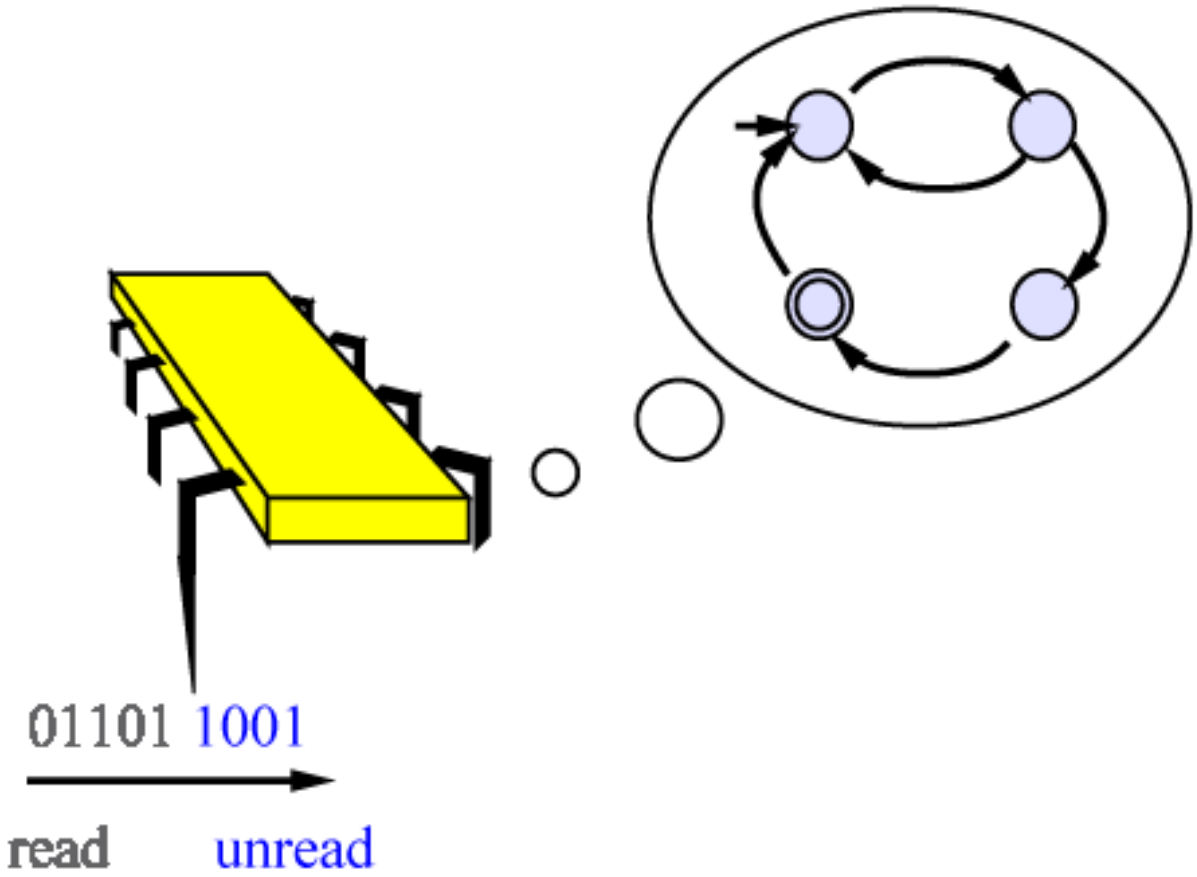
Comp. Models - Lecture 6, Spring 2011

- Turing Machines
- Turing Machines
- Turing Machines

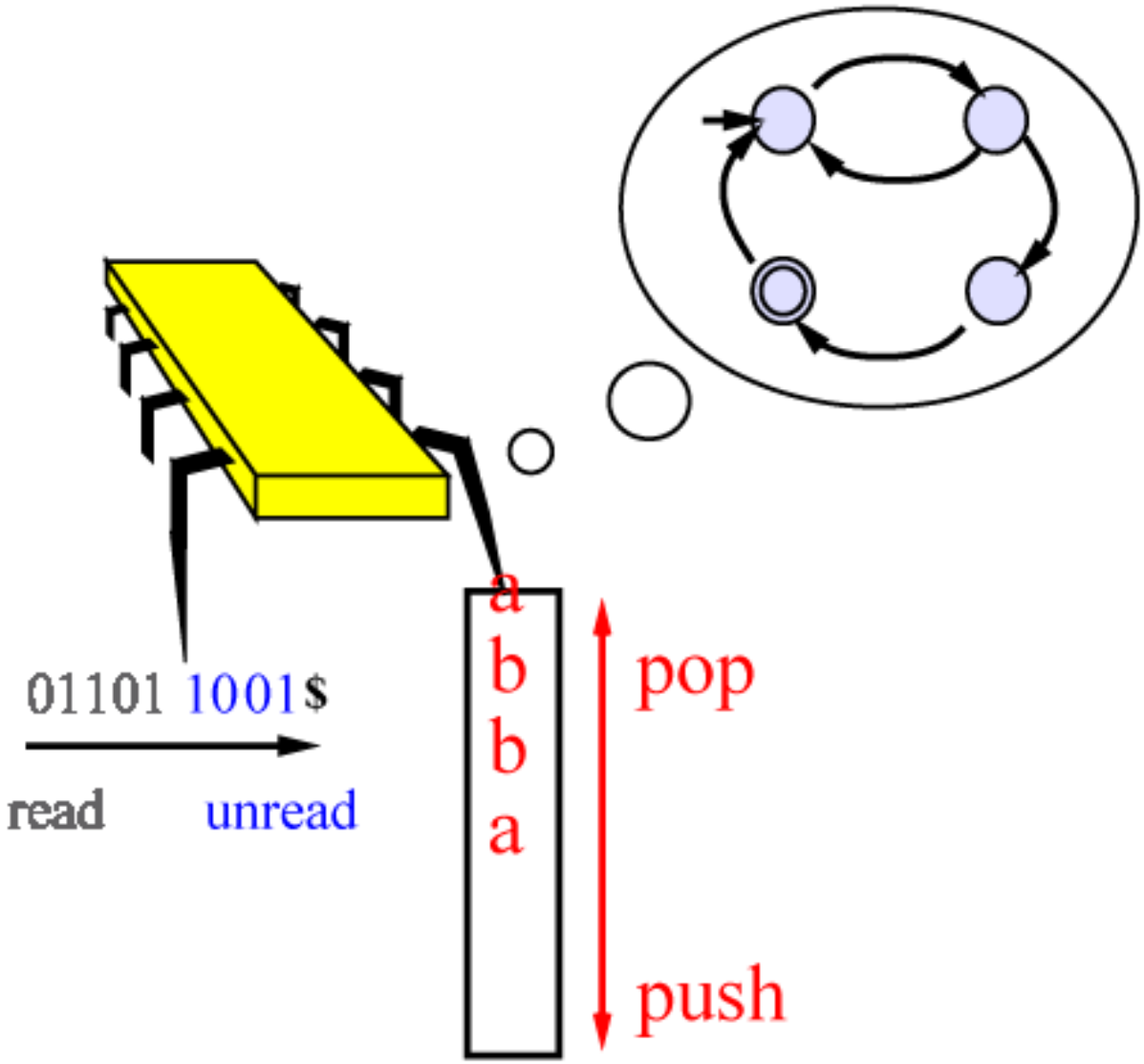
- Alternative Models of Computers
- Multitape TMs, RAMs, Non Deterministic TMs
- The language classes $\mathcal{R} = \mathcal{RE} \cap co\mathcal{RE}$

