Computational Models — Lecture 7[1]

Handout Mode

Ronitt Rubinfeld and Iftach Haitner.

Tel Aviv University.

April 11/13, 2016

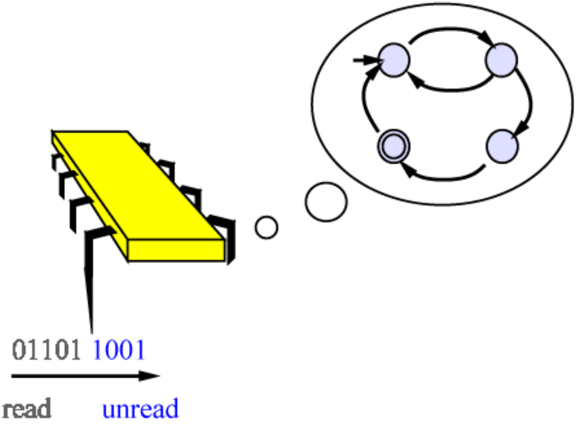
[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

Talk Outline

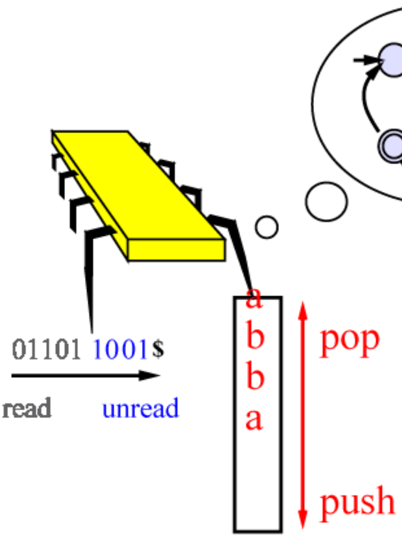
- Turing Machines (TMs)
- Multitape TMs, RAMs
- Church-Turing Thesis

- Sipser's book, 3.1 , 3.2, & 3.3

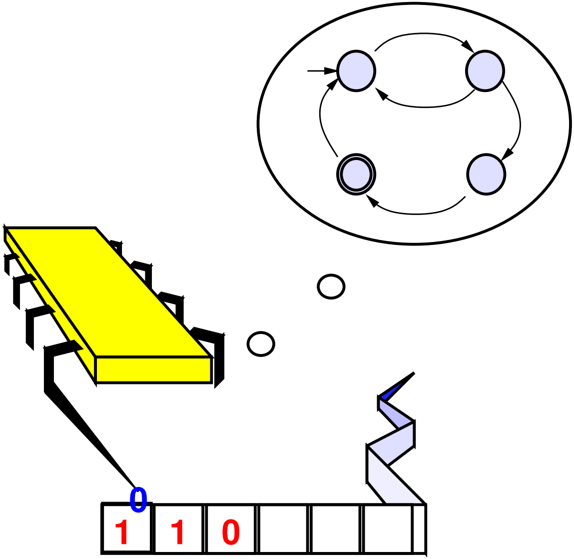
A Finite Automaton



A Pushdown Automaton



A Turing Machine



Part I

Turing Machines (TMs)

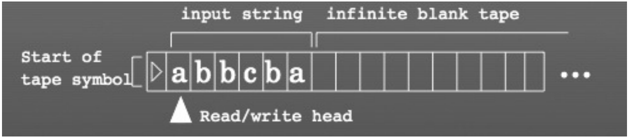
Turing Machines

- Machines so far (DFA, PDA) read input only once

Turing Machines

- Can go back and forth over the input
- Can overwrite the input
- Can write information on tape and come back to it later

- Input string is written on a tape:



- At each step, machine reads a symbol, and then
  - writes a new symbol, and
  - moves read/write head to either right or left.
  - changes its state

TM vs. PDA vs. DFA

- A Turing machine can both write on the tape and read from it.
- A PDA is restricted to reading from the stack in LIFO manner.
- A DFA has no media for writing anything – it must all be in its finite state.
- The TM read-write head can move both to the left and to the right
- The TM read-write tape is infinite to the right
- The special final (accepting/rejecting) states of TM take immediate effect (so the head need not be at some special position)

Effects of a single step

- Changes current state
- Changes head position and tape content at current position.

