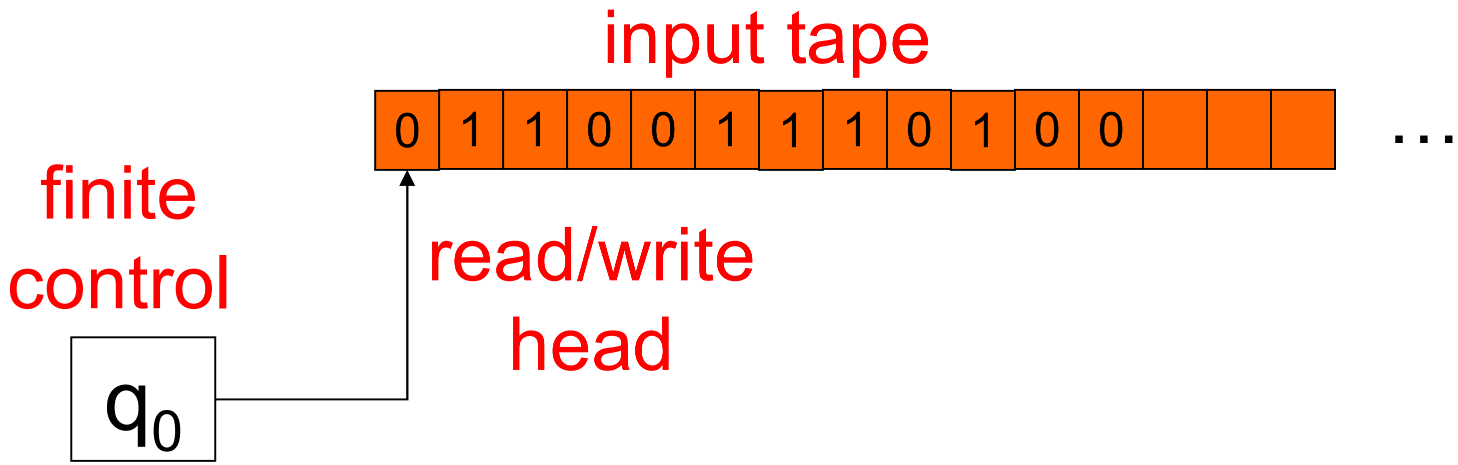
Introduction to the Theory of Computation

Part II: Computability Theory

3. The Church-Turing Thesis

- 3.1 Turing Machines
- 3.2 Variants of Turing Machines
  - Multitape Turing Machines
  - Nondeterministic Turing Machines
  - Enumerators
  - Equivalence with other models
- 3.3 The Definition of Algorithm

Turing Machines

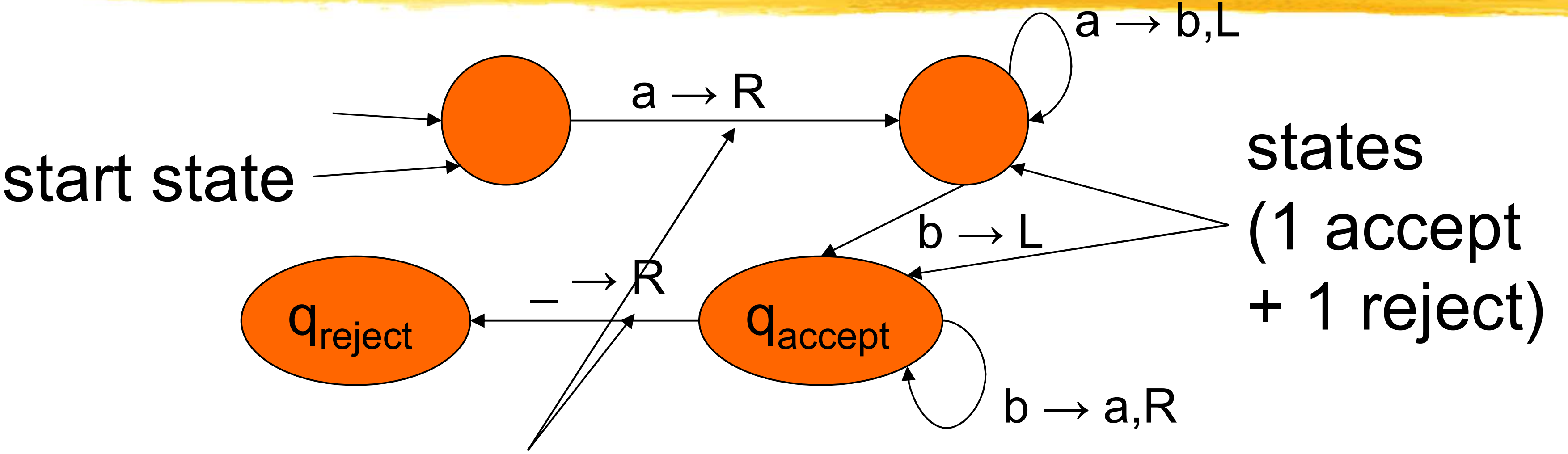


- New capabilities:
  - infinite tape
  - can read OR write to tape
  - read/write head can move left and right

