

# Efficient Turing Machines

CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN

Chinese University of Hong Kong

Fall 2016

# Undecidability of PCP (optional)

# Undecidability of PCP

$PCP = \{ \langle T \rangle \mid T \text{ is a collection of tiles} \\ \text{contains a top-bottom match} \}$

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

$\langle M \rangle \quad \longmapsto \quad T$  (collection of tiles)  
 $M$  accepts  $w \quad \iff \quad T$  contains a match

**Idea:** Matches represent **accepting history**

$\#q_0ab\%ab\#xq_1b\%ab\#\dots\#xx\%xq_ax\#$

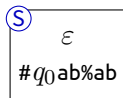
$\#q_0ab\%ab\#xq_1b\%ab\#\dots\#xx\%xq_ax\#$

|               |          |     |     |      |     |     |      |          |     |
|---------------|----------|-----|-----|------|-----|-----|------|----------|-----|
| $\varepsilon$ | $\#q_0a$ | $b$ | $a$ | $\%$ | $a$ | $b$ | $\#$ | $xq_1\%$ | ... |
| $\#q_0ab\%ab$ | $\#xq_1$ | $b$ | $a$ | $\%$ | $a$ | $b$ | $\#$ | $x\%q_2$ |     |

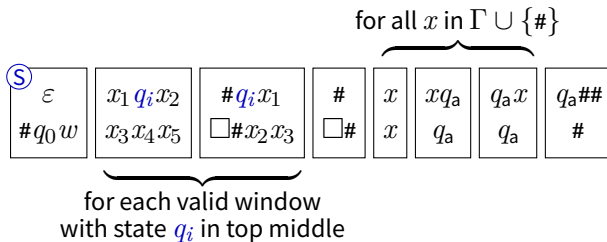
# Undecidability of PCP

$$\begin{aligned} \langle M \rangle &\longmapsto T \text{ (collection of tiles)} \\ M \text{ accepts } w &\iff T \text{ contains a match} \end{aligned}$$

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



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



# Undecidability of PCP

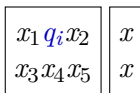
tile type

purpose

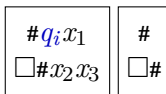
---



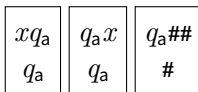
represents **initial configuration**



represents **valid transitions** between configurations



adds **blank spaces** before # if necessary



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

$\#xx\%xq_ax\#xx\%xq_a\#\dots\#q_a\#\#$

$\#xx\%xq_ax\#xx\%xq_a\#\dots\#q_a\#\#$

|     |        |        |           |
|-----|--------|--------|-----------|
| $x$ | $xq_a$ | $q_ax$ | $q_a\#\#$ |
| $x$ | $q_a$  | $q_a$  | $\#$      |

## Undecidability of PCP

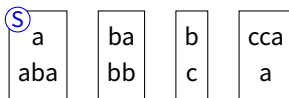
If  $M$  rejects on input  $w$ , then  $q_{rej}$  appears on the bottom at some point, but it cannot be matched on top

If  $M$  loops on  $w$ , then matching goes on forever

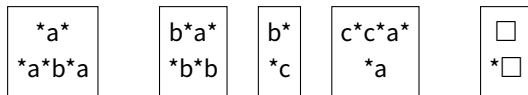


## Getting rid of the starting tile

We assumed that one tile is marked as the starting tile



We can simulate this assumption by changing tiles a bit

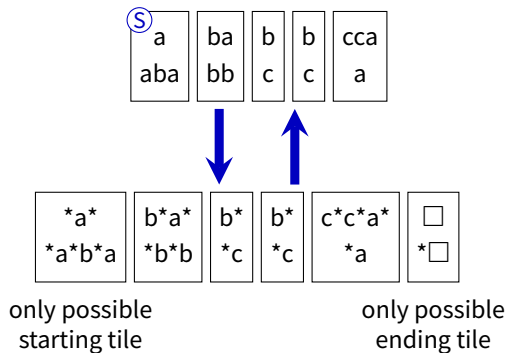


“starting tile”  
begins with \*

“middle tiles”

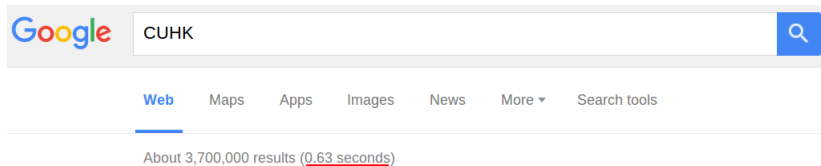
“ending tiles”

## Getting rid of the starting tile



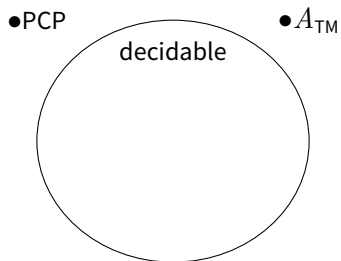
Polynomial time

# Running time



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

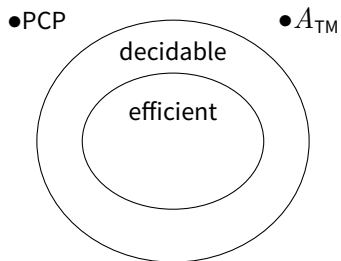
# Efficiency



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  $t_M(n)$ :

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

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

---

|   |                |
|---|----------------|
| $M$ : On input $x$ , until you reach #          | $O(n)$ times   |
| Read and cross off 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

$$L = \{0^n 1^n \mid n \geq 0\}$$

---

*M*: On input  $x$ ,

|  |                |
|--|----------------|
| Check that the input is of the form $0^*1^*$ | $O(n)$ steps   |
| Until everything is crossed off:             | $O(n)$ times   |
| Cross off the leftmost 0                     | } $O(n)$ steps |
| Cross off the following 1                    |                |
| If everything is crossed off, accept         | $O(n)$ steps   |