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

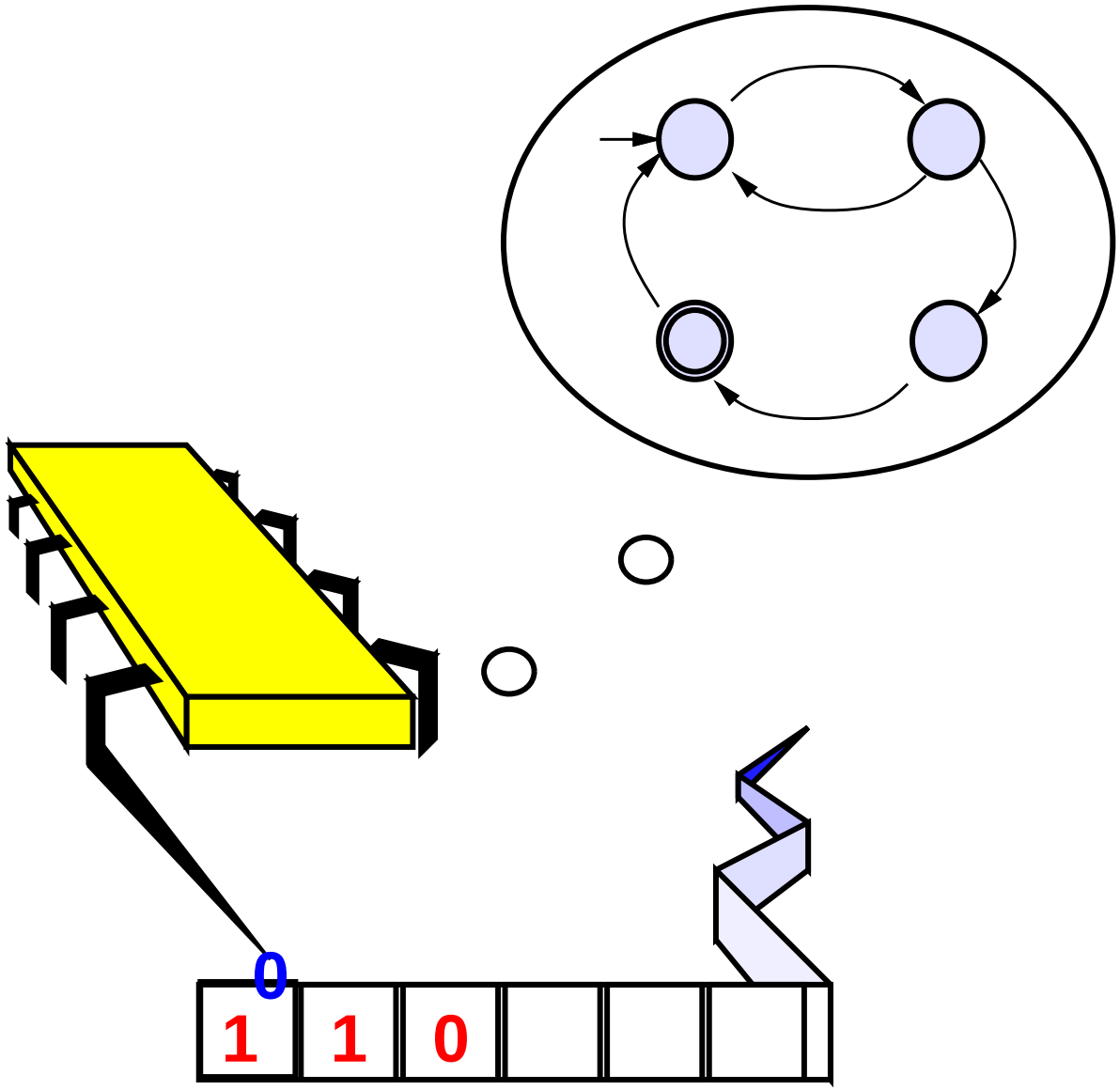
A Finite Automaton



A Pushdown Automaton



A Turing Machine

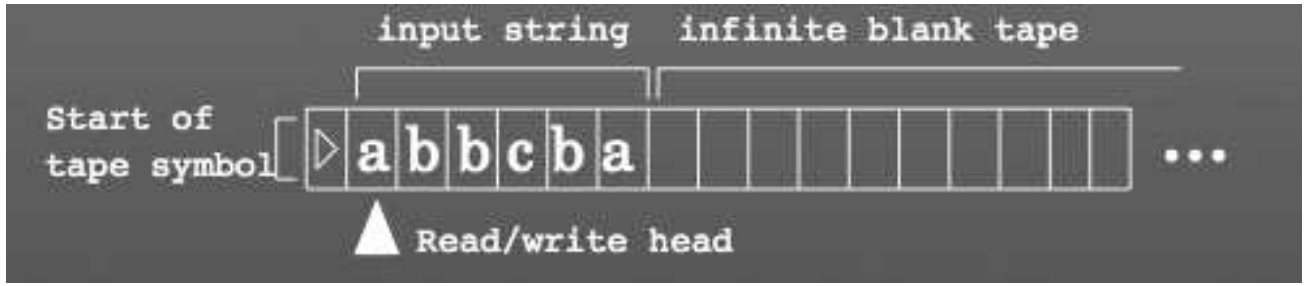


Turing Machines

- Machines so far (DFA, PDA) read input only once
- Turing Machines
  - Can back up over the input
  - Can overwrite the input
  - Can write information on tape and come back to it later

Turing Machines

- Input string is written on a tape:



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

TM vs. PDA vs. DFA: Differences

- 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 One Step

One step of computation changes

- current state,
- current head position,
- and tape contents at current position.