Turing Machines

- Informal description:
  - input written on left-most squares of tape
  - rest of squares are blank
  - at each point, take a step determined by
    - current symbol being read
    - current state of finite control
  - a step consists of
    - writing new symbol
    - moving read/write head left or right
    - changing state

Turing Machine Diagrams



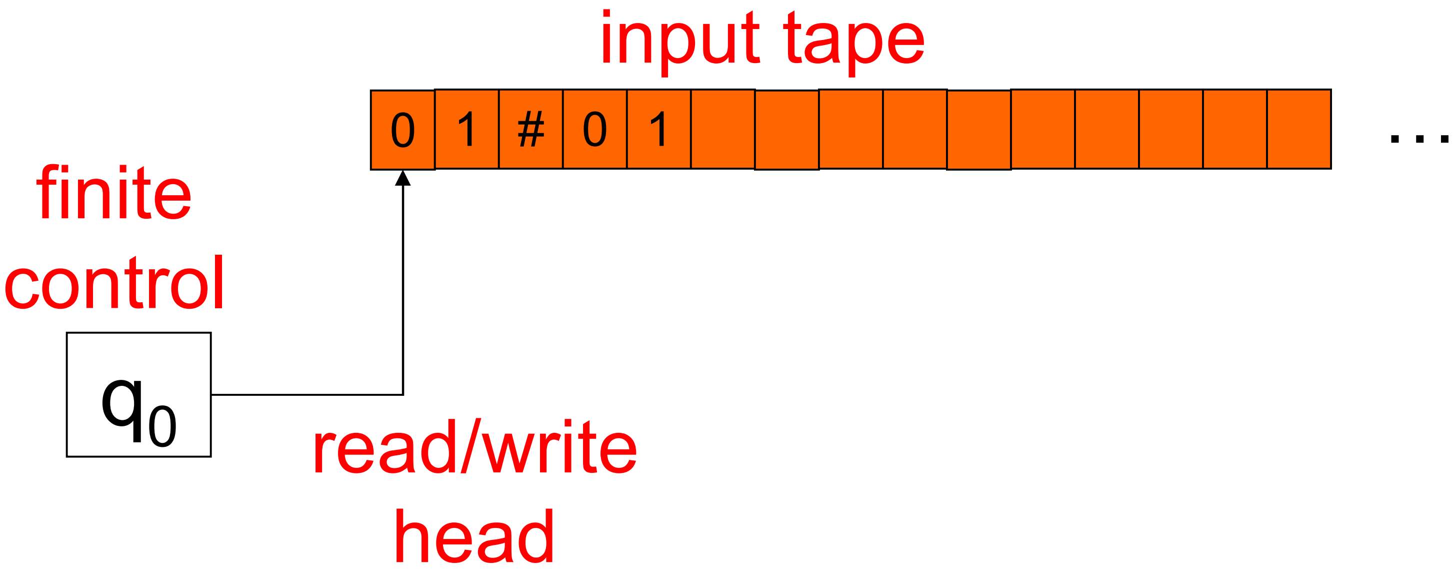
transition label: (tape symbol read → tape symbol written, direction moved)

- a → R means "read a, move right"
- a → L means "read a, move left"
- a → b, R means "read a, write b, move right