- Each step has very local, small effect.
- Yet, many small effects can accumulate to a meaningful task.

Example $B = \{w\#w \colon w \in \{0,1\}^*\}$

Algorithm 1 (A TM deciding B (pseudocode))

1. Check for a single $\#$,
   - If false, reject.
2. Zig-zag across the tape, check identical letters, and replace them by X.
   - If not identical, reject.
3. When all the letters left of $\#$ are marked X, check for remaining letters right of $\#$
   - If there are remaining letters, reject.
   - Otherwise, accept

Example $B = \{w\#w \colon w \in \{0,1\}^*\}$ cont.

- Input: $0\ 1\ 1\ 0\ 0\ 0\ \#\ 0\ 1\ 1\ 0\ 0\ 0$
- $\bar{0}\ 1\ 1\ 0\ 0\ 0\ \#\ 0\ 1\ 1\ 0\ 0\ 0$
- $X\ \bar{1}\ 1\ 0\ 0\ 0\ \#\ 0\ 1\ 1\ 0\ 0\ 0$

…

- $X\ 1\ 1\ 0\ 0\ 0\ \#\ \bar{0}\ 1\ 1\ 0\ 0\ 0$
- $X\ 1\ 1\ 0\ 0\ 0\ \bar{\#}\ X\ 1\ 1\ 0\ 0\ 0$

…

- $\bar{X}\ 1\ 1\ 0\ 0\ 0\ \#\ X\ 1\ 1\ 0\ 0\ 0$
- $X\ \bar{1}\ 1\ 0\ 0\ 0\ \#\ X\ 1\ 1\ 0\ 0\ 0$
- $X\ X\ \bar{1}\ 0\ 0\ 0\ \#\ X\ 1\ 1\ 0\ 0\ 0$

…

- $X\ X\ X\ X\ X\ X\ \#\ X\ X\ X\ X\ X\ X$

Turing Machine – Formal Definition

Definition 2 (Turing machine)

A Turing machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where

- Q is a finite set of states.
- Σ the input alphabet not containing the blank symbol $\sqcup$.
- Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$.
- $\delta : Q \setminus \{q_a, q_r\} \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}$ is the transition function.
- $q_0 \in Q$ is the start state.
- $q_a \in Q$ is the accept state.
- $q_r \in Q \setminus \{q_a\}$ is the reject state.

Transition Function δ

$$\delta : Q \setminus \{q_a, q_r\} \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}$$

Informally, $\delta(q, a) = (r, b, L)$ “means":
in state q where head reads tape symbol a, the machine:

- Writes b over a ($a = b$ is possible),
- Enters state r,
- Moves the head left (this is what the L stands for).

Model of Computation (informal)

Before we start:

- Input $w = w_1 w_2 \ldots w_n \in \Sigma^*$ is placed on n leftmost tape squares, one tape square per input letter.

  Rest of tape contains blanks $\sqcup$
- since $\sqcup \notin \Sigma$, leftmost blank indicates end of input.
- read/write head is on leftmost square of tape

The “computation":
M “computes" according to transition function δ.
Computation continues until q_a or q_r is reached. Otherwise M runs forever.

- If M tries to move head beyond left-hand-end of tape, it doesn’t move (still M does not crash)

Configurations

A TM configuration is a convenient notation for recording the state, head location, and tape contents of a TM in a given instant. Think of it as a snapshot.

For example, configuration $1011q_7 0111$ means:

- Current state is q_7
- Left hand side of tape (to the left of the head) is 1011
- Right hand side of tape is 0111 followed by infinite number of $\sqcup$'s
- Head is on 0 (leftmost entry of right hand side).
  (i.e., head location is fifth cell)

Configuration is a finite string in $\Gamma^* Q \Gamma^*$.