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

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

- Turing Machine algorithm:
  - Check for single $\#$,
    - If false then reject.
  - Zig-zag across the tape, check identical letters, and replace them by X.
    - If not identical then reject.
  - When all the letters left of $\#$ are marked X, check for remaining letters right of $\#$
    - If there remaining letters are reject,
    - otherwise accept

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

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

Formal Definition

We start with the transition function

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\} \ .$$

$\delta(q, a) = (r, b, L)$ means:

- in state q where head reads tape symbol a,
- the machine writes b, replacing the a ($a = b$ is possible),
- enters state r,
- and moves the head left
  (this is what the L stands for).

Formal Definition (2)

Now the transition function, with a move to the right

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\} \ .$$

$\delta(q, a) = (r, b, R)$ means:

- in state q where head reads tape symbol a,
- the machine writes b, replacing the a ($a = b$ is possible),
- enters state r,
- and moves the head right
  (this is what the R stands for).

Formal Definition (3)

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 $\Sigma \subset \Gamma$.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
- $q_0 \in Q$ is the start state,
- $q_a \in Q$ is the accept state, and
- $q_r \in Q$ is the reject state.

Formal Definition (4)

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ computes as follows

- an input of length n, $w = w_1 w_2 \ldots w_n \in \Sigma^*$
- is placed on n leftmost tape squares, one tape square per input letter
- rest of tape contains blanks $\sqcup$
- read/write head is on leftmost square of tape
- since $\sqcup \notin \Sigma$, leftmost blank indicates end of input.

Formal Definition (5)

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$.

When computation starts,

- M proceeds according to transition function δ.
- If M tries to move head beyond left-hand-end of tape, it doesn't move (still M does not crash).
- Computation continues until q_a or q_r is reached,
- otherwise M runs forever.

TM 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 $1011q_70111$ 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,
- and head is on 0 (leftmost entry of right hand side).

Configurations: The Yield Relation

- If $\delta(q_i, b) = (q_j, c, L)$ then configuration uaq_ibv yields configuration uq_jacv.
- If $\delta(q_i, b) = (q_j, c, R)$, then configuration uaq_ibv yields configuration $uacq_jv$.
- Special case (1): When head is at left end and tries to move left, it changes state and writes on tape but does not move, so if $\delta(q_i, b) = (q_j, c, L)$, configuration q_ibv yields q_jcv.
- Special case (2): What happens when head is at right end? We let wq_i and $wq_i\sqcup$ denote the same configuration, so moves to the right can now be accomodated.

Configurations: Special Case 2

- Special case (2): What happens when head is at right end? We let wq_i and $wq_i\sqcup$ denote the same configuration, so moves to the right can now be accomodated.
- In special case (2), the new configuration is longer as it "annexed" one blank. This allows configurations to grow in length with computation. Yet at any given moment, they are finitely long.

More on Configurations

We have

- 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 $w_0 q_r w_1$

Accepting a Language

A Turing machine M accepts an input w if there is a sequence of configurations $C_1, C_2, \ldots, C_k$ such that

- C_1 is start configuration of M on w,
- each C_i yields C_{i+1},
- C_k is an accepting configuration.

The collection of strings accepted by M is called the language of M, and is denoted $L(M)$.

Enumerable Languages

Definition: A language L is called (recursively) enumerable (RE) if some Turing machine accepts L.
(In the book called Turing-recognizable)

Enumerable Languages (2)

On an input, w, a TM may

- accept
- reject
- loop (run forever)

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

Decidable Languages

Definition: A TM decides a language if for every input $w \in \Sigma^*$, the TM halts.

Namely the TM either reaches state q_a (in case $w \in L(M)$) or it reaches state q_r (in case $w \notin L(M)$), but it does not loop.

Such TM is called a decider.

Definition: A language L is decidable if some Turing machine decides it.
(In the book called Turing-decidable also called recursive)

Example: check if $x = 2^n$

- Function POWER2(x)
- IF $x = 1$ return TRUE
- ELSE IF $x \bmod 2 = 1$ return FALSE
- ELSE POWER2($x/2$)

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

On input w:

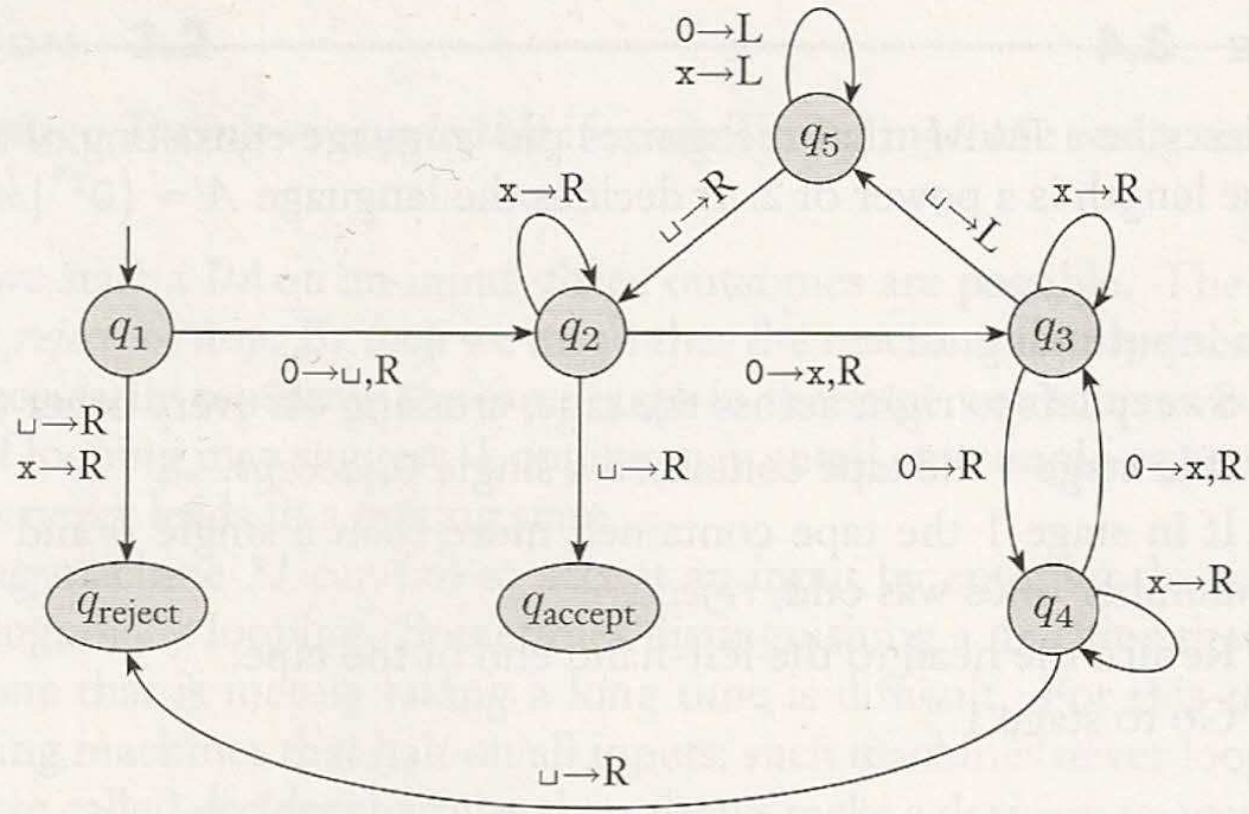
- Stage I: move to the right, erasing every other 0
- If in stage I contains a single 0, then accept.
- If after stage I the tape contains more than a single 0, but an odd number of 0, then reject.
- Return to the start of the tape
- Go to stage I.

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

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

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

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

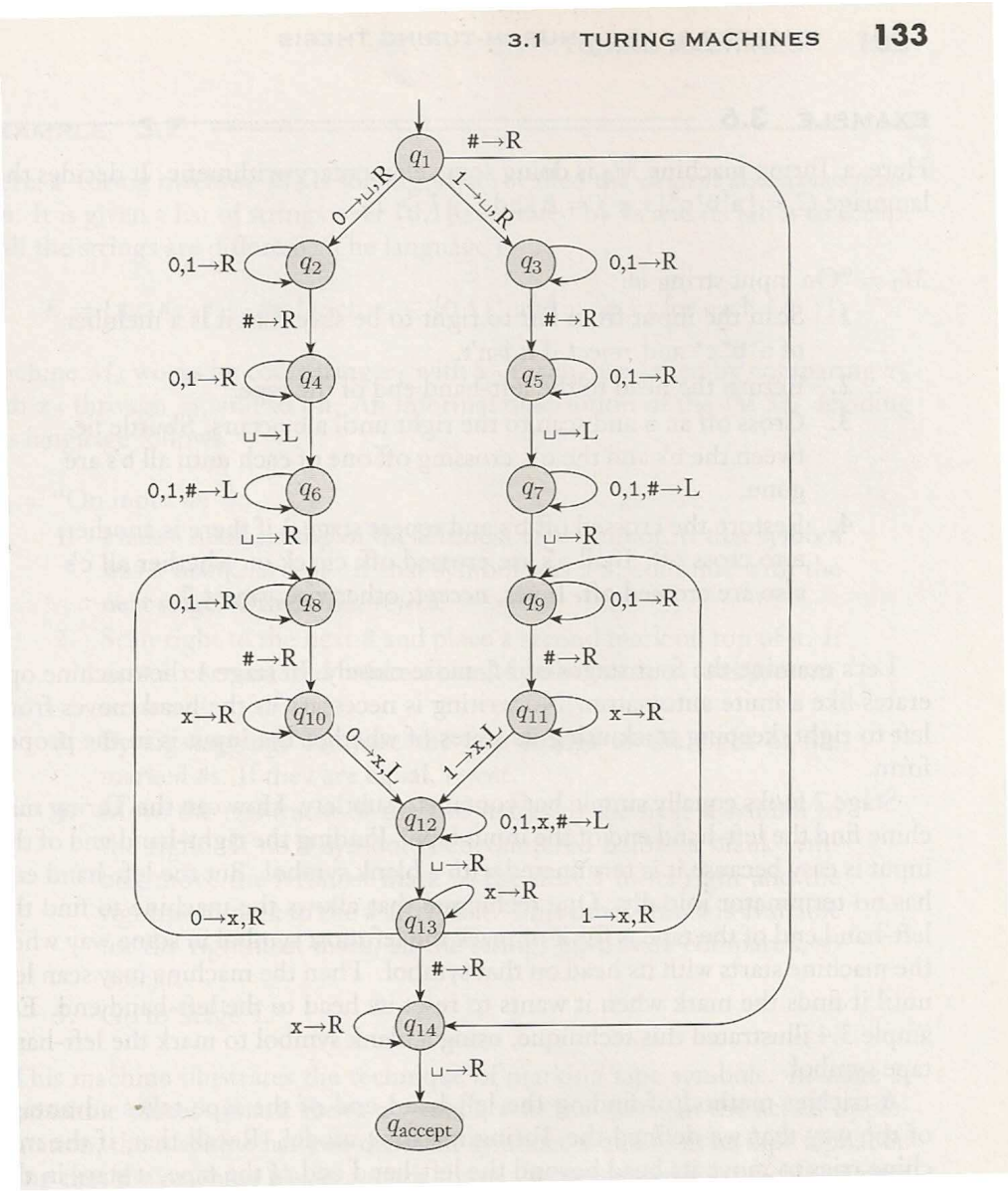


Example: $B = \{w\#w \mid w \in \{0,1\}^*\}$

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

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

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



Example C

Test $i \times j = k$ where $i, j, k \geq 1$

- FUNCTION MULTIPLY(i, j, k)
- $tmp = k$
- WHILE $i \leq 0$
- $tmp = tmp - j$
- $i = i - 1$
- END WHILE
- IF $tmp = 0$ return TRUE
- ELSE return FALSE

Example C

Here is a high level description of a TM that decides the (non context free) language

$$C = \{a^i b^j c^k \mid i \times j = k \text{ where } i, j, k \geq 1\}$$

- scan from left to right to check that input is $a^* b^* c^*$
- return to start of tape
- 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.
- 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.