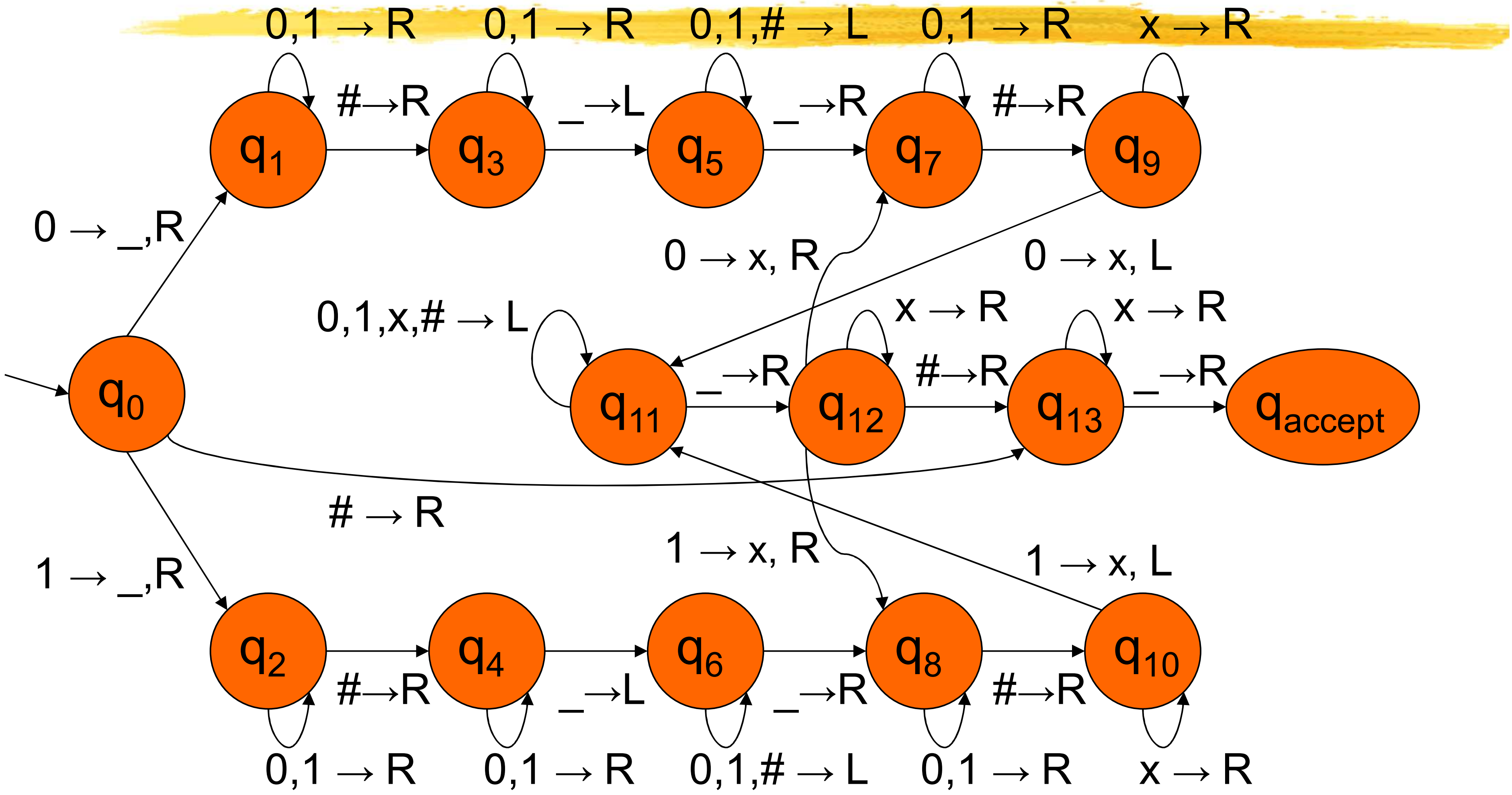
"_" means blank tape square

Example Turing Machine

language $L = \{w\#w : w \in \{0,1\}^*\}$



Example TM Diagram



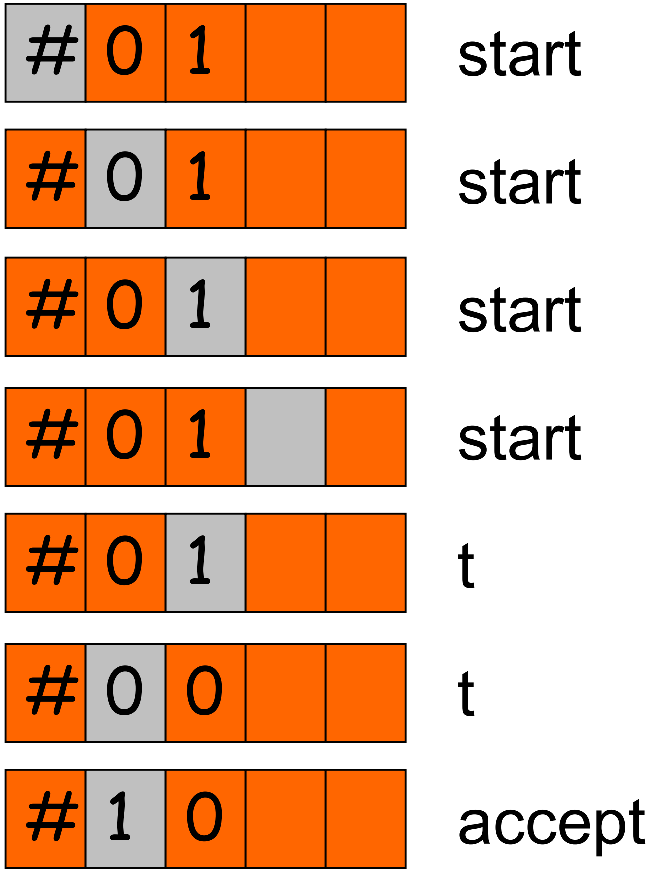
TM Formal Definition

- A TM is a 7-tuple

  $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where:

  - Q is a finite set called the states
  - Σ is a finite set called the input alphabet
  - Γ is a finite set called the tape alphabet, $\Gamma \supseteq \Sigma \cup \{`\_'\}$
  - δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is a function called the transition function
  - q_0 is an element of Q called the start state
  - q_{accept}, q_{reject} are the accept and reject states

Example TM Operation



program
for “binary
successor”

q	σ	δ(q,σ)
start	0	(start, 0, R)
start	1	(start, 1, R)
start	_	(t, _, L)
start	#	(start, #, R)
t	0	(accept, 1, -)
t	1	(t, 0, L)
t	#	(accept, #, R)

TM Configurations

- At every step in a computation, a TM is in a configuration determined by:
  - the current tape contents
  - the current state
  - the current head location
- next step completely determined by current configuration
- shorthand: string uqv with $u,v \in \Gamma^*$, $q \in Q$

TM Configurations

- configuration C_1 yields configuration C_2 if TM can legally move from C_1 to C_2 in 1 step
  - notation: $C_1 \Rightarrow C_2$
  - also: "yields in 1 step" notation: $C_1 \Rightarrow^1 C_2$
  - "yields in k steps" notation: $C_1 \Rightarrow^k C_2$

  if there exists configurations $D_1, D_2, \ldots, D_{k-1}$ for which $C_1 \Rightarrow D_1 \Rightarrow D_2 \Rightarrow \ldots \Rightarrow D_{k-1} \Rightarrow C_2$
  - also: "yields in some # of steps" ($C_1 \Rightarrow^* C_2$)

TM Configurations

- Formal definition of "yields":

$$uaq_ibv \Rightarrow uq_jacv$$

if $\delta(q_i, b) = (q_j, c, L)$, and

$$uaq_ibv \Rightarrow uacq_jv$$

if $\delta(q_i, b) = (q_j, c, R)$

- two special cases:
  - left end: $q_ibv \Rightarrow q_jcv$ if $\delta(q_i, b) = (q_j, c, L)$
  - right end: uaq_i same as $uaq_i\_$

$u,v \in \Gamma^*$
$a,b,c \in \Gamma$
$q_i, q_j \in Q$

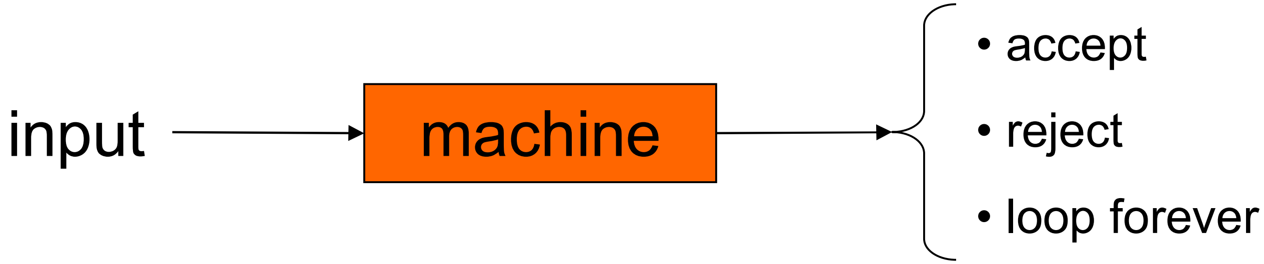
TM Acceptance

- start configuration: q_0w (w is input)
- accepting config.: any config. with state q_{accept}
- rejecting config.: any config. with state q_{reject}
- accepting config. and rejecting config. are halting config.