Special Configurations

- Starting configuration: $q_0 w$
- Accepting configuration: $w_0 q_a w_1$
- Rejecting configuration: $w_0 q_r w_1$
- Halting configurations: $w_0 q_a w_1$ and $w_0 q_r w_1$

The yield relation

Configuration C yields C' with respect to TM $M = (\ldots, \delta, \ldots)$, if the transition from C to C' is justified by δ:

Let $C = (w_1 qaw_2)$ yields $C' = (w'_1 q' w'_2)$, if $\delta(q, a) = (q', b, X)$, and

1. $Head(C')$ (location of head in C') is $Head(C) - 1$ if $X = L$ and $Head(C) + 1$ otherwise (i.e., $X = R$).
2. The content of the tape-cell at location $Head(C)$ in C' is b
3. All but the $Head(C)$'th tape cell in both configurations are the same
4. $|C| = |C'|$ (not necessarily condition, but implies uniqness)

Example: Configuration $uaqbv$ yields $uq'acv$, if $\delta(q, b) = (q', c, L)$.

Special cases:

1. If $Head(C) = 1$ (i.e., head is at left end) and $\delta(q, a) = (q', b, L)$, then item 2 changes to $Head(C')$=$Head(C)$
2. $C = wq$ implies C', if $wq\sqcup$ implies C'

   The new configuration is longer, as it "annexed" one blank. This allows configurations to grow in length with computation.

Accepting and rejecting a word

A sequence of configurations $C_1, C_2, \ldots, C_k$ is valid, with respect to TM M and $w \in \Sigma^*$, if

1. C_1 is the start configuration of M on w (i.e., $C_1 = q_0 w$)
2. Each C_i yields C_{i+1}

A pair of TM M and input $w \in \Sigma^*$, induces a single, (possibly) infinite sequence of configurations $C_1, C_2, C_3 \ldots$, that any prefix of it is a valid sequence, and any valid sequence is its prefix.

A sequence of configurations $C_1, \ldots, C_k$ is accepting, if C_k is an accepting configuration.

TM M accepts an input $w \in \Sigma^*$, if exists a **valid** with respect to M and w, **accepting** sequence of configurations.

A sequence of configurations $C_1, \ldots, C_k$ is rejecting, if C_k is a rejecting configuration.

TM M rejects an input $w \in \Sigma^*$, if exists a **valid** with respect to M and w, **rejecting** sequence of configurations.

Enumerable and decidable languages

The strings in Σ^* accepted by a TM M, denoted $L(M)$, is the language of M.

Definition 3

A language is recursively enumerable, $\mathcal{RE}$, if some TM accepts it.
(In the book called Turing-recognizable)

On given input, a TM may

- halt — either accept or reject
- loop — not halt (run forever)

Major concern: In general, we never know if the TM will halt.

Definition 4

A TM M decides a language, if it accepts it, and halts on every word in Σ^*.

Such TM is called a decider.

Definition 5

A language is decidable, $\mathcal{R}$, if some TM decides it.
(In the book called Turing-decidable, also known as, recursive)

Part II

Examples

Example $A = \{0^{2^n} : n \geq 0\}$

Algorithm 6 (A TM deciding A (pseudocode))

On input w:

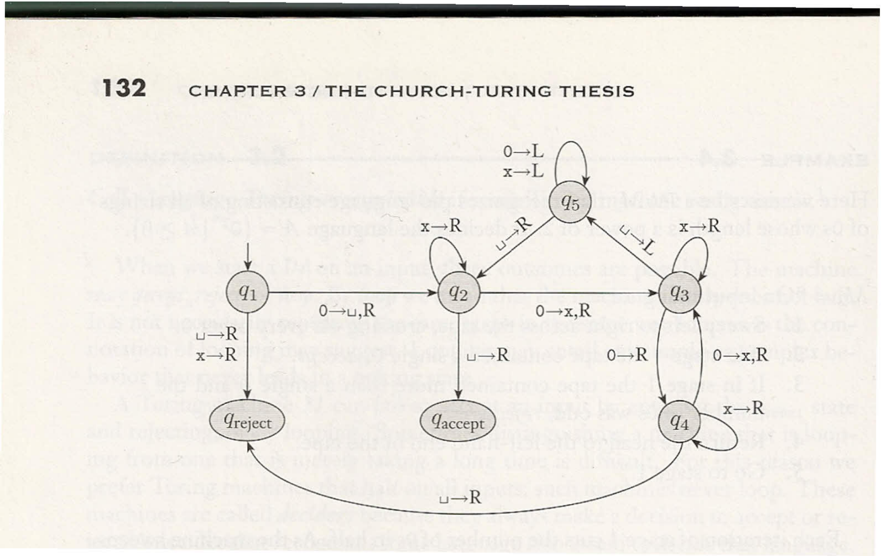
1. Reject, if tape is empty.
2. Accept, if tape contains a single 0.
3. Reject, if tape contains odd number of 0.
4. Move to the right, “erasing" every other 0
5. Return to the start of the tape, and go to Step 2

