Efficient Turing Machines

CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN
Fall 2019

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

PCP = \{ \langle T \rangle \mid T \text{ is a collection of tiles 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

\[
\langle M, w \rangle \quad \leftrightarrow \quad T \text{ (collection of tiles)}
\]

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

Idea: Matches represent accepting history

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

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

\[ \langle M \rangle \quad \mapsto \quad T \quad \text{(collection of tiles)} \]

\[ M \text{ accepts } w \quad \iff \quad T \text{ contains a match} \]

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

\[ \varepsilon \# q_0 \text{ab%ab} \]

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 <strong>initial configuration</strong></td>
</tr>
<tr>
<td>$# q_0 w$</td>
<td></td>
</tr>
<tr>
<td>$x_1 q_i x_2$</td>
<td>represents <strong>valid transitions</strong> between configurations</td>
</tr>
<tr>
<td>$x_3 x_4 x_5$</td>
<td>$x$</td>
</tr>
<tr>
<td>$\square # x_2 x_3$</td>
<td>$\square #$</td>
</tr>
<tr>
<td>$# q_i x_1$</td>
<td>$#$</td>
</tr>
<tr>
<td>$\square # x_2 x_3$</td>
<td>$\square #$</td>
</tr>
<tr>
<td>$x q_a$</td>
<td>$q_a x$</td>
</tr>
<tr>
<td>$q_a$</td>
<td>$q_a$</td>
</tr>
</tbody>
</table>

matching **completes** if computation accepts
Once the accepting state symbol occurs, the last two tiles can “eat up” the rest of the symbols

#\times%\times q_a \times \#\times%\times q_a \# \ldots q_a \#\#

#\times%\times q_a \times \#\times%\times q_a \# \ldots q_a \#\#
If $M$ rejects on input $w$, then $q_{\text{rej}}$ 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
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

- Decidable
- Efficient

• PCP
• $A_{TM}$

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

**running time:** $O(n^2)$
Another example

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

**M**: On input \(x\),

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

Until everything is crossed off:

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

Cross off the following \(1\)

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

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

\[ L = \{ \theta^n 1^n \mid n \geq 0 \} \]

**M**: On input \( x \),

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

Until everything is crossed off:

- Find **parity** of number of \( \theta \)s \( O(\log n) \) times
- Find **parity** of number of \( 1 \)s
- If the parities don’t match, reject
- Cross off every other \( \theta \) and every other \( 1 \)

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

running time:  \( O(n \log n) \)
What if we have a two-tape Turing machine?

\[ L = \{ \theta^n 1^n \mid n \geq 0 \} \]

**M**: On input \( x \),

- Check that the input is of the form \( \theta^* 1^* \) \( O(n) \) steps
- Copy \( \theta^* \) part of input to second tape \( O(n) \) steps

Until \( \square \) is reached:
- Cross off next 1 from first tape \( O(n) \) steps
- Cross off next \( \theta \) from second tape \( O(n) \) steps

If both tapes reach \( \square \) simultaneously, accept \( O(n) \) steps

**running time**: \( O(n) \)
How about a Java program?

```java
L = {0^n1^n | n ≥ 0}

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 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...

- Java
- RAM
- Turing machine
- Multitape TM

...without considering running time
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: slow
- 2-tape TM
- RAM
- Java: 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

\[ s = \text{right most cell ever visited} \]

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

\[ n = \text{input length} \]

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

\[ = O(t^2) \text{ if } t \geq n \]

multi-tape TM \(\quad \Rightarrow \quad\) single tape TM

quadratic slowdown
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)

$L_{01} = \{0^n1 \mid n \geq 0\}$

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

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

PATH = \{ \langle G, s, t \rangle \mid \text{Graph } G \text{ has a path from node } s \text{ to node } t \} \n
G \text{ has } n \text{ vertices, } m \text{ edges} \n
M = \text{On input } \langle G, s, t \rangle \n
where \ G \text{ is a graph with nodes } s \text{ and } t \n
\text{Place a mark on node } s \n
\text{Repeat until no additional nodes are marked:} \quad O(n) \n
\quad \text{Scan the edges of } G \quad O(m) \n
\quad \text{If some edge has both marked and unmarked endpoints} \n
\quad \text{Mark the unmarked endpoint} \n
\text{If } t \text{ is marked, accept} \n
\text{running time: } \quad 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.