TM M accepts input w if there exist configurations $C_1, C_2, \ldots, C_k$

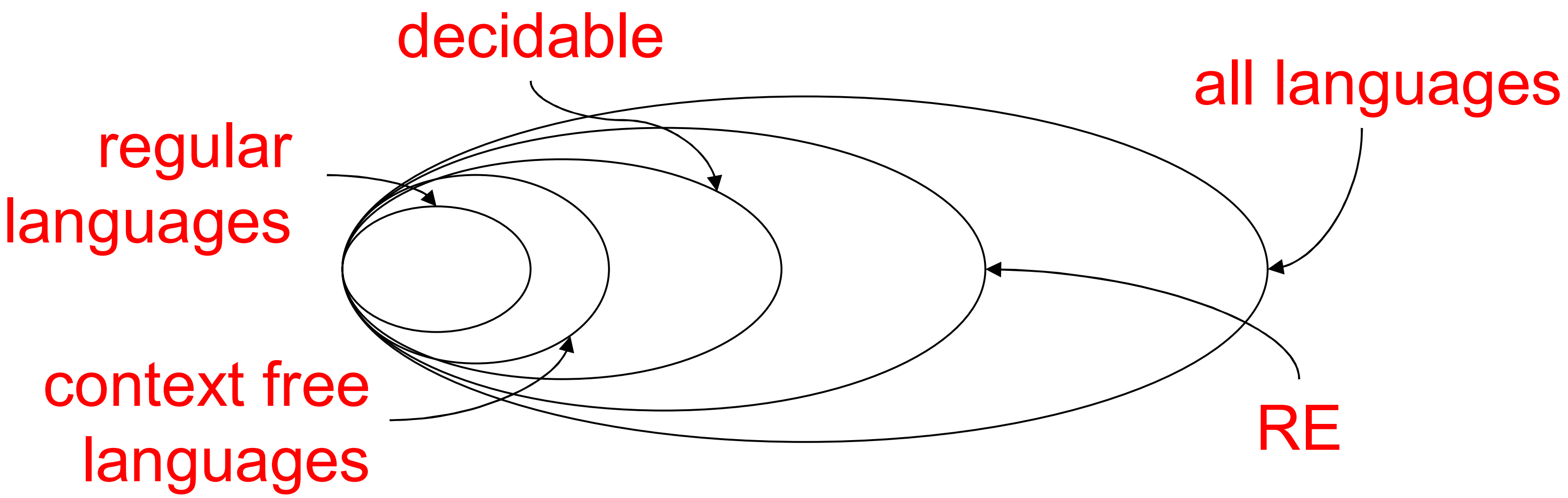
- C_1 is start configuration of M on input w
- $C_i \Rightarrow C_{i+1}$ for $i = 1, 2, 3, \ldots, k-1$
- C_k is an accepting configuration

Deciding and Recognizing



- TM M:
  - L(M) is the language it recognizes
  - if M rejects every x $\notin$ L(M) it decides L
  - set of languages recognized by some TM is called Turing-recognizable or recursively enumerable (RE)
  - set of languages decided by some TM is called Turing-decidable or decidable or recursive