A Minor, Technical Point

Question: To implement algorithm, should be able to tell when a TM is at the left end of the tape.

Answer Mark it with a special symbol. When head reads this symbol, TM “knows” it is on the leftmost tape square.

Example E

Consider the element distinctness problem

$E = \{\#x_1\#x_2\#\ldots\#x_\ell \mid$
each $x_i \in \{0,1\}^*$ and for each $i \neq j, x_i \neq x_j\}$.

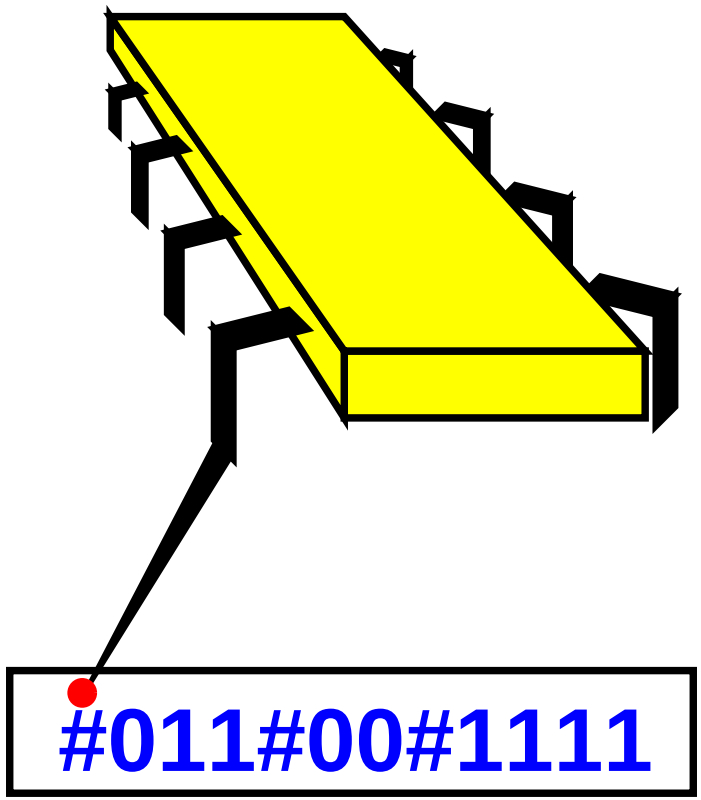
Verbally,

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

Element Distinctness – Solution

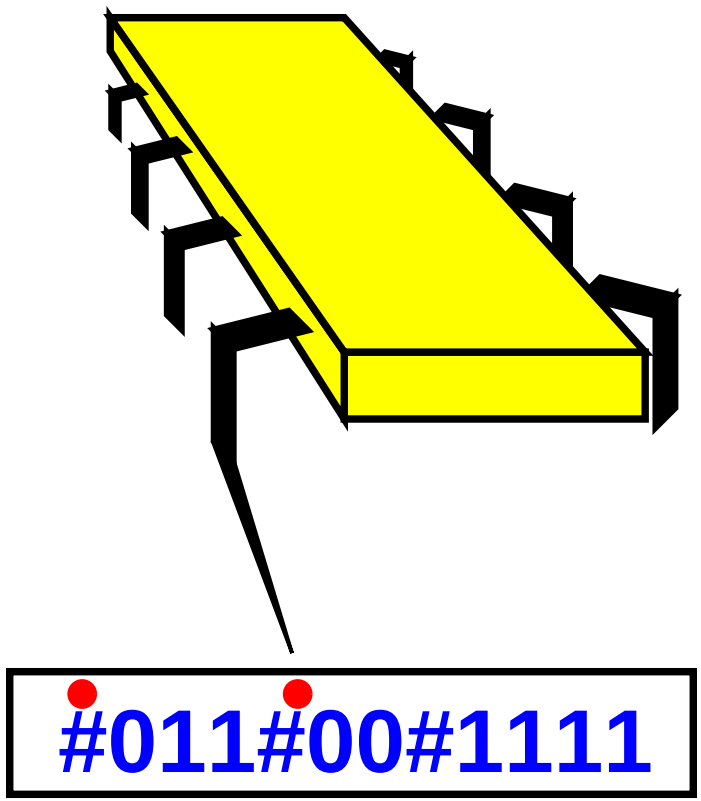
On input w

- place a mark on leftmost tape symbol. If symbol not $\#$, reject.



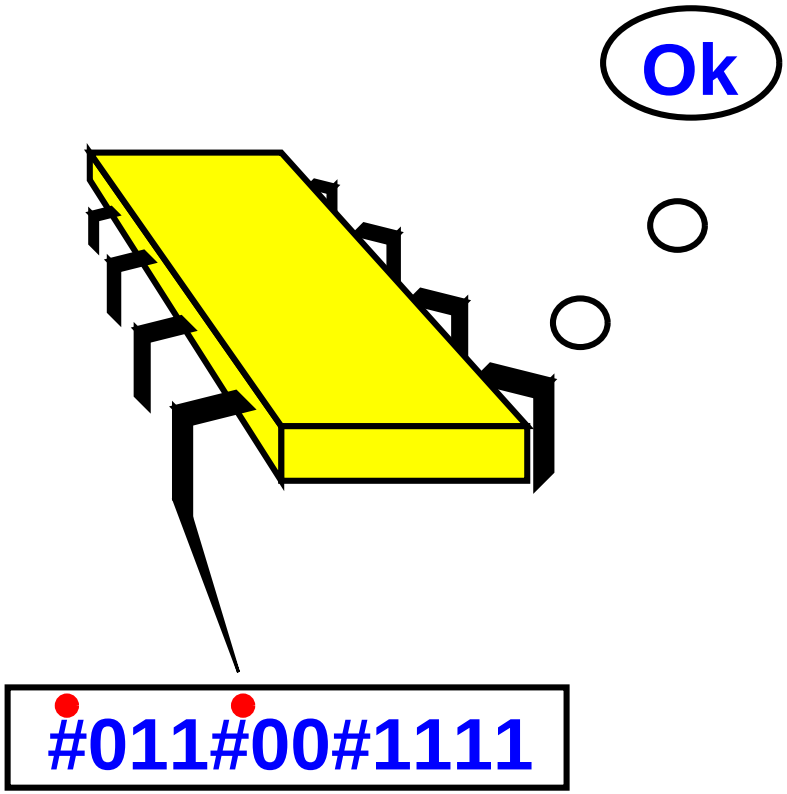
Element Distinctness – Solution (2)