---

running time:  $O(n^2)$



## A faster way

$$L = \{0^n 1^n \mid n \geq 0\}$$

---

*M*: On input  $x$ ,

Check that the input is of the form  $0^* 1^*$

$O(n)$  steps

Until everything is crossed off:

$O(\log n)$  times

Find **parity** of number of 0s

Find **parity** of number of 1s

If the parities don't match, reject

Cross off **every other** 0 and **every other** 1

}  $O(n)$  steps

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?

$$L = \{0^n 1^n \mid n \geq 0\}$$

---

*M*: On input  $x$ ,

Check that the input is of the form  $0^* 1^*$   $O(n)$  steps

Copy  $0^*$  part of input to second tape  $O(n)$  steps

Until  $\square$  is reached:

    Cross off next 1 from first tape  $\left. \vphantom{\begin{array}{l} \text{Cross off next 1 from first tape} \\ \text{Cross off next 0 from second tape} \end{array}} \right\} O(n)$  steps

    Cross off next 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?

```
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();  
}
```

$$L = \{0^n 1^n \mid n \geq 0\}$$

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
```

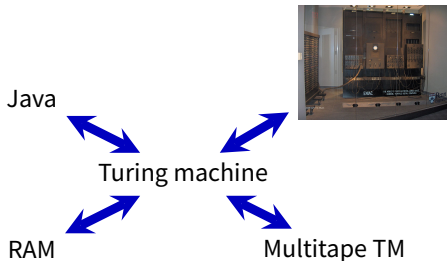
Turing machine

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

all mean different things!

# Efficiency and the Church–Turing thesis

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



...without considering running time

## Cobham–Edmonds thesis

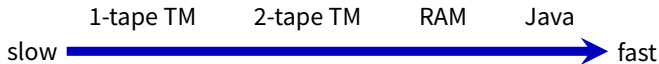
An extension to Church–Turing thesis, stating

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

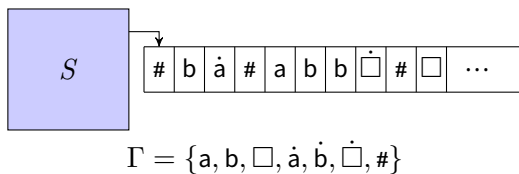
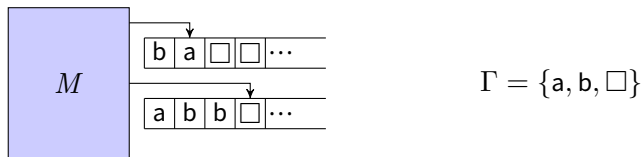


But if you ignore polynomial overhead, the difference is **irrelevant**

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

## Example of efficient simulation

Recall simulating two tapes on a single tape





## Running time of simulation

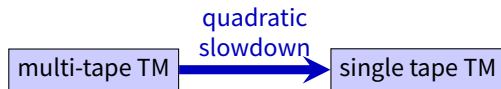
Each move of the multitape TM might require traversing the whole single tape

1 step of 2-tape TM  $\Rightarrow O(s)$  steps of single tape TM

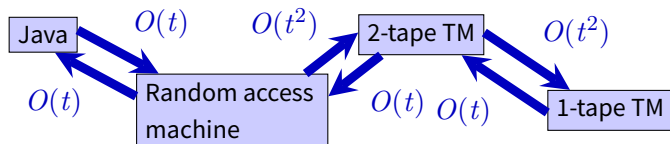
$s =$  right most cell ever visited

after  $t$  steps  $\Rightarrow s \leq 2t + O(1)$

$t$  steps of 2-tape  $\Rightarrow O(ts) = O(t^2)$  single tape steps



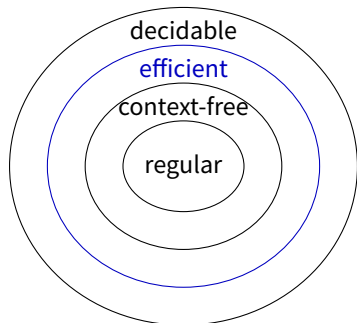
## Simulation slowdown



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

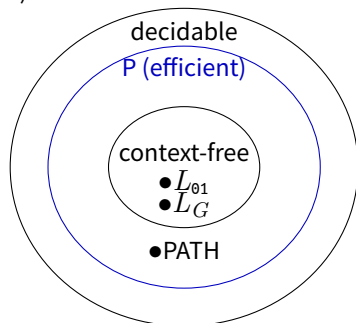
## Examples of languages in P

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

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

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

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



## 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  $A \rightarrow x_i$

Put  $A$  in table cell  $T[i, 1]$

For cells  $T[i, \ell]$

If there is a production  $A \rightarrow BC$

where  $B$  is in cell  $T[i, j]$

and  $C$  is in cell  $T[i + j, \ell - j]$

Put  $A$  in cell  $T[i, \ell]$

|        |       |       |       |       |       |     |
|--------|-------|-------|-------|-------|-------|-----|
| $\ell$ |       |       |       |       |       |     |
| 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

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

Scan the edges of  $G$ .

If some edge has both marked and unmarked endpoints

Mark the unmarked endpoint

If  $t$  is marked, accept

$O(n)$  times

$O(m)$  steps

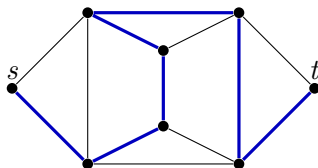
---

running time:  $O(mn)$

# Hamiltonian paths

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

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



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