Example $A = \{0^{2^n} : n \geq 0\}$ – TM formal definition

Definition 7 ($M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_a, q_r)$)

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_a, q_r\}$
- $\Sigma = 0$ input alphabet
- $\Gamma = \{0, x, \sqcup\}$ tape alphabet
- q_1 start state
- q_a accept state
- q_r reject state

Example $A = \{0^{2^n} : n \geq 0\}$ – Transition Function



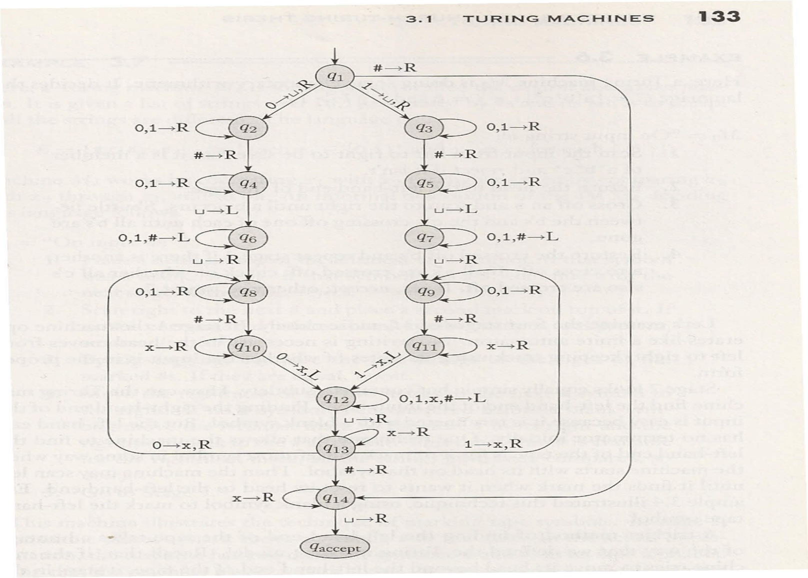
- $\delta(q, a) = (q', L)$, stands for $\delta(q, a) = (q', a, L)$

Example $B = \{w\#w \colon w \in \{0,1\}^*\}$ – TM formal definition

Definition 8 ($M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_a, q_r)$)

- $Q = \{q_1, \ldots, q_{14}, q_a, q_r\}$.
- $\Sigma = \{0, 1, \#\}$ input alphabet
- $\Gamma = \{0, 1, \#, x, \sqcup\}$ tape alphabet
- q_1 start state
- q_a accept state
- q_r reject state

Example $B = \{w\#w \colon w \in \{0,1\}^*\}$ – Transition Function



- If $\delta(q,a)$ is undefined, it means $\delta(q,a) = (q_r,\cdot,\cdot)$

Example: $C = \{a^i b^j c^k : i \cdot j = k \ \wedge \ i, j, k \geq 1\}$

Algorithm 9 (A TM deciding C (pseudocode))

1. Scan from left to right to check that input is $a^+ b^+ c^+$
2. Return to start of tape
3. Cross off one a and scan right until b occurs.
   Shuttle between b's and c's, crossing off one of each, until all b's are gone.
4. Restore the crossed-off b's and repeat previous step if more a's exist.
   If all a's crossed off, check if all c's crossed off. If yes, accept, otherwise reject.

Example: Element distinctness

Consider the element distinctness problem

$E = \{\#x_1\#x_2\#\ldots\#x_\ell : x_i \in \{0,1\}^* \wedge i \neq j \implies x_i \neq x_j\}$

Verbally,

- List of strings in $\{0,1\}^*$ separated by $\#$'s.
- List is in language (& machine should accept), if all strings are different.