- scan right to next # and place mark on top. If none encountered, reject



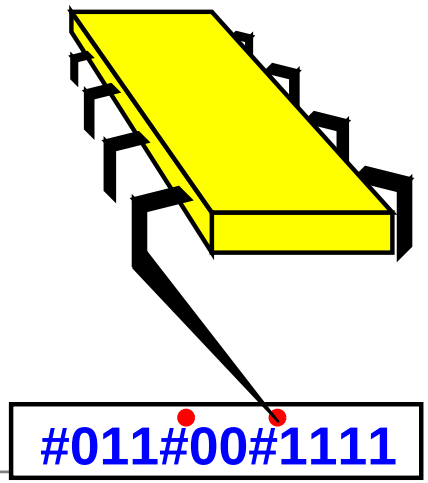
Element Distinctness – Solution (3)

- By zig-zagging, compare the two strings to the right of the marked #'s. If equal, reject.



Element Distinctness – Solution (4)

- Iterate:
- Move rightmost mark to next # on right, if any.
- Compare two strings to the right of the marked #'s, as before.
- Otherwise move leftmost mark to next # on right and rightmost mark to # after that.
- If not possible, accept.



Element Distinctness – Solution (end)

Question: How do we “mark” a symbol?

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

TMs Variants

Alternative Turing machine definitions abound.

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

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

Question: Does this add any power?

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

Important notion here: Two-way simulation (model A capable of simulating model B; model B capable of simulating model A).

Multitape Turing Machines

- each tape has its own head
- initially, input string on tape 1 and rest blank

For the transition function: $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$,
the expression $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$
means

- machine starts in state q_i
- if heads 1 through k reading $a_1, \ldots, a_k$,
- then machine goes to state q_j,
- heads 1 thought k write $b_1, \ldots, b_k$,
- and moves each head right or left as specified.

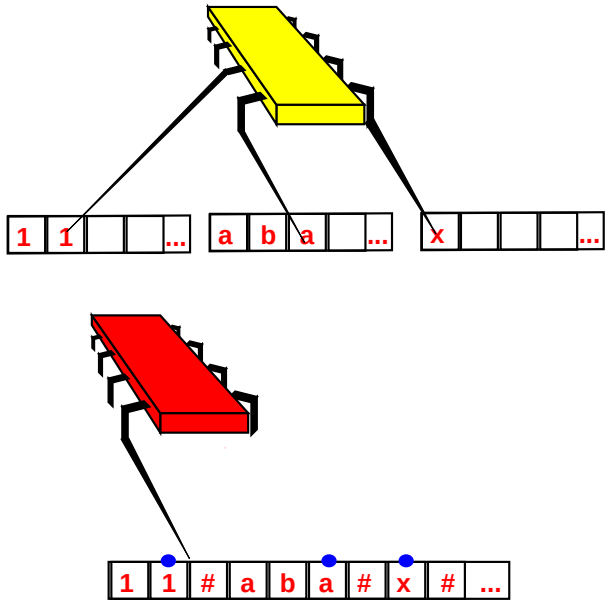
Equivalence

Theorem: A language is enumerable if and only if there is some multitape Turing machine that accepts it.

One direction is trivial.

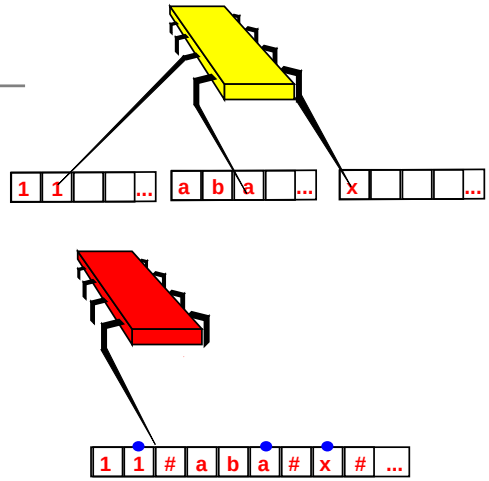
To prove the other direction, we will show how to convert a multitape TM, M, into an equivalent single-tape TM, S.

Simulation Idea



- S simulates 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.

Simulation (2)