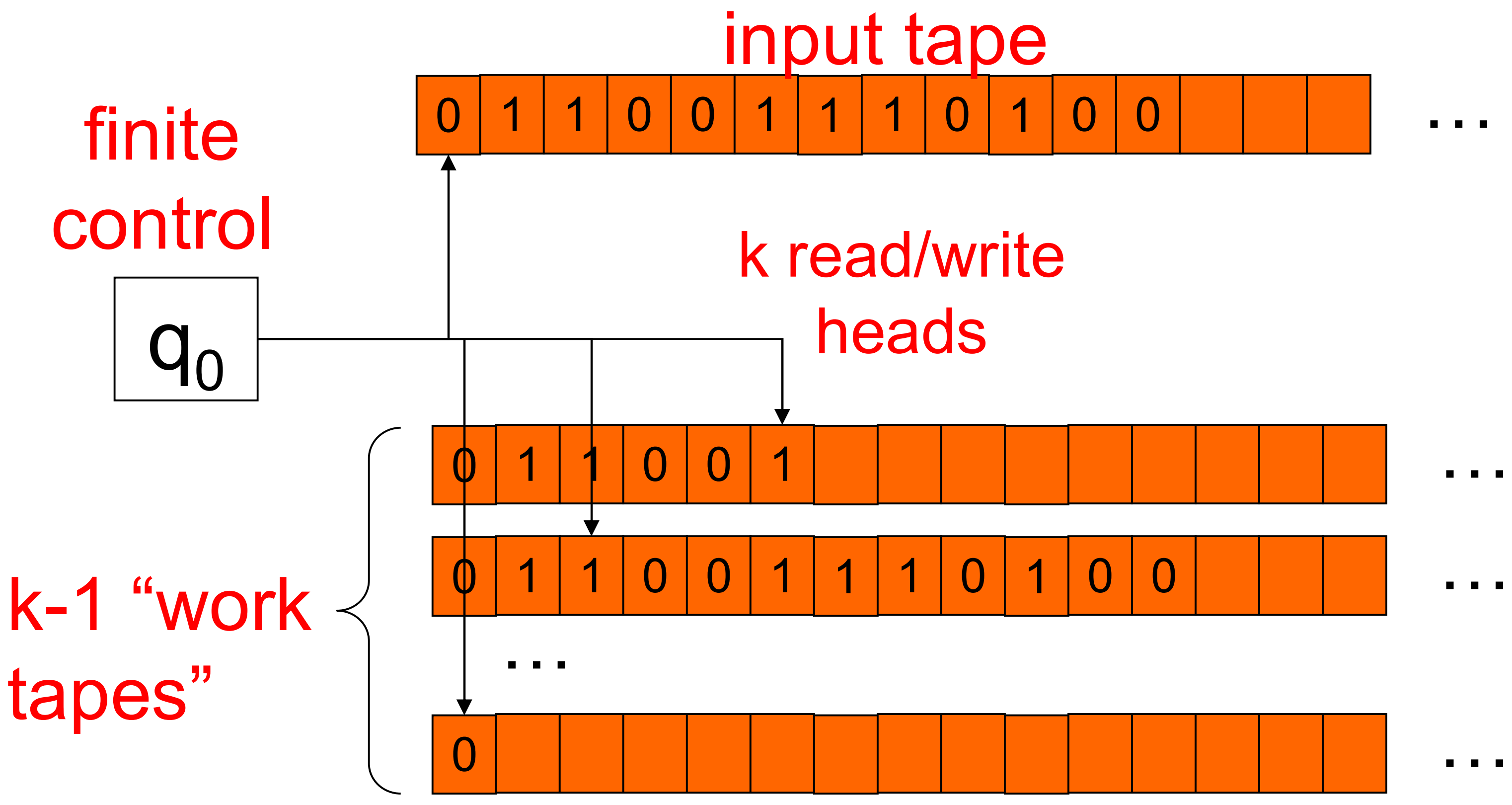
Classes of Languages



- We know: regular $\subset$ CFL (proper containment)
- CFL $\subset$ decidable?
- decidable $\subset$ RE $\subset$ all languages?

Multitape TMs

- A useful variant: k-tape TM



Multitape TMs

- Informal description of k-tape TM:
  - input written on left-most squares of tape #1
  - rest of squares are blank on all tapes
  - at each point, take a step determined by
    - current k symbols being read on k tapes
    - current state of finite control
  - a step consists of
    - writing k new symbols on k tapes
    - moving each of k read/write heads left or right
    - changing state

Multitape TM formal definition

- A TM is a 7-tuple

  $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where:

  - everything is the same as a TM except the transition function:

$$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$$

$\delta(q_i, a_1,a_2,\ldots,a_k) = (q_j, b_1,b_2,\ldots,b_k, L, R,\ldots, L) =$

"in state q_i, reading $a_1,a_2,\ldots,a_k$ on k tapes, move to state q_j, write $b_1,b_2,\ldots,b_k$ on k tapes, move L, R on k tapes as specified."

Multitape TMs

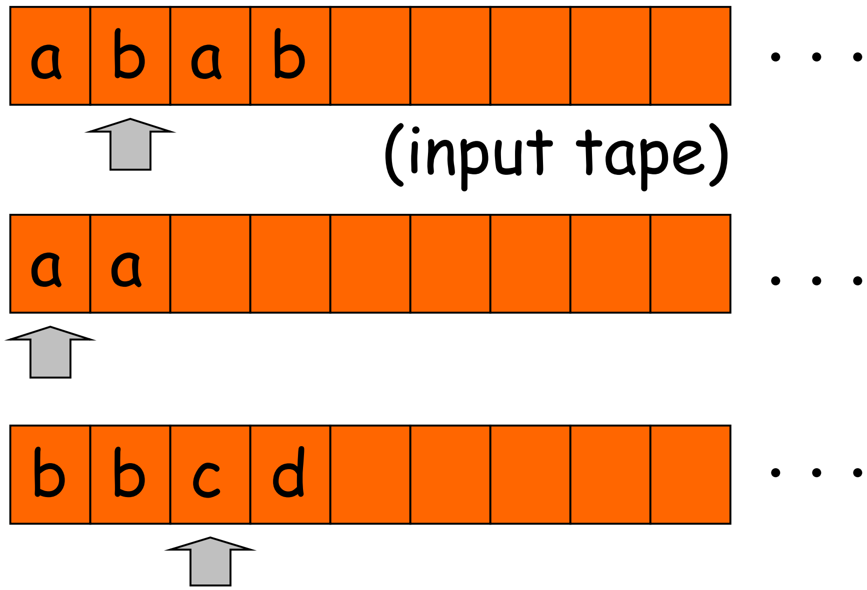
Theorem: every k-tape TM has an equivalent single-tape TM.

Proof:

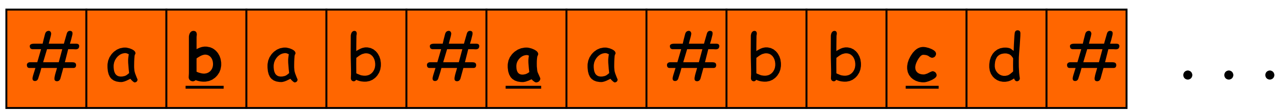
- Idea: simulate k-tape TM on a 1-tape TM.

