Efficient Turing Machines

CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN
Fall 2020

Chinese University of Hong Kong
Undecidability of PCP (optional)
Undecidability of PCP

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

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

\[ \langle M, w \rangle \iff C \text{ (collection of tiles)} \]

\( M \) accepts \( w \) \iff \( C \) contains a match

**Idea:** Matches represent accepting history

\[ #_{q_0} a b \% a b \# x_{q_1} b \% a b \# \ldots \# x \% x \# q_a x \# \]

\[ #_{q_0} a b \% a b \# x_{q_1} b \% a b \# \ldots \# x \% x \# q_a x \# \]
Undecidability of PCP

\[ \langle M \rangle \quad \iff \quad C \text{ (collection of tiles)} \]
\( M \) accepts \( w \) \iff \( C \) contains a match

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:

For all \( x \) in \( \Gamma \cup \{\#\} \)

For each valid window with state \( q_i \) in top middle
Undecidability of PCP

<table>
<thead>
<tr>
<th>tile type</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\varepsilon)</td>
<td>represents initial configuration</td>
</tr>
<tr>
<td>(#q_0w)</td>
<td></td>
</tr>
<tr>
<td>(x_1 q_i x_2)</td>
<td>represents valid transitions between configurations</td>
</tr>
<tr>
<td>(x_3 x_4 x_5)</td>
<td></td>
</tr>
<tr>
<td>(#q_i x_1)</td>
<td>adds blank spaces before # if necessary</td>
</tr>
<tr>
<td>(\square # x_2 x_3)</td>
<td></td>
</tr>
<tr>
<td>(#)</td>
<td></td>
</tr>
<tr>
<td>(\square #)</td>
<td></td>
</tr>
<tr>
<td>(x q_a)</td>
<td>matching completes if computation accepts</td>
</tr>
<tr>
<td>(q_a)</td>
<td></td>
</tr>
<tr>
<td>(q_a x)</td>
<td></td>
</tr>
<tr>
<td>(q_a ##)</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td></td>
</tr>
</tbody>
</table>
Once the accepting state symbol occurs, the last two tiles can “eat up” the rest of the symbols

\[
\text{#xx%x} q_a x \text{#xx%x} q_a \# \# q_a \#
\]

\[
\text{#xx%x} q_a x \text{#xx%x} q_a \# \# q_a \#
\]
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)

We can simulate this assumption by changing tiles a bit

“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
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
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:** \( L = \{w\#w \mid w \in \{a, b\}^*\} \)

$M$: On input $x$, until you reach $\#$
- Read and cross of first $a$ or $b$ before $\#$
- Read and cross off first $a$ or $b$ after $\#$
- If mismatch, reject
- If all symbols except $\#$ are crossed off, accept

$$\text{running time: } O(n^2)$$
Another example

\[ L = \{0^n1^n \mid n \geq 0\} \]

\[ M: \text{On input } x, \]

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

Until everything is crossed off:

Cross off the leftmost \(0\) \(O(n)\) times

Cross off the following \(1\) \(O(n)\) steps

If everything is crossed off, accept \(O(n)\) steps

**running time:** \(O(n^2)\)
A faster way

\[ L = \{0^n1^n \mid n \geq 0\} \]

\[ M: \text{On input } x, \]

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

Until everything is crossed off:

Find parity of number of 0s \(O(\log n)\) times
Find parity of number of 1s \(O(n)\) steps

If the parities don’t match, reject \(O(n)\) steps

Cross off every other 0 and every other 1 \(O(n)\) steps

If everything is crossed off, accept \(O(n)\) steps

\[ \text{running time: } O(n \log n) \]
Running time vs model

What if we have a two-tape Turing machine?

\[ L = \{0^n1^n \mid n \geq 0\} \]

\[ M: \text{On input } x, \]
Check that the input is of the form \( \theta^*1^* \) \( \mathcal{O}(n) \) steps
Copy \( \theta^* \) part of input to second tape \( \mathcal{O}(n) \) steps
Until \( \Box \) is reached:
\[ \begin{align*}
\text{Cross off next } 1 & \text{ from first tape} \\
\text{Cross off next } \theta & \text{ from second tape}
\end{align*} \]
If both tapes reach \( \Box \) simultaneously, accept \( \mathcal{O}(n) \) steps

\text{running time:} \quad \mathcal{O}(n)
Running time vs model

How about a Java program?

$$L = \{0^n1^n \mid n \geq 0\}$$

```java
M(int[] x) {
    n = x.length;
    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

<table>
<thead>
<tr>
<th>Model</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-tape TM</td>
<td>(O(n \log n))</td>
</tr>
<tr>
<td>2-tape TM</td>
<td>(O(n))</td>
</tr>
<tr>
<td>Java</td>
<td>(O(n))</td>
</tr>
</tbody>
</table>
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!
Church–Turing thesis says all these have the same computing power...
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 \( M_2 \)
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

\[ \Gamma = \{a, b, \Box\} \]

\[ \Gamma = \{a, b, \Box, \hat{a}, \hat{b}, \checkmark, \#\} \]
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 \( \Rightarrow \) \( s \leq n + 2t + O(1) = O(n + t) \)

\( n = \) input length

t steps of 2-tape \( \Rightarrow \) \( O(ts) = O(t(n + t)) \) single tape steps

\( = O(t^2) \) if \( t \geq n \)

multi-tape TM \( \longrightarrow \) single tape TM
Simulation slowdown

Cobham–Edmonds thesis:

$M_1$ can be simulated on $M_2$ with at most polynomial slowdown
The class $P$ 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)

\[ L_{01} = \{ \theta^n \mathbf{1}^n \mid n \geq 0 \} \]

\[ L_G = \{ w \mid \text{CFG } G \text{ generates } w \} \]

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

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:

- 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

\[ \text{running time: } O(mn) \]
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