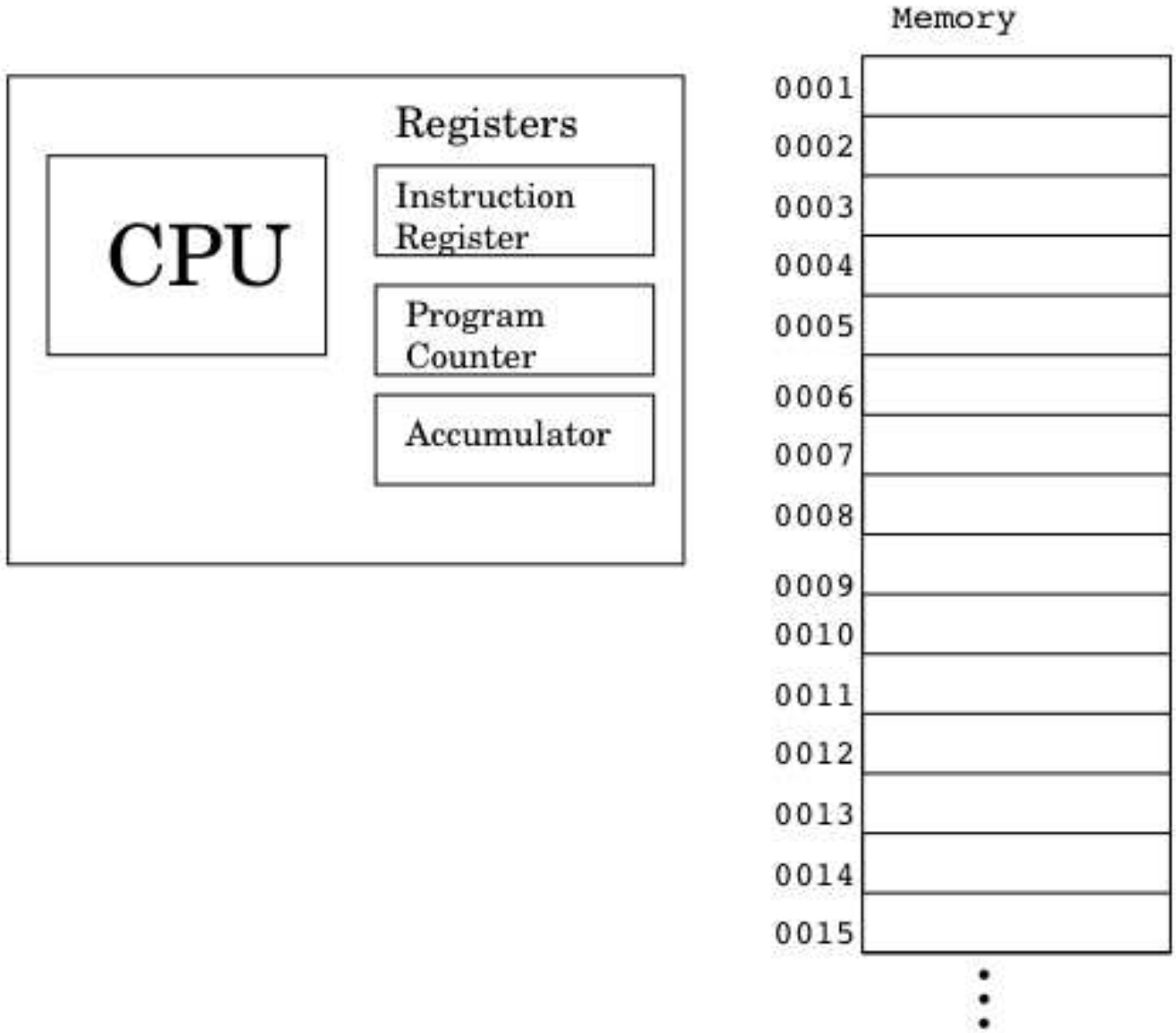
On input $w = w_1 \cdots w_n$, S:

- writes on its tape $\# \overset{\bullet}{w_1}\ w_2 \cdot w_n \# \overset{\bullet}{\sqcup} \# \overset{\bullet}{\sqcup} \# \cdots \#$
- scans its tape from first $\#$ to $k+1$-st $\#$ to read symbols under "virtual" heads.
- rescans to write new symbols and move heads
- S tries to move virtual head onto $\#$ when M is trying to move head onto unused blank square. S writes blank $\sqcup$ on tape, and shifts rest of tape one square to the right. ♠

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.
- All instructions are doable on a TM.

RAM


Memory
CPU
Registers
Instruction Register
Program Counter
Accumulator
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
0011
0012
0013
0014
0015

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

Here is 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

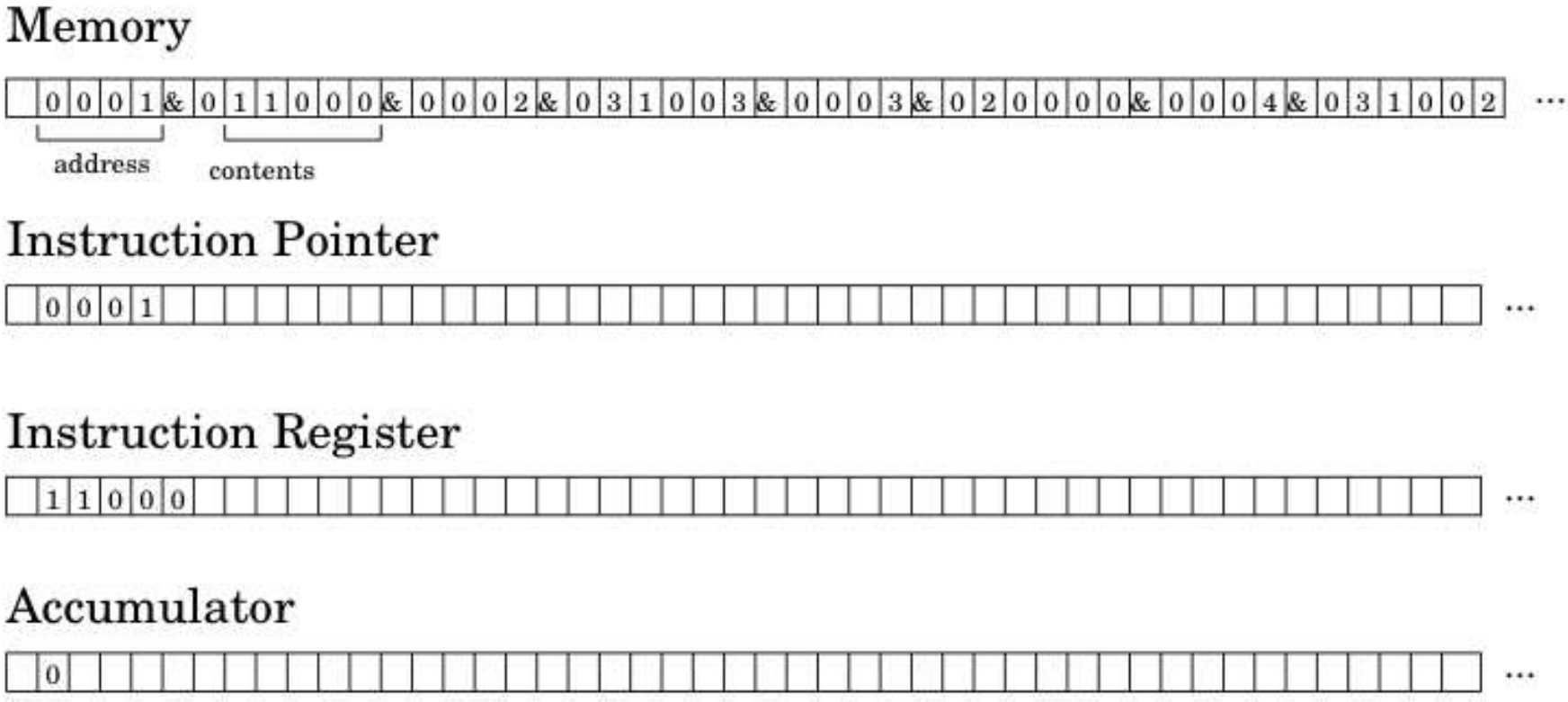
Computers & TMs

Theorem: A multi-tape Turing machine can simulate this RAM model.