Multitape TMs

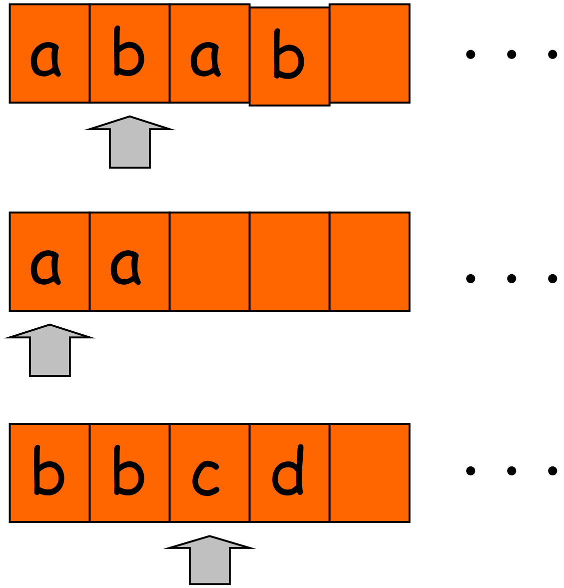
simulation of k-tape TM by single-tape TM:



- add new symbol **x** for each old x
- marks location of "virtual heads"



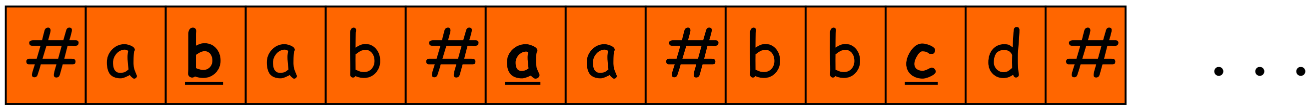
Multitape TMs



Repeat:

- scan tape, remembering the symbols under each virtual head in the state (how many new states needed?)
- make changes to reflect 1 step of M
- if hit #, shift to right to make room

if M halts, erase all but 1st string



Nondeterministic TMs

- An important variant: nondeterministic TM
- informally, several possible next configurations at each step
- formally, A NTM is a 7-tuple

  $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where:

  - everything is the same as a TM except the transition function:

$$\delta: Q \times \Gamma \rightarrow \wp(Q \times \Gamma \times \{L, R\})$$

NTM acceptance

- start configuration: $q_0 w$ (w is input)
- accepting config.: any config. with state q_{accept}
- rejecting config.: any config. with state q_{reject}

TM M accepts input w if there exist configurations $C_1, C_2, \ldots, C_k$

- C_1 is start configuration of M on input w
- $C_i \Rightarrow C_{i+1}$ for $i = 1, 2, 3, \ldots, k-1$
- C_k is an accepting configuration

Nondeterministic TMs

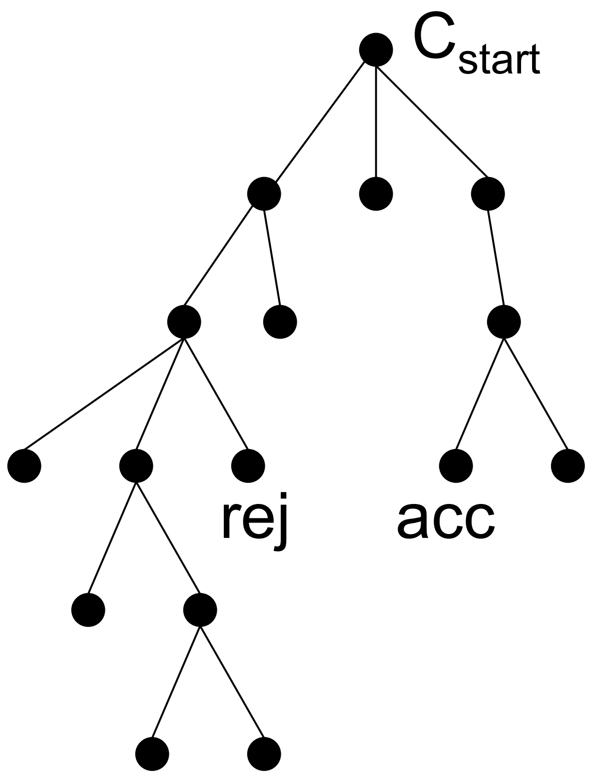
Theorem: every NTM has an equivalent (deterministic) TM.

