Oracle Turing Machines

Sadika Amreen
Reazul Hoque
History

- Alan Mathison Turing (1912–1954), a mathematician, was influential in the development of computer science.

- Turing first described the Turing machine in his 1936 article "On Computable Numbers, with an Application to the Entscheidungsproblem."

- The Turing machine plays a significant role in the creation of the modern computer.
Turing Machines

- A Turing machine is a hypothetical machine.
- Despite its simplicity, the machine can simulate ANY computer algorithm, no matter how complicated it is!
Turing Machines

- For simplicity, we assume that this machine can only process 0, 1 and Blank.
- The machine has a head which is positioned over one of the squares on the tape.
- The head can perform the following three basic operations:
  - **Read** the symbol on the square under the head.
  - **Edit** the symbol by writing a new symbol or erasing it.
  - **Move** the tape left or right by one square so that the machine can read and edit the symbol on a neighboring square.
Turing Machines

- A very simple representation of a Turing Machine would consist of an infinitely long tape (memory).
- Each square on the tape is Blank at the start and symbols can be written over it.
How a Turing Machine Works

We will try to print symbols “1 1 0” on an initially blank tape

We write a “1” on the square under the head

Next, we move the tape left by one square

Now, write “1” on the square under the head

We move the tape left by one square

Finally, write a “0”
How a Turing Machine Works

- Let’s run a simple program of inverting the bits we have written before i.e. from “1 1 0” to “0 0 1”
- We will resume from where we left off i.e. the position of the head remains unchanged
### How a Turing Machine Works

<table>
<thead>
<tr>
<th>Symbol Read</th>
<th>Write Instruction</th>
<th>Move Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>0</td>
<td>Write 1</td>
<td>Move tape to the right</td>
</tr>
<tr>
<td>1</td>
<td>Write 0</td>
<td>Move tape to the right</td>
</tr>
</tbody>
</table>

The current symbol under the head is “0”, so we write “1” and move the tape to the right by one square.

```
1 1 0
```

The current symbol under the head is “1”, so we write “0” and move the tape to the right by one square.

```
1 1 1 1
```
How a Turing Machine Works

<table>
<thead>
<tr>
<th>Symbol Read</th>
<th>Write Instruction</th>
<th>Move Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>0</td>
<td>Write 1</td>
<td>Move tape to the right</td>
</tr>
<tr>
<td>1</td>
<td>Write 0</td>
<td>Move tape to the right</td>
</tr>
</tbody>
</table>

The current symbol under the head is “1”, so we write “0” and move the tape to the right by one square:

1 0 1  
0 0 1

The machine head now reads a “Blank”. How does this indicate that the program does not halt?
# How a Turing Machine Works

## The State Table

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol read</th>
<th>Write Instruction</th>
<th>Move Instruction</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 0</td>
<td>Blank</td>
<td>None</td>
<td>None</td>
<td>Stop state</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>Write 1</td>
<td>Move tape to the right</td>
<td>State 0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Write 0</td>
<td>Move tape to the right</td>
<td>State 0</td>
</tr>
</tbody>
</table>
How a Turing Machine Works

- Inverting the “0 0 1” state back to initial state of “1 1 0” using a two-symbol, three-state Turing Machine

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol Read</th>
<th>Write Instruction</th>
<th>Move Instruction</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 0</td>
<td>Blank</td>
<td>Blank</td>
<td>Left</td>
<td>State 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Right</td>
<td>State 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Right</td>
<td>State 0</td>
<td></td>
</tr>
<tr>
<td>State 1</td>
<td>Blank</td>
<td>Blank</td>
<td>Right</td>
<td>Stop state</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Left</td>
<td>State 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Left</td>
<td>State 1</td>
<td></td>
</tr>
</tbody>
</table>
How a Turing Machine Works

Legend:
- Instruction when 'Blank' symbol is read
- Instruction when '0' symbol is read
- Instruction when '1' symbol is read
Addition using a Turing Machine

- Let’s add 3 and 5 using a Turing Machine

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol Read</th>
<th>Write Instruction</th>
<th>Move Instruction</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td>Blank</td>
<td>-</td>
<td>Left</td>
<td>State 1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>-</td>
<td>Left</td>
<td>State 1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>Left</td>
<td>State 2</td>
</tr>
<tr>
<td>State 2</td>
<td>Blank</td>
<td>1</td>
<td>-</td>
<td>Stop state</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>Stop state</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>-</td>
<td>Left</td>
<td>State 2</td>
</tr>
</tbody>
</table>
Addition using a Turing Machine

1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1

0 1 1 1 1 1 1 1

0 1 1 1 1 1 1 1

0 1 1 1 1 1 1 1
In 1936 Alan Turing proved that it's not possible to decide whether an arbitrary program will eventually halt or run forever.