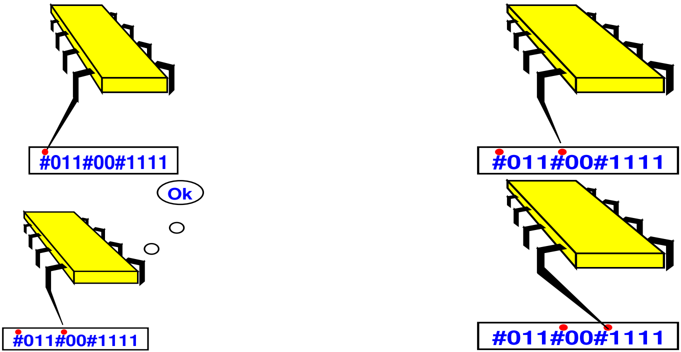
Decider for $E = \{\#x_1\#x_2\#\ldots\#x_\ell : x_i \in \{0,1\}^* \wedge i \neq j \implies x_i \neq x_j\}$

Algorithm 10 (A TM deciding E (pseudocode))

Input: w

1. Place a “mark" on leftmost tape symbol. If symbol not $\#$, reject.
2. Scan right to next $\#$ and place mark on top. If none, reject.
3. Compare the two strings to right of marked $\#$'s (how?). If equal, reject.
4. If possible, move rightmost mark to next $\#$ on right and go to Step 3.
5. If possible, move leftmost mark to next $\#$ on right and rightmost mark to $\#$ after that, and go to Step 3.
6. Accept.

Decider for $E = \{\#x_1\#x_2\#\ldots\#x_\ell : x_i \in \{0,1\}^* \wedge i \neq j \implies x_i \neq x_j\}$



Question 11

How do we “mark” a symbol?

Answer: For each tape symbol $\#$, add tape symbol $\overset{\bullet}{\#}$ to the tape alphabet Γ.

Part III

Alternative Definitions

Turing machine variants

For example, suppose the Turing machine head is allowed to stay put.

$$\delta : Q \setminus \{q_a, q_r\} \times \Gamma \mapsto Q \times \Gamma \times \{L, R, S\}$$

Question 12

Does this add any power?

Answer: No. Replace each S transition with two transitions: R then L. (ahmm ... why not vice-versa?)

Question 13

Does this reduce power?

Machine M emulates machine N, if M **accepts** the same inputs and **halts** on the same inputs as N does.

Computational models A abd B are equivalent, if there is two-way emulation: every machine in model A has a machine in model B that emulates it, and vice versa.

Multitape Turing machines

- Constant number of infinite tapes
- Each tape has its own head
- Initially, input string is placed on tape 1 and rest tape are blank

Transition function is of the form

$$\delta : Q \setminus \{q_a, q_r\} \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, R\}^k$$

The expression $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$ "means":
assuming the machine is in state q_i and heads 1 through k reading $a_1, \ldots, a_k$:

- the machine goes to state q_j,
- heads 1 thought k write $b_1, \ldots, b_k$,
- each head moves right or left as specified.

Multitape TM's, cont

Transition function is of the form $\delta\colon Q \setminus \{q_a, q_r\} \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, R\}^k$.

Configuration $w_1 q w_1' \$\ w_2 q w_2' \$\ \ldots \$\ w_k q w_k'$ "means":

1. State is q
2. Content of i'th tape is $w_i w_i'$
3. Head location of i'th tape is $|w_i| + 1$

Starting configuration (for input w) is $qw\$\ q\$\ \ldots \$\ q$.

The yield relation, and accepting and rejecting an input, are defined analogously to the single tape case.

Equivalence of multitape TM and singletape ones

It is clear that a multitape TM can emulate a singletape TM.

Theorem 14

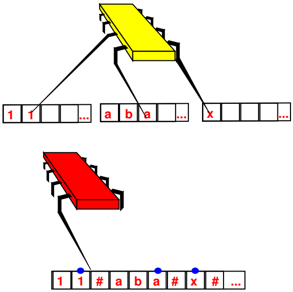
For any multitape TM there exists a singletape TM that emulates it.