Proof:

- Idea: simulate NTM with a deterministic TM

Nondeterministic TMs

Simulating NTM M with a deterministic TM:



- computations of M are a tree
- nodes are configurations
- fanout is b = maximum number of choices in transition function
- leaves are accept/reject configurations.

Nondeterministic TMs

Simulating NTM M with a deterministic TM:

- idea: breadth-first search of tree
- if M accepts: we will encounter accepting leaf and accept
- if M rejects: we will encounter all rejecting leaves, finish traversal of tree, and reject
- if M does not halt on some branch: we will not halt as that branch is infinite…

Nondeterministic TMs

Simulating NTM M with a deterministic TM:

- use a 3 tape TM:
  - tape 1: input tape (read-only)
  - tape 2: simulation tape (copy of M's tape at point corresponding to some node in the tree)
  - tape 3: which node of the tree we are exploring (string in {1,2,…b}*)
- Initially, tape 1 has input, others blank

Nondeterministic TMs

Simulating NTM M with a deterministic TM:

- STEP 1: copy tape 1 to tape 2
- STEP 2: use tape 2 to simulate M on one branch of its nondeterministic computation, consult the string on tape 3 to determine which choice to make at each step
  - if encounter blank, or a number larger than the number of choices available at this step, abort, go to STEP 3
  - if get to a rejecting configuration, go to STEP 3
  - if get to an accepting configuration, ACCEPT
- STEP 3: replace tape 3 with lexicographically next string and go to STEP 1

Recursive Enumerability

- Why is "Turing-recognizable" called RE?
- Definition: a language $L \subset \Sigma^*$ is recursively enumerable if there exists a TM (an "enumerator") that writes on its output tape

  $$\#x_1\#x_2\#x_3\#\ldots$$

  and $L = \{x_1, x_2, x_3, \ldots\}$.

- The output may be infinite

Recursive Enumerability

Theorem: A language is Turing-recognizable iff some enumerator enumerates it.

Proof:

($\Leftarrow$) Let E be the enumerator. On input w:

- Simulate E. Compare each string it outputs with w.
- If w matches a string output by E, accept.

Recursive Enumerability

Theorem: A language is Turing-recognizable iff some enumerator enumerates it.

Proof:

($\Rightarrow$) Let M recognize language L $\subset \Sigma^*$.

- let s_1, s_2, s_3, … be enumeration of Σ^* in lexicographic order.
- for i = 1,2,3,4,…
  - simulate M for i steps on s_1, s_2, s_3, …, s_i
- if any simulation accepts, print out that s_j

Church-Turing Thesis

- many other models of computation
  - we saw multitape TM, nondeterministic TM
  - others don't resemble TM at all
  - common features:
    - unrestricted access to unlimited memory
    - finite amount of work in a single step
  - every single one can be simulated by TM
  - many are equivalent to a TM
- The underlying class of algorithms that different computational models describe is unique!

Church-Turing Thesis

- the intuitive notion of an algorithm, or an effective or mechanical method, M:
  - M is set out in terms of a finite number of instructions;
  - M will, if carried out without error, produce the desired result in a finite number of steps;
  - M can (in practice or in principle) be carried out by a human being unaided by any machinery save paper and pencil;
  - M demands no insight or ingenuity on the part of the human being carrying it out;

Church-Turing Thesis

- Church-Turing thesis captures the intuitive notion of algorithms precisely
  - Turing's thesis: every effective computation can be carried out by a Turing machine.
  - Church: λ-calculus
  - They turned out to be equivalent

Church-Turing Thesis

- There are well-defined mathematical problems that cannot be solved by effective methods.
- E.g. Hilbert's tenth problem: determination of the solvability of a Diophantine equation.
  - Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: *To devise a process according to which it can be* determined by a finite number of operations *whether the equation is solvable in rational integers.*

Hilbertian Dream

- 1900, Hilbert --- *mathematicians should seek to express mathematics in the form of a consistent, complete and decidable formal system*.

Goedel and Turing

(1906-1978)

(1912-1954)

Goedel and Turing

- **1931**, Goedel --- *there can be no consistent and complete formal system of arithmetic (incompleteness theorem)*.
- **1936**, Church and Turing --- *no consistent formal system of arithmetic is decidable*.

High-Level Description

- Convince yourself that the following types of operations are easy to implement as part of TM "program"

  (but perhaps tedious to write out…)

  - copying
  - moving
  - incrementing/decrementing
  - arithmetic operations +, -, *, /

Encoding of Input

- the input to a TM is always a string in Σ^*
- we must encode our input as such a string
- examples:
  - tuples separated by #: #x#y#z
  - 0/1 matrix given by: #n#x# where $x \in \{0,1\}^{n^2}$
  - graph in adjacency list format
- any reasonable encoding is OK
- emphasize "encoding of X" by writing <X>