takes on any input of length } n$$

Example: $L = \{w\#w \mid w \in \{\mathsf{a}, \mathsf{b}\}^*\}$

M: On input x, until you reach #	$O(n)$ times
Read and cross of first a or b before #	$O(n)$ steps
Read and cross off first a or b after #	
If mismatch, reject	
If all symbols except # are crossed off, accept	$O(n)$ steps
running time:	$O(n^2)$

Another example

$$L_{\texttt{01}} = \{\texttt{0}^n\texttt{1}^n \mid n \geqslant 0\}$$

M: On input x,	
Check that the input is of the form $\texttt{0}^*\texttt{1}^*$	$O(n)$ steps
Until everything is crossed off:	$O(n)$ times
Cross off the leftmost $\texttt{0}$	$O(n)$ steps (both lines)
Cross off the following $\texttt{1}$	
If everything is crossed off, accept	$O(n)$ steps
running time:	$O(n^2)$

A faster way

$$L_{\texttt{01}} = \{\texttt{0}^n\texttt{1}^n \mid n \geqslant 0\}$$

M: On input x,	
Check that the input is of the form $\texttt{0}^*\texttt{1}^*$	$O(n)$ steps
Until everything is crossed off:	$O(\log n)$ times
Find parity of number of 0s	$O(n)$ steps (for the five steps)
Find parity of number of 1s	
If the parities don't match, reject	
Cross off every other 0	
Cross off every other 1	
If everything is crossed off, accept	$O(n)$ steps
running time:	$O(n \log n)$

Running time vs model

What if we have a two-tape Turing machine?

$$L_{\texttt{01}} = \{\texttt{0}^n\texttt{1}^n \mid n \geqslant 0\}$$

M: On input x,	
Check that the input is of the form $\texttt{0}^*\texttt{1}^*$	$O(n)$ steps
Copy $\texttt{0}^*$ part of input to second tape	$O(n)$ steps
Until $\square$ is reached:	$O(n)$ steps
Cross off next $\texttt{1}$ from first tape	
Cross off next $\texttt{0}$ from second tape	
If both tapes reach $\square$ simultaneously, accept	$O(n)$ steps
running time:	$O(n)$

Running time vs model

How about a Java program? $L_{01} = \{0^n 1^n \mid n \geqslant 0\}$

```
M(int[] x) {
  n = x.len;
  if (n % 2 != 0) reject();
  for (i = 0; i < n/2; i++) {
    if (x[i] != 0) reject();
    if (x[n-i+1] != 1) reject();
  }
  accept();
}
```

running time: $O(n)$

Running time can change depending on the model

1-tape TM	2-tape TM	Java
$O(n \log n)$	$O(n)$	$O(n)$

Measuring running time

What does it mean when we say

This algorithm runs in time T

One "time unit" in

Java

```
if (x > 0)
  y = 5*y + x;
```

Random access machine

```
write r3
```

Turing machine

$\delta(q_3, \mathtt{a}) = (q_7, \mathtt{b}, R)$

all mean different things!

Efficiency and the Church–Turing thesis

Church–Turing thesis says all these have the same computing power...

Java

Turing machine

RAM

Multitape TM

...without considering running time

Cobham–Edmonds thesis

An extension to Church–Turing thesis, stating

For any realistic models of computation M_1 and M_2

M_1 can be simulated on M_2 with at most polynomial slowdown

So any task that takes time $t(n)$ on M_1 can be done in time (say) $O(t^3)$ on M_2

Efficient simulation

The running time of a program depends on the model of computation

1-tape TM   2-tape TM   RAM   Java

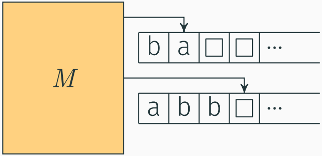
slow → fast

But if you ignore polynomial overhead, the difference is irrelevant

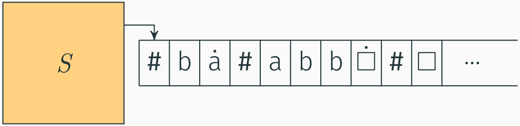
Every reasonable model of computation can be simulated efficiently on any other