Idea: We can simulate 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.

Computers & TMs



Computers & TMs

- Simulating one RAM step on TM:
  - Scan through memory until reach an address that matches the PC
  - Copy contents of memory at that address to the IR
  - Increment PC
  - 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

RAMs, Computers & TMs

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

Turing Completeness

- A computation model is called “Turing Complete” if it can simulate a (general) Turing Machine.
- Turing Complete $\Rightarrow$ can compute anything a TM could
  - Of course it might not be convenient ...
- Church-Turing Thesis: any computational model can be simulated by a Turing machine.
- More next lecture

Non-Deterministic Turing Machines

Transition function:

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

- Computation of a deterministic TM can be viewed as a path in configuration space.
- Computation of a non-deterministic TM (NTM) can be viewed as a tree in configuration space.

- NTM accepts an input if there is ($\exists$) an accepting branch in its computation tree.
- NTM rejects an input if all ($\forall$) branches in its computation tree are either rejecting or infinite (looping).

Equivalence

Theorem: A language is enumerable ($\mathcal{RE}$) if and only if there is some non-deterministic Turing machine that accepts it.

One direction is trivial.

To prove the other direction, we will show how to convert a non-deterministic TM, N, into an equivalent deterministic TM, D.

Simulating Non-Determinism

Basic idea:

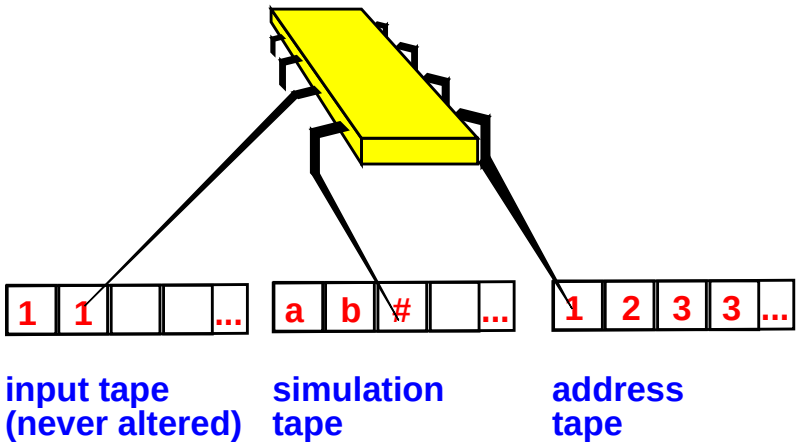
- D tries all possible branches
- If D finds any accepting branch, it accepts.
- If all branches reject, D rejects.
- If all branches reject or loop, D loops.

Simulating Non-Determinism (2)

N's computation is a tree in the configurations' space.

- each tree branch is a branch of N's non-deterministic computation
- each tree node is a configuration of N
- root is starting configuration
- the number of children of each node, denoted by b, is at most the number of N's states, times the size of Γ, times 2 (left/right).
- depth-first search doesn't work (why?)
- breadth-first search does work, as we'll show.

Simulating Non-Determinism (3)



The simulating machine, D, has three tapes

- the input tape is never altered (only read from)
- the simulation tape is a copy of N's tape
- the address tape keeps track of D's location in N's computation tree.

Simulating Non-Determinism (4)

The address tape:

- every node in the tree has at most b children
- every node in the tree is assigned an address that is a string over the alphabet $\Sigma_b = \{1, 2, \ldots, b\}$
- to get to node with address 231
  - start at root
  - take second child of root
  - take third child of current node
  - take first child of current node
- ignore meaningless addresses (choices not available for configuration along branch)

Simulating Non-Determinism (5)

1) Initially, the input tape contains w, and the other two tapes are empty.
2) Copy input tape to simulation tape.
3) Use simulation tape to simulate N on input w on a finite portion of one non-deterministic branch. On each choice, consult the next symbol on address tape. Accept if accepting configuration reached. Skip to next step if
   - symbols on address tape are exhausted
   - non-deterministic choice is invalid
   - rejecting configuration was reached
4) Replace string on address tape with the lexicographically next string. Go back to Step 2, to simulate this branch of N's computation.

Decidability vs. Enumerability

- Decidability is a stronger notion than enumerability.
- If a language L is decidable then clearly it is enumerable (the other dirction does not hold, as we'll show in a couple of lectures).
- It is also clear that if L is decidable then so is $\overline{L}$, and thus $\overline{L}$ is also enumerable.
- Let $\mathcal{RE}$ denote the class of enumerable languages, and let $co\mathcal{RE}$ denote the class of languages whose complement is enumerable.
- Let $\mathcal{R}$ denote the class of decidable languages. Then what we just argued implies $\mathcal{R} \subseteq \mathcal{RE} \cap co\mathcal{RE}$.

Decidability vs. Enumerability (2)

Theorem: $\mathcal{R}=\mathcal{RE} \cap co\mathcal{RE}$.

Proof: We should prove the $\supseteq$ direction. Namely if $L \in \mathcal{RE} \cap co\mathcal{RE}$, then $L \in \mathcal{R}$.

In other words, if both L and its complement are enumerable, then L is decidable.

Let M_1 be a TM that accepts L.
Let M_2 be a TM that accepts $\overline{L}$.
We describe a TM, M, that decides L.

On input x, M runs M_1 and M_2 in parallel.
If M_1 accepts, M accepts.
If M_2 accepts, M rejects.

Should now show that indeed M decides L.

In Parallel?

Question: What does it mean to run $M_1, \ldots, M_k$ in parallel?

- M has two tapes
- One tape has the current configuration of each M_i
- The second tape is a simulation tape.
- M, does for ever,
- for $i = 1, \ldots, k$.
  - copies the configuration of M_i,
  - simulates a step of M_i,
  - writes the new configuration of M_i back.
  - If this is accepting configuration, HALT and return i.

Claim

We claim that M decides L.

- Every string is in L or in $\overline{L}$ (of course not in both).
- Thus either M_1 or M_2 accepts the input w.
- Consequently, since M halts whenever M_1 or M_2 accepts, M always halts, and hence is a decider.
- Moreover, M accepts strings in L and rejects strings in $\overline{L}$.

Therefore, M decides L, so L is decidable. ♣

Revised View of the World of Languages