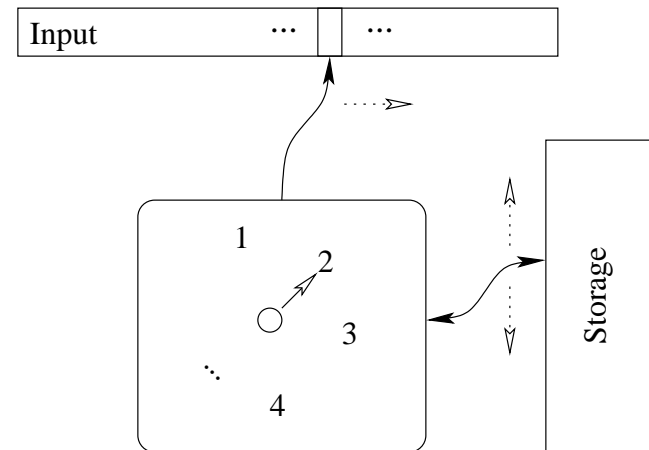


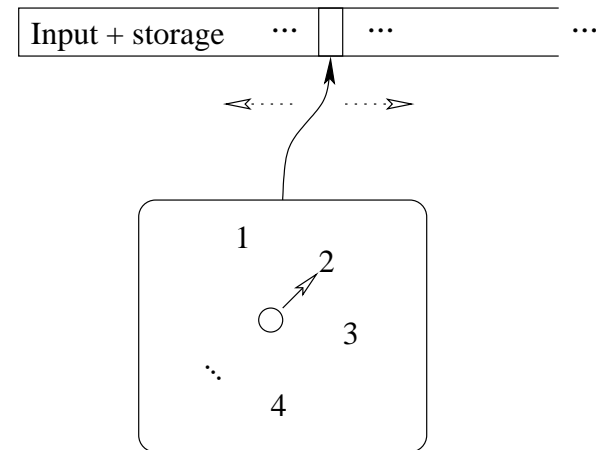
TURING MACHINES

- The most general kind of automaton
- Has access to a general form of storage



TURING MACHINES

- The most general kind of automaton
- Has access to a general form of storage
- In fact storage and input are put together on a single, infinite **tape**
- The machine can move the head in any direction
- Formally, $M = (K, \Sigma, \delta, s, H)$
- K, Σ as before; $\blacktriangleright, \# \in \Sigma$; $L, R \notin \Sigma$
 - $\sqcup, \leftarrow, \rightarrow$ also common instead of $\#, L, R$
- $H \subseteq K$ (halting states)
 - often $H = \{h\}$; more convenient that $h \notin K$
- $\delta : (K \setminus H) \times \Sigma \rightarrow K \times (\Sigma \cup \{L, R\})$ such that for all $q \in K \setminus H$:
 - $\delta(q, \blacktriangleright) = (p, b)$ implies $b = R$
 - $\delta(q, a) = (p, b)$ implies $b \neq \blacktriangleright$
- Configuration: $K \times \Sigma^* \times (\Sigma^*(\Sigma \setminus \{\#\}) \cup \{\varepsilon\})$
 - Configuration (q, wa, w') commonly written as $(q, w\underline{a}w')$

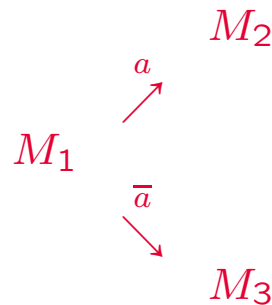


TURING MACHINES (CONT'D)

- Yields in one step: $(q_1, w_1 \underline{a_1} u_1) \vdash_M (q_2, w_2 \underline{a_2} u_2)$ iff $\delta(q_1, a_1) = (q_2, b)$ for some $b \in \Sigma \cup \{L, R\}$ and either
 - $b \in \Sigma, w_1 = w_2, u_1 = u_2, a_2 = b,$
 - $b = L, w_1 = w_2 a_2,$
 - * $u_2 = a_1 u_1$ if $a_1 \neq \#$ or $u_1 \neq \varepsilon$
 - * $u_2 = \varepsilon$ if $a_1 = \#$ or $u_1 = \varepsilon,$
 - $b = R, w_2 = w_1 a_1,$
 - * $u_1 = a_2 u_2$ if $a_2 \neq \#$
 - * $u_1 = u_2 = \varepsilon$ if $a_2 = \#$
- Yields: \vdash_M^* , the reflexive and transitive closure of \vdash_M
- Yields in n steps: $C_0 \vdash_M^n C_n$ iff $C_0 \vdash_M C_1 \vdash_M \cdots \vdash_M C_{n-1} \vdash_M C_n$