Corollary 15

A language is enumerable [resp., decidable], if and only if there is some multitape Turing machine that accepts [resp., decides] it.

Proof: We will show how to "convert" a multitape TM, M, into an equivalent singletape TM, S.

Emulation idea



- S emulates k tapes of M by storing them all on a single tape with delimiter $\#$.
- S marks the current positions of the k heads by placing • "above" the letters in current positions. It "knows" which tape the mark belongs to by counting (up to k) from the $\#$'s to the left.

The emulator

Algorithm 16 (The emulator (pseudocode))

On input $w = w_1 \cdots w_n$:

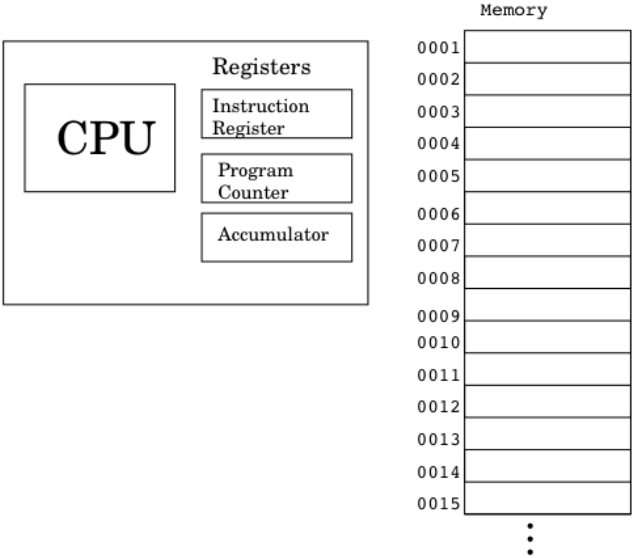
1. Write on the tape $\# \overset{\bullet}{w}_1 \; w_2 \cdot w_n \# \overset{\bullet}{\llcorner\lrcorner} \# \overset{\bullet}{\llcorner\lrcorner} \# \cdots \#$
2. Scan the tape from first $\#$ to $(k+1)$-st $\#$ to read symbols under "virtual" heads.
3. Rescan the tape to write new symbols and move heads
4. When M moves head onto unused blank square, S will try to move virtual head onto $\#$.

   S handles it by writing blank $\llcorner\lrcorner$ on tape, and shifting the rest of tape one square to the right.

Random Access Machine (RAM)

- CPU
- 3 Registers (Instruction Register (IR), Program Counter (PC), Accumulator (ACC))
- Memory
- Operation:
  - Increment PC
  - Set IR $\leftarrow$ MEM[PC]
  - Execute instruction in IR
  - Repeat
- Instructions are typically compare, add/subtract, multiply/divide, shift left/right.
- We assume no limit on the registers' size
- All instructions are doable on a TM.

RAM: schematic picture



RAM: instructions

	Instruction	Meaning
00	HALT	Stop Computation
01	LOAD x	ACC $\leftarrow$ MEM[x]
02	LOADI x	ACC $\leftarrow$ x
03	STORE x	MEM[x] $\leftarrow$ ACC
04	ADD x	ACC $\leftarrow$ ACC + MEM[x]
05	ADDI x	ACC $\leftarrow$ ACC + x
06	SUB x	ACC $\leftarrow$ ACC - MEM[x]
07	SUBI x	ACC $\leftarrow$ ACC - x
08	JUMP x	PC $\leftarrow$ x
09	JZERO x	PC $\leftarrow$ x if ACC = 0
10	JGT x	PC $\leftarrow$ x if ACC $>$ 0

RAM: example program

A program that multiplies two numbers (in locations 1000 & 1001), and stores the result in 1002

Memory	Machine Code	Assembly
0001	011000	LOAD 1000
0002	031003	STORE 1003
0003	020000	LOADI 0
0004	031002	STORE 1002
0005	021003	LOAD 1003
0006	090013	JZERO 0013
0007	070001	SUBI 1
0008	031003	STORE 1003
0009	011002	LOAD 1002
0010	041001	ADD 1001
0011	080004	JUMP 4
0013	000000	HALT

