Universal Turing Machine
Turing Machine

move the head left or right by one cell

read

write

sequentially accessed infinite memory

finite memory (state)

next-action look-up table

Variants don’t change which languages are recognizable/decidable
Today

$k$-tape TM

Subroutines & Recursion

Universal TM

Simulating a Random Access Machine

Church-Turing Thesis
Extension: multiple tapes

\( k \)-tape TM

\( k \) different (2-way infinite) tapes

\( k \) different independently controllable heads

input initially on tape 1; tapes 2, 3, ..., \( k \), blank.

Single move:

read symbols under all heads

print (possibly different) symbols under heads

move all heads (possibly in different directions)

go to new state
k-tape TM transition function

\[ \delta(q, a_1, a_2, \ldots, a_k) = (p, b_1, b_2, \ldots, b_k, D_1, D_2, \ldots, D_k) \]

Symbols scanned on the \( k \) different tapes
Symbols to be written on the \( k \) different tapes
Directions to move in (\( D_i \) is one of L, R, S)

Utility of multiple tapes:
makes programming a whole lot easier

Example: \( L = \{ w#w^R \mid w \in \{0,1\}^* \} \)
With single tape, need \( \Omega(n^2) \) steps

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>#</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1’</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With 2 tapes, \( n+1 \) steps:
copy till # to 2\(^{nd} \) tape.
Scan it backwards after that
Can’t compute more with k tapes

Theorem: If $L$ is accepted by a $k$-tape TM $M$, then $L$ is accepted by some 1-tape TM $M'$.

Idea: $M'$ uses $k$ tracks to simulate tapes of $M$

$M'$ will use $2k$ tracks to simulate tapes+heads of $M$
Snapshot of simulation \((k = 2)\)

Single move: 
\[ \delta(q_1,1,1) = (q_2,0,0,R,L) \]

Track \(2i-1\) holds tape \(i\).
Track \(2i\) holds position of head \(i\)
Snapshot of simulation \((k = 2)\)

Single move:
\[ \delta(q_1, 1, 1) = (q_2, 0, 0, R, L) \]

Make two sweeps over the tape (up to the rightmost head)
Snapshot of simulation \((k = 2)\)

Single move:
\[\delta(q_1,1,1) = (q_2,0,0,R,L)\]

First pass: record
(old state, head-1 symbol, head-2 symbol)

\[
\begin{array}{cccc}
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
$ & 0 & 0 & 1 \\
$ & 0 & 1 & 1 \\
$ & 0 & 0 & 1 \\
$ & 0 & 1 & 0 \\
\end{array}
\]
Snapshot of simulation (k = 2)

Single move:
\[ \delta(q_1,1,1) = (q_2,0,0,R,L) \]
Snapshot of simulation \( (k = 2) \)

Single move:
\[
\delta(q_1,1,1) = (q_2,0,0,R,L)
\]

Sweep back, implementing the changes

If \( M \) takes \( T \) steps, \( M' \) takes \( O(T^2) \) steps
Subroutine calls

Mechanism for $M_1$ to “call” $M_2$ on an argument

Goal: $M_1$ calls from state $q_{\text{call}}$ returns to $q_{\text{return}}$

Rename start state of $M_2$ as $q_{\text{call}}$ & halt state $q_{\text{return}}$

$M$ will have state space $Q = Q_1 \cup Q_2$

$(Q_1 \cap Q_2 = \{q_{\text{call}}, q_{\text{return}}\})$
Subroutine calls

\[ \ldots \ldots \ldots \ldots \ldots \# \ a_1 \ a_2 \ \ldots \ \ldots \ a_n \]

\[ M_1 \text{ work space} \]

\[ \\]

\[ q_{\text{call}} \]

\[ M_2 \text{ runs, and when done:} \]

\[ \ldots \ldots \ldots \ldots \ldots \# \ b_1 \ b_2 \ \ldots \ \ldots \ b_k \]

\[ M_1 \text{ work space} \]

\[ q_{\text{return}} \]

\[ M_2 \text{ returned value} \]
Subroutine calls

Mechanism for $M_1$ to “call” $M_2$ on an argument

Goal: $M_1$ calls from state $q_{\text{call}}$ returns to $q_{\text{return}}$

Rename start state of $M_2$ as $q_{\text{call}}$ & halt state $q_{\text{return}}$

$M$ will have state space $Q = Q_1 \cup Q_2$

$(Q_1 \cap Q_2 = \{q_{\text{call}}, q_{\text{return}}\})$

Recursion:
Now $M_2$ can call itself (without adding more states).
$M_1$ may just be a wrapper (“main” function)
Alphabet Reduction

For any TM

\[ M = (Q, \Gamma, \Sigma, B, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}}) \]

there exists an "equivalent" TM

\[ M' = (Q', \Gamma', \Sigma', B', q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}}) \]

with \( \Gamma' = \Sigma' = \{0, 1\} \), \( B' = 0 \)

Will need to encode input in \( \Sigma^* \) using \( \{0, 1\} \)

Let \( \Sigma = \{1, 2, \ldots, d\} \), \( \Gamma = \{0, 1, 2, \ldots, k-1\} \) (\( B=0 \))

Encode \( i \in \Gamma \) in binary using \( \lceil \log k \rceil \) bits

\( n \) characters on \( M' \)’s tape \( \rightarrow \) \( O(n \log k) \) bits for \( M' \)
Alphabet Reduction

$|Q'| = O(k \log k \cdot |Q|)$

A single step becomes $O(\log k)$ steps.
Universal TM

So far: for each problem we design a new TM

Early Computer “Programming”

Rewire the computer!