A COMPOSITIONAL APPROACH TO TURING MACHINES

- Basic machines: $a : \forall b \in \Sigma : \delta(s, b) = (h, a)$
 $L : \forall b \in \Sigma : \delta(s, b) = (h, L)$ $R : \forall b \in \Sigma : \delta(s, b) = (h, R)$
- Combining machines:

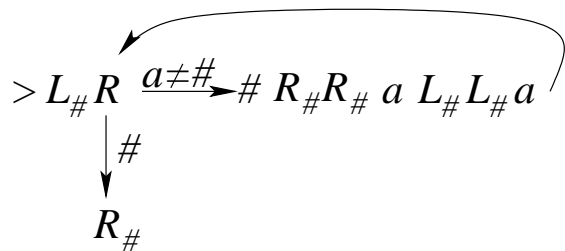


M_1 halts and then either M_2 or M_3 start, depending on whether a is read (or not) by the head when M_1 halts

- Supplementary, handy notations:
 - $M \rightarrow N$ or just MN for M followed immediately by N
 - $M \xrightarrow{x=\alpha} M_x$
 - R_x for $R \curvearrowright \bar{x}$; $R_{\bar{x}}$ for $R \curvearrowright x$; similar for $L_x, L_{\bar{x}}$

SAMPLE MACHINES

$$\begin{array}{lcl} (s, \#) & = & (q_2, L) \quad (q_3, \#) = (q_4, I) \\ (q_2, I) & = & (q_2, \#) \quad (q_4, I) = (q_4, R) \\ (q_2, \#) & = & (q_3, L) \quad (q_4, \#) = (h, \#) \\ (q_3, I) & = & (q_3, L) \end{array}$$



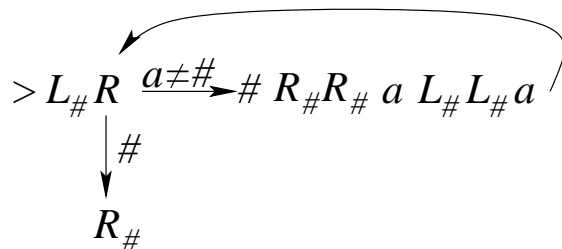
SAMPLE MACHINES

$$\begin{array}{l}
 (s, \#) = (q_2, L) \\
 (q_2, I) = (q_2, \#) \\
 (q_2, \#) = (q_3, L) \\
 (q_3, I) = (q_3, L)
 \end{array}
 \quad
 \begin{array}{l}
 (q_3, \#) = (q_4, I) \\
 (q_4, I) = (q_4, R) \\
 (q_4, \#) = (h, \#)
 \end{array}$$

More concisely: $\triangleright L\#L\bar{I}IR\bar{I}$

A copy machine:

To accept $\{a^n b^n c^n : n \geq 0\}$:

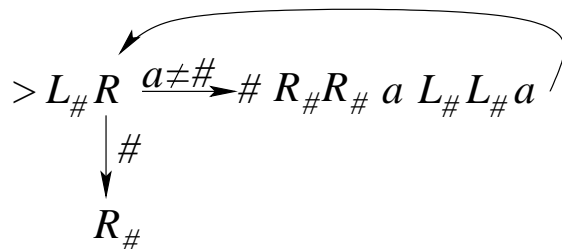


SAMPLE MACHINES

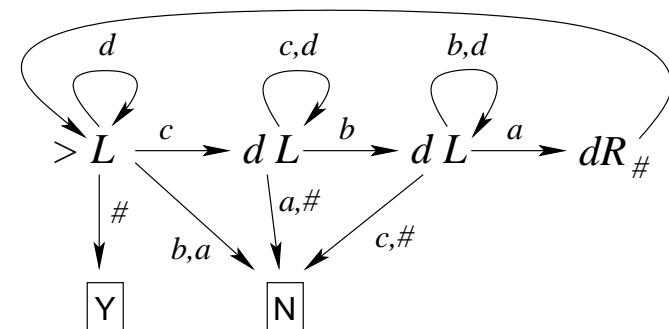
$$\begin{array}{lcl}
 (s, \#) & = & (q_2, L) \\
 (q_2, I) & = & (q_2, \#) \\
 (q_2, \#) & = & (q_3, L) \\
 (q_3, I) & = & (q_3, L)
 \end{array}
 \quad
 \begin{array}{lcl}
 (q_3, \#) & = & (q_4, I) \\
 (q_4, I) & = & (q_4, R) \\
 (q_4, \#) & = & (h, \#)
 \end{array}$$

More concisely: $\triangleright L\#L\bar{I}R\bar{I}$

A copy machine:



To accept $\{a^n b^n c^n : n \geq 0\}$:



RECURSIVE LANGUAGES AND FUNCTIONS

- Two variants of accepting a language: we always halt and produce a positive or negative answer, **or** we either halt or not halt
- **Recursive languages**: Languages **decided** by Turing machines
 - two halting states, one accepting the other rejecting **or**
 - one halting state, writes “Y” or “N” on the tape
 - decided = **always halt**
- **Recursive functions**:
 - One halting state, output is what is left on the tape
 - $M(w)$ = output of M on input w (defined only when M halts)
 - $f : \Sigma^* \rightarrow \Sigma^*$ is **recursive** iff $\exists M : \forall w \in \Sigma^* : M(w) = f(w)$
- A Turing machine can also compute **numerical functions** using an **encoding**:
 - we put $\{0, 1, ;\} \in \Sigma$ and then M computes $f : \mathbb{N}^k \rightarrow \mathbb{N}$ iff $M(w_1; w_2; \dots; w_k) = f(w_1, w_2, \dots, w_k)$ for all $w_1, w_2, \dots, w_k \in \{0\} \cup \{1\}\{0, 1\}^*$

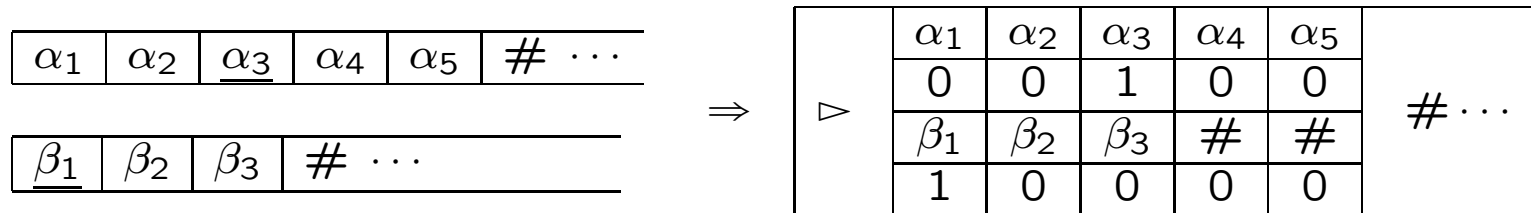
RECURSIVELY ENUMERABLE LANGUAGES

- M **semidecides** $L \in \Sigma^*$ whenever M halts on input w iff $w \in L$ for all $w \in \Sigma^*$
- **Recursively enumerable languages** include exactly all the languages semidecided by Turing machines
- **Any recursive language is recursively enumerable**
 - $M' = M \xrightarrow{N} N \hookrightarrow$ semidecides the language decided by M
- **Recursive languages are closed under complementation**
 - We just flip the accepting and rejecting state (or the writing of Y and N)

EXTENSION OF TURING MACHINES

Multiple tapes (natural; actual definition on p. 201) $\delta : (K \setminus H) \times \Sigma^k \rightarrow K \times (\Sigma \cup \{L, R\})^k$

- We put all the tapes as tracks on a single tape



- On every move we must scan the whole non-blank tape \Rightarrow **quadratic slowdown**