TM is (at least) as strong as RAM

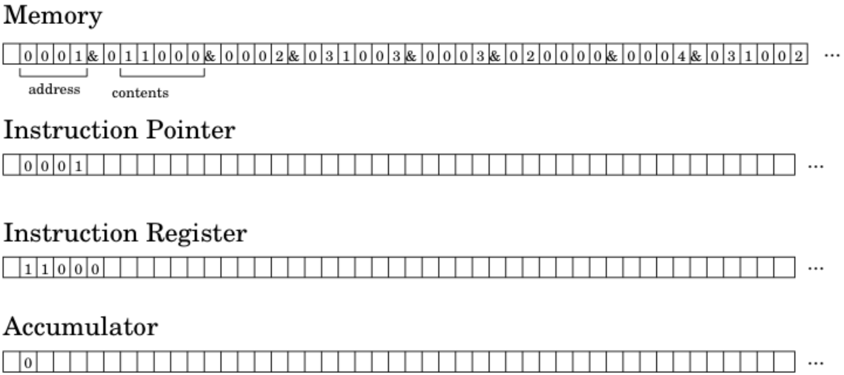
Theorem 17

A Turing machine can emulate this RAM model.

Proof's idea: We can emulate the RAM model with a multi-tape Turing machine:

- One tape for each register (IR, PC, ACC)
- One tape for the Memory
- Memory tape will be entries of the form <address> <contents> in increasing order of addresses.

Emulator's Tapes



The emulator

Algorithm 18 (Emulator (high-level description))

1. Scan through memory until reach an address that matches the PC
2. Copy contents of memory at that address to the IR
3. Increment PC
4. Based on the instruction code:
   - Copy a value into PC
   - Copy a value into Memory
   - Copy a value into the ACC
   - Perform an arithmetic operation, a shift, or a comparison

Part IV

Turing Completeness

Turing completeness

- A computation model is called Turing complete, if it can emulate a Turing Machine.
- Turing complete machine can compute anything a TM could
  - Of course it might not be convenient ...
- We just seen (the easy part of the proofs) that multitape TM, and RAM machines are Turing complete

Part V

Church-Turing Thesis

Beyond RAM

- A RAM can be modeled (emulated) by a Turing Machine.
- Any current machine (architecture, manufacturer, operating system, power supply, etc.) can be modeled by a Turing Machine.
- Note that at this point, we don't care how long it might take, just that it can be done.
- Hence, if there is an "algorithm" for it, a Turning Machine can do it.

What is an algorithm?

Informally:

1. A recipe

2. A procedure
3. A computer program
4. Who cares? I know it when I see it :-(
5. The notion of algorithm has long history in Mathematics (starting with Euclid's gcd algorithm), but not precisely defined until 20'th century

Informal notions rarely questioned, still they were insufficient

Computation models

- Many models have been proposed for general-purpose computation. Remarkably, all "reasonable" models were shown to be equivalent to Turing machines.
- The notion of an algorithm is model-independent!
- We don't really care about Turing machines *per se*.
- We do care about understanding computation, and because of their simplicity, Turing machines are a good model to use.

Models equivalent to TM

- All "reasonable" programming languages (e.g., Java, Pascal, C, Python, Scheme, Mathematica, Maple, Cobol,. . .).
- λ-calculus of Alonzo Church
- Turing machines of Alan Turing
- Recursive functions of Godel and Kleene
- Counter machines of Minsky
- Normal algorithms of Markov
- Unrestricted grammars
- Two stack automata
- Random access machines (RAMs)

$\vdots$

Church-Turing Thesis

"The intuitive notion of reasonable models of computation equals Turing machine algorithms".

“Wild" Models of Computation

Consider MUntel's $\aleph$-AXP10© processor (to be released ...)

Definition 19 ($\aleph$-AXP10©)

Like a Turing machine, except

- Takes first step in 1 second.
- Takes second step in $1/2$ second.
- Takes i-th step in 2^{-i} seconds ...

After 2 seconds, the $\aleph$-AXP© decides any enumerable language!

Question 20

Does the $\aleph$-AXP© invalidate the Church-Turing Thesis?