Efficient Turing Machines
CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN

Chinese University of Hong Kong

Fall 2017
Undecidability of PCP (optional)
Undecidability of PCP

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

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

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

\[\begin{align*}
M \text{ accepts } w & \iff T \text{ contains a match} \\
\end{align*}\]

**Idea:** Matches represent accepting history

\[
\begin{align*}
#q_0 a b a b # x q_1 b a b # \ldots # x x % x q_a x # \\
# q_0 a b a b # x q_1 b a b # \ldots # x x % x q_a x #
\end{align*}
\]
Undecidability of PCP

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

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

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

\[ \begin{array}{c}
  \varepsilon \\
  \#q_{0}ab%ab
\end{array} \]

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_0 )</td>
<td></td>
</tr>
<tr>
<td>( x_1 ) ( q ) ( x_2 )</td>
<td>( x )</td>
</tr>
<tr>
<td>( x_3 ) ( x_4 ) ( x_5 )</td>
<td>( x )</td>
</tr>
<tr>
<td>( #q_i ) ( x_1 )</td>
<td></td>
</tr>
<tr>
<td>( # ) ( x_2 ) ( x_3 )</td>
<td>( # )</td>
</tr>
<tr>
<td>( xq_a )</td>
<td></td>
</tr>
<tr>
<td>( q_a )</td>
<td>( q_a )</td>
</tr>
<tr>
<td>( q_a )</td>
<td></td>
</tr>
<tr>
<td>( q_a )</td>
<td></td>
</tr>
</tbody>
</table>

represents valid transitions between configurations

adds blank spaces before \( \# \) if necessary

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

\[ \#xx\%x q_a x \#xx\%x q_a \# \ldots \# q_a \# \]

\[ \#xx\%x q_a x \#xx\%x q_a \# \ldots \# q_a \#
\]

\[
\begin{array}{cccc}
  x & x & q_a & q_a \\
  x & q_a & q_a & #
\end{array}
\]
Undecidability of PCP

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

```
  S
 a aba
 aba ba bb b c cca a
```

We can simulate this assumption by changing tiles a bit

```
  *a*
  *a*b*a
  *a*b*a

  b*a*
  b* a b*b
  b* b*b

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

  □*
  □ *□
```

“starting tile” begins with *

“middle tiles”

“ending tiles”
Getting rid of the starting tile

Only possible starting tile

Only possible ending 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
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 off first $a$ or $b$ before #
- Read and cross off first $a$ or $b$ after #
- If mismatch, reject

$O(n)$ steps

If all symbols except # are crossed off, accept $O(n)$ steps

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

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

\[ M: \text{On input } x, \]
\[ \begin{aligned}
\text{Check that the input is of the form } \theta^* 1^* & \quad O(n) \text{ steps} \\
\text{Until everything is crossed off:} & \quad O(n) \text{ times} \\
\text{Cross off the leftmost } \theta & \quad \bigg\} O(n) \text{ steps} \\
\text{Cross off the following } 1 & \\
\text{If everything is crossed off, accept} & \quad O(n) \text{ steps} \\
\end{aligned} \]

\text{running time: } \quad O(n^2)
A faster way

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

\[ M: \text{On input } x, \]
Check that the input is of the form \( \theta^* 1^* \) \( O(n) \) steps
Until everything is crossed off:
\[
\begin{align*}
\text{Find parity of number of } \theta \text{s} & \quad \text{ } O(\log n) \text{ times} \\
\text{Find parity of number of } 1 \text{s} & \quad \text{ } \\
\text{If the parities don’t match, reject} & \quad \text{ } \\
\text{Cross off every other } \theta \text{ and every other } 1 & \quad \text{ } \\
\end{align*}
\]
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?

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

\[ M: \text{On input } x, \]
- 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 \(O(n)\) steps
  - Cross off next 0 from second tape \(O(n)\) steps
- If both tapes reach \(\square\) simultaneously, accept \(O(n)\) steps

\textbf{running time:} \(O(n)\)
Running time vs model

How about a Java program?

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

Running time can change depending on the model:

<table>
<thead>
<tr>
<th>Model</th>
<th>1-tape TM</th>
<th>2-tape TM</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td>(O(n \log n))</td>
<td>(O(n))</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…

…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

The difference depends on the model:

- 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

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

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

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

multi-tape TM \( \xrightarrow{\text{quadratic slowdown}} \) single tape TM
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)

\[ L_{01} = \{ \theta^n 1 \mid n \geq 0 \} \]
\[ L_G = \{ w \mid \text{CFG } G \text{ generates } w \} \]
\[ \text{PATH} = \{ \langle G, s, t \rangle \mid \text{Graph } G \text{ has a path from node } s \text{ to node } t \} \]
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 $T[i, \ell]$

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

\(M = \text{On input } \langle G, s, t \rangle\)
- where \(G\) is a graph with nodes \(s\) and \(t\)
- Place a mark on node \(s\)
- Repeat until no additional nodes are marked:
  - \(O(n)\) times
  - \(O(m)\) steps
    - Scan the edges of \(G\).
    - 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.