ENIAC
(1946-1955)
Programmers:
Kay McNulty,
Betty Jennings,
Betty Snyder,
Marlyn Wescoff,
Fran Bilas,
Ruth Lichterman
Universal TM

Modern Computers: Program is just data

The computer’s finite control remains the same, doing the following in a loop:

**Read an instruction from the address in PC register**

**Carry out that instruction** (possibly reading from/writing to other addresses)

**Update the PC** (as specified by the instruction)

The alphabet of the computer is also the same for all programs
Universal TM

Modern Computers: Program is just data

Universal TM $U$:

Accepts as input $z\#w$
where $z$ is interpreted as the description of a TM (with $\Sigma = \Gamma = \{0,1\}$)
and $w$ as an input to it

$Simulates$ the execution of $M_z$ on $w$:

$U(z\#w)$ halts iff $M_z(w)$ halts
$U(z\#w)$ accepts iff $M_z(w)$ accepts

Will use 3 tapes and a larger alphabet $\Gamma_U$
Universal TM

Given a string $z$, what is the TM $M_z$?

For $M_z$ we fix $\Sigma = \Gamma = \{0,1\}$,
$q_{\text{start}} = 0$, $q_{\text{accept}} = 1$, $q_{\text{reject}} = 2$,

Then $z$ can just specify the transition function
(which implicitly specifies $Q$ as well)

e.g., $z$ is of the form $\# \ 0^h \ 1 \ 0^i \ 1 \ 0^j \ 1 \ 0^k \ 1 \ 0^d \ # \ldots$
indicates $\delta(q_h,i) = (q_k,j,D_d)$ etc.
with $d \in \{1,2\}$, and $D_1=\text{L}$, $D_2=\text{R}$

if $z$ is not of this form, $M_z$ is the “null TM”
which rejects all inputs
Universal TM

1. Check syntax of \( z \)

2. Copy \( w \) to tape 2, 0 to tape 3

3. In a loop, until a halting state in tape 3:
   Scan tape 1 to find the correct transition, and update tapes 2 & 3.

A 3 tape Universal TM:

Tape of \( M_z \) (initialized to \( w \)).
Head where \( M_z \)'s head is
State of \( M_z \)
Language of Universal TM

Language recognized by $U$:

$$L(U) = \{ z#w \mid U \text{ accepts } z#w \}$$
$$= \{ z#w \mid M_z \text{ accepts } w \}$$

Will later see:

$L(U)$ is undecidable!
A Higher-Level Model: RAM

RAM: Random Access Machine

A “CPU” that can directly access any location in an infinite array of integers, by specifying its address

CPU has a finite number of integer registers, including a “program counter” (automatically incremented after each step)

Instructions written in the infinite memory

<table>
<thead>
<tr>
<th>Load, 〈Reg〉, 〈addr〉</th>
<th>LoadI, 〈Reg〉, 〈addr〉</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store, 〈Reg〉, 〈addr〉</td>
<td>StoreI, 〈Reg〉, 〈addr〉</td>
</tr>
<tr>
<td>LoadC, 〈Reg〉, 〈num〉</td>
<td>Add, 〈Reg〉, 〈Reg〉</td>
</tr>
<tr>
<td>JmpZero, 〈Reg〉, 〈addr〉</td>
<td>Halt</td>
</tr>
</tbody>
</table>
A Higher-Level Model: RAM

RAM: Random Access Machine
Input follows code. Rest of memory has 0s.

Program counter initialized to 1 and incremented after each step (unless overwritten by an instruction)

Realistic cost: Executing an instruction costs $O(\log k)$ steps where $k$ is max of absolute values of the integers in the instruction

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>⟨Reg⟩, ⟨addr⟩</td>
</tr>
<tr>
<td>LoadI</td>
<td>⟨Reg⟩, ⟨addr⟩</td>
</tr>
<tr>
<td>Store</td>
<td>⟨Reg⟩, ⟨addr⟩</td>
</tr>
<tr>
<td>StoreI</td>
<td>⟨Reg⟩, ⟨addr⟩</td>
</tr>
<tr>
<td>LoadC</td>
<td>⟨Reg⟩, ⟨num⟩</td>
</tr>
<tr>
<td>Add</td>
<td>⟨Reg⟩, ⟨Reg⟩</td>
</tr>
<tr>
<td>JmpZero</td>
<td>⟨Reg⟩, ⟨addr⟩</td>
</tr>
<tr>
<td>Halt</td>
<td></td>
</tr>
</tbody>
</table>
TM simulating a RAM

Use a tape to hold the register contents, another to hold the memory (array) contents. Also an input tape & work tape.

All integers are encoded in binary

Memory tape is a list of pairs (addr,val) for all the locations addressed by the RAM so far, +code+input locations. Initialized from code built into finite control, and input tape.

For each RAM step, our TM does the following:
- Scan the memory & register tape and copy information for current instruction to the work tape.
- Compute changes to registers & memory.
- Update the register & memory tapes (shifting as necessary)
TM simulating a RAM

If RAM takes $T$ time steps then the numbers accessed at any step are $O(T)$ bits long. Our TM uses $O(T)$ tape cells and polynomial($T$) time.

For this the (addr, val) representation of memory tape is important. If memory tape simulated the array contiguously, will incur exponential blow-up.
Church-Turing Thesis

A “central dogma” of Computer Science:

A TM can simulate any “physically realizable” model of computation.

Remains true even with probabilistic computation and even quantum computation

(Open whether these models allow polynomial-time computation of problems which a TM cannot solve in polynomial-time)