This was later called the Halting Problem by Martin Davis.

It is one of the first examples of a decision problem.
The Halting Problem

The **halting problem** is the problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running or continue to run forever.

For example,

```python
while (true) continue
```

**Does not halt**; rather, it goes on forever in a infinite loop.

On the other hand,

```python
print "Hello World!"
```

**Does halt**
The Halting Problem

What about more complex problems?

Why does the Halting Problem matter?

• We often want to know if a program converges (halts), but not possible to provide one algorithm that answers this for all programs.

• Many problems are shown to be undecidable by being reduced to halting problem
Oracle Turing Machines

- An oracle machine is an abstract machine used to study decision problems.
- It can be visualized as a Turing machine with a black box, called an oracle, which is able to decide certain decision problems in a single operation.
- The problem can be of any complexity class. Even undecidable problems, like the halting problem, can be used.
Oracle Turing Machines

- An oracle machine, like a Turing machine, includes:
  - a work tape, read/write head and a control mechanism
- In addition to these components, an oracle machine also includes:
  - an oracle tape
  - an oracle head
  - two special states: the QUERY state and the RESPONSE state.
Oracle Turing Machines

- From time to time, the oracle machine may enter the QUERY state. When this happens, the following actions are performed in a single computational step:
  - Content of the oracle tape is read
  - the oracle is consulted, and the contents of the oracle tape are replaced with the solution to that instance of the problem;
  - the state of the oracle machine is changed to RESPONSE either of $Q_{\text{yes}}$ or $Q_{\text{no}}$

The effect of changing to the QUERY state is thus to receive, in a single step, a solution to the problem instance that is written on the oracle tape.
Oracle Turing Machines

Finite Set of Controls

Oracle Tape Head in state $p_0$

Input Tape Head in state $q_0$
### Oracle Turing Machines

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol Read</th>
<th>Write Instruction</th>
<th>Move Instruction</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 0</td>
<td>Blank</td>
<td>Blank</td>
<td>Left</td>
<td>State 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Right</td>
<td>State 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Right</td>
<td>State 0</td>
<td></td>
</tr>
<tr>
<td>State 1</td>
<td>Blank</td>
<td>Blank</td>
<td>Right</td>
<td>Stop state</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Left</td>
<td>State 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Left</td>
<td>State 1</td>
<td></td>
</tr>
<tr>
<td>Q_{yes}</td>
<td>Blank</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>0</td>
<td>....</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>1</td>
<td>....</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>Q_{no}</td>
<td>Blank</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>0</td>
<td>....</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>1</td>
<td>....</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>
Oracle Turing Machines

- In simple words, OTMs can be seen as computations which use subroutines.
- The time spent in these subroutines counts only as **one step**, regardless of complexity.
- This makes OTMs an unrealistic computation model.
Importance of OTMs

- Used widely in computer science theory, to help us study the relative difficulty between different problems.
- They help us encode the notion of Turing reductions.
- They help identify some barriers to proving results in complexity theory, like proving that P≠NP.
**Turing Reduction**

- A Turing reduction from problem A to problem B is a reduction which solves A, assuming the solution to B is already known.
- In simpler terms, it is an algorithm that could be used to solve A if it had available to it a subroutine for solving B.
Applications

- No practical application
- Theoretical application includes:
  - Relativizing proofs
  - Study the relative difficulty between different problems
  - Turing Reduction
Application in Cryptography

- Reductions play a central role in cryptography
- Designer of a crypto-system wants the system to be difficult to break
- In order to provide a “proof” of security for this scheme, the crypto-analysis problem for the new system is computationally equivalent to a reference problem believed to be intractable
- The existence of such reduction imply the security of the new scheme
References

- http://cs.stackexchange.com/questions/40605/why-is-oracle-turing-machine-important
- https://www.youtube.com/watch?v=macM_MtS_w4&feature=iv&src_vid=dNRDvLACg5Q&annotation_id=annotation_3596554153
- Recent Advances in RSA Cryptography by Stefan Katzenbeisser, p. 20-21
Thank You!

Questions?
The Evil Program

```c
bool does_it_halt( char * program, char * input ) {
    if( some terribly clever test for halting )
        { return TRUE; }
    else
        { return FALSE; }
}

bool evil_program( char * program ) {
    if( does_it_halt( program, program ) )
        { while( 1 ) {} return FALSE; }
    else
        { return TRUE; }
}
```

What does evil_program(evil_program) do?

Halting Problem Solution

```python
DEFINE DOES IT HALT (PROGRAM):
{
    RETURN TRUE;
}

THE BIG PICTURE SOLUTION
TO THE HALTING PROBLEM
```