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

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

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

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

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

Idea: Matches represent accepting history

\[ \# q_0 a b a b \# x \ q_1 b a b \# \ldots \# x \ x \ x \ q_a x \# \]

\[ \# q_0 a b a b \# x \ q_1 b a b \# \ldots \# x \ x \ x \ q_a x \# \]

\[
\begin{array}{cccccccc}
\varepsilon & \# q_0 a b a b \\
\# q_0 a b a b & \# q_0 a b a b & b & a & \% & a & b & \# & x q_1 \% & x \% q_2 \\
\end{array}
\]

\ldots
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:

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

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 initial configuration</td>
</tr>
<tr>
<td>(#q_0 w)</td>
<td></td>
</tr>
<tr>
<td>(x_1 q_i x_2)</td>
<td>(x) represents valid transitions between configurations</td>
</tr>
<tr>
<td>(x_3 x_4 x_5)</td>
<td>(x)</td>
</tr>
<tr>
<td>(#q_i x_1)</td>
<td>(#) adds blank spaces before # if necessary</td>
</tr>
<tr>
<td>(\Box#x_2 x_3)</td>
<td>(\Box#)</td>
</tr>
<tr>
<td>(x q_a)</td>
<td>(q_a x) matching completes if computation accepts</td>
</tr>
<tr>
<td>(q_a)</td>
<td>(q_a #)</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

#xx%x qₐ x#xx%x qₐ #…# qₐ ##
#xx%x qₐ x#xx%x qₐ #…# qₐ ##

\[ \begin{array}{c|c|c|c}
    x & xqₐ & qₐ x & qₐ # \\
  \hline
    x & qₐ & qₐ & # \\
\end{array} \]
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

\[
\begin{array}{cccc}
S & a & ba & b & cca \\
aba & bb & c & a \\
\end{array}
\]

We can simulate this assumption by changing tiles a bit

\[
\begin{array}{cccc}
*a* & b*a* & b* & c*c*a* \\
*a*b*a & b*b & c & a \\
\end{array}
\]

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

<table>
<thead>
<tr>
<th>$M$: On input $x$, until you reach $#$</th>
<th>$O(n)$ times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read and cross of first a or b before $#$</td>
<td></td>
</tr>
<tr>
<td>Read and cross off first a or b after $#$</td>
<td>$O(n)$ steps</td>
</tr>
<tr>
<td>If mismatch, reject</td>
<td></td>
</tr>
<tr>
<td>If all symbols except $#$ are crossed off, accept</td>
<td>$O(n)$ steps</td>
</tr>
</tbody>
</table>

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

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

\[ M: \text{On input } x, \]
Check that the input is of the form \( \theta^* 1^* \) \( O(n) \) steps
Until everything is crossed off:
- Cross off the leftmost \( \theta \) \( O(n) \) times
- Cross off the following \( 1 \)
\[ \begin{cases} \text{Running time:} & O(n^2) \end{cases} \]
If everything is crossed off, accept \( O(n) \) steps
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 \textit{parity} of number of 0s \( O(\log n) \) times
- Find \textit{parity} of number of 1s
- If the parities don’t match, reject \( O(n) \) steps
- Cross off every other 0 and every other 1

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

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

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:

\[ \left\{ \begin{array}{c}
\text{Cross off next 1 from first tape} \\
\text{Cross off next} \ 0 \ \text{from second tape}
\end{array} \right\} \quad O(n) \text{ steps} \]

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

**running time**: \( O(n) \)
Running time vs model

How about a Java program?

```java
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();
}
```

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

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

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

Random access machine

```plaintext
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…

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

$M$

\[
\begin{array}{c}
\text{b a } \square \square \ldots \\
\text{a b b } \square \ldots \\
\end{array}
\]

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

$S$

\[
\begin{array}{c}
\text{# b } \dot{a} \# a b b \checkmark \# \square \ldots \\
\end{array}
\]

$\Gamma = \{a, b, \square, \dot{a}, \dot{b}, \checkmark, \#\}$
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
\[ s = \text{right most cell ever visited} \]

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 \quad \text{quadratic slowdown} \quad \text{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)

- \( L_{01} = \{0^n1 \mid n \geq 0\} \)
- \( L_G = \{w \mid \text{CFG } G \text{ generates } w\} \)
- \( \text{PATH} = \{\langle G, s, t \rangle \mid \text{Graph } G \text{ has a path from node } s \text{ to node } t\} \)
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:

1. If there is a production $A \rightarrow x_i$, put $A$ in the table cell $T[i, 1]$.
2. 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 $O(n^3)$. 

```
1: B  A|C  A|C  B  A|C  
  1 2  3  4  5  i
b  a  a  b  a

2: S|A  B  S|C  S|A
  3

3:

4:

5:
```
**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
  - Scan the edges of $G$. $O(m)$ steps
  - 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.