Example of efficient simulation

Recall simulating two tapes on a single tape



$$\Gamma = \{a, b, \square\}$$

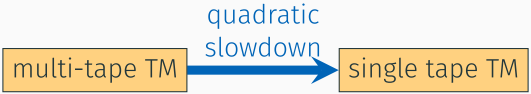


$$\Gamma = \{a, b, \square, \dot{a}, \dot{b}, \dot{\square}, \#\}$$

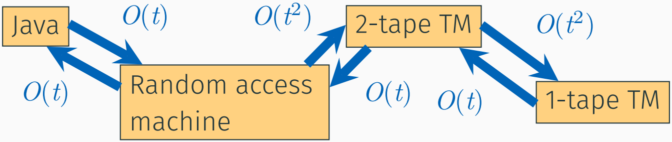
Running time of simulation

Each move of the multitape TM might require traversing the whole single tape

1 step of 2-tape TM	$\Rightarrow$	$O(s)$ steps of single tape TM $s =$ right most cell ever visited
after t steps	$\Rightarrow$	$s \leqslant n + 2t + O(1) = O(n + t)$ $n =$ input length
t steps of 2-tape	$\Rightarrow$	$O(ts) = O(t(n + t))$ single tape steps $= O(t^2)$ if $t \geqslant n$



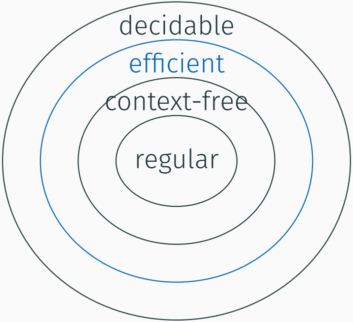
Simulation slowdown



Cobham–Edmonds thesis:

M_1 can be simulated on M_2 with at most polynomial slowdown

The class P



P is the class of languages that can be decided on a TM with polynomial running time

By Cobham–Edmonds thesis, they can also be decided by any realistic model of computation e.g. Java, RAM, multitape TM

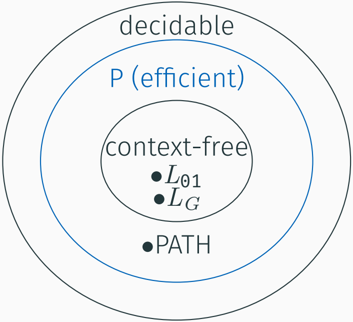
Examples of languages in P

P is the class of languages that are decidable in polynomial time (in the input length)

$L_{\mathtt{01}} = \{\mathtt{0}^n\mathtt{1}^n \mid n \geqslant 0\}$

$L_G = \{w \mid \text{CFG } G \text{ generates } w\}$

$\text{PATH} = \{\langle G, s, t\rangle \mid \text{Graph } G \text{ has a path from node } s \text{ to node } t\}$



Context-free languages in polynomial time

Let L be a context-free language, and G be a CFG for L in Chomsky Normal Form

CYK algorithm:

If there is a production $A \to x_i$
    Put A in table cell $T[i, 1]$
For cells $T[i, \ell]$
    If there is a production $A \to BC$
        where B is in cell $T[i, j]$
        and C is in cell $T[i + j, \ell - j]$
    Put A in cell $T[i, \ell]$

ℓ						
5						
4						
3						
2	$S\|A$	B	$S\|C$	$S\|A$		
1	B	$A\|C$	$A\|C$	B	$A\|C$	
	1	2	3	4	5	i
	b	a	a	b	a	

On input x of length n, running time is $O(n^3)$

PATH in polynomial time

$$\text{PATH} = \{\langle G, s, t\rangle \mid \text{Graph } G \text{ has a path from node } s \text{ to node } t\}$$

G has n vertices, m edges

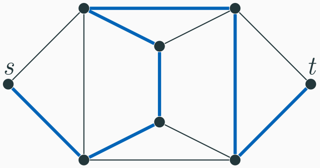
$M =$ On input $\langle G, s, t\rangle$
  where G is a graph with nodes s and t
  Place a mark on node s
  Repeat until no additional nodes are marked: $O(n)$
    Scan the edges of G $O(m)$
    If some edge has both marked and unmarked endpoints
      Mark the unmarked endpoint
  If t is marked, accept

running time: $O(mn)$

Hamiltonian paths

A Hamiltonian path in G is a path that visits every node exactly once

$$\text{HAMPATH} = \{\langle G, s, t\rangle \mid \text{Graph } G \text{ has a Hamiltonian path from node } s \text{ to node } t\}$$



We don't know if HAMPATH is in P, and we believe it is not