Two-way infinite tape

- We pick a point and we fold the tape at that point into a two-track tape
- Every state q is replaced by two states q^\uparrow and q^\downarrow
- q^\uparrow behaves like the original q and operates on the upper track
- q^\downarrow behaves like the original except that it reverses the directions of movement (and operates on the lower track)
- **Constant slowdown**

EXTENSION OF TURING MACHINES (CONT'D)

Nondeterministic Turing machine: Same definition except that we have a transition **relation** $\Delta \subseteq (K \setminus H) \times \Sigma \times K \times (\Sigma \cup \{L, R\})$

- Configuration, \vdash_M , etc. identical, but now a configuration can yield in one step more than one configuration
- M **accepts** w iff $(s, \#w\#) \vdash_M^* (h, uav)$ for some $u, v \in \Sigma^*$, $a \in \Sigma$
 - We have **one** terminating computation (others may be non-terminating)
- M **semidecides** a language L whenever M accepts w iff $w \in L$
- M **decides** L iff

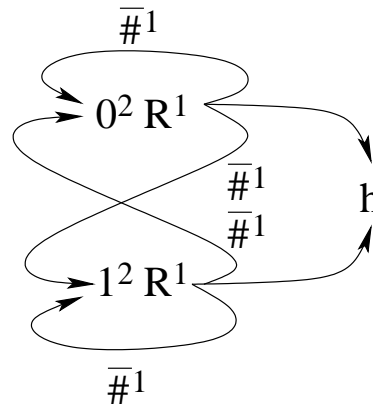
EXTENSION OF TURING MACHINES (CONT'D)

Nondeterministic Turing machine: Same definition except that we have a transition **relation** $\Delta \subseteq (K \setminus H) \times \Sigma \times K \times (\Sigma \cup \{L, R\})$

- Configuration, \vdash_M , etc. identical, but now a configuration can yield in one step more than one configuration
- M **accepts** w iff $(s, \#w\#) \vdash_M^* (h, uav)$ for some $u, v \in \Sigma^*$, $a \in \Sigma$
 - We have **one** terminating computation (others may be non-terminating)
- M **semidecides** a language L whenever M accepts w iff $w \in L$
- M **decides** L iff
 1. For some finite N (depending on M and w) there exists no configuration C such that $(s, \#w\#) \vdash_M^N C$ (M **always halts**)
 2. $w \in L$ iff $(s, \#w\#) \vdash_M^* (h, uav)$ for some $u, v \in \Sigma^*$, $a \in \Sigma$ (**some** accepting computation, others may be rejecting)
- M **computes** f iff $\forall w \in \Sigma : (s, \#w\#) \vdash_M^* (h, \#f(w)\#)$ **and** Item 1 above applies

EXAMPLE OF NONDETERMINISTIC COMPUTATION: COMPOSITE NUMBERS

- Language: $\{x \in \{0\} \cup \{1\}\{0, 1\}^* : \exists p, q \in \{0\} \cup \{1\}\{0, 1\}^* : x = p \times q\}$
- Decided by a very simple and fast nondeterministic Turing machine with two tapes:
 1. First tape contains input x (a binary number)
 2. **Guess** on the second tape a number $p \leq x$ and again a number $q \leq x$



3. Multiply p and q , compare the result with x , accept iff they are equal

DETERMINISM VERSUS NONDETERMINISM

Theorem: If a nondeterministic Turing machine M decides/semidecides L or computes f then there exists a deterministic Turing machine M' that does the same thing

- **Crux:** Let $C \vdash_M C_1, C \vdash_M C_2, \dots, C \vdash_M C_n$. Is there an upper bound for n ?

DETERMINISM VERSUS NONDETERMINISM

Theorem: If a nondeterministic Turing machine M decides/semidecides L or computes f then there exists a deterministic Turing machine M' that does the same thing

- **Crux:** Let $C \vdash_M C_1, C \vdash_M C_2, \dots, C \vdash_M C_n$. Is there an upper bound for n ?
 - Sure: $n \leq r = |K| \times (|\Sigma| + 2)$
- We first construct a machine M_d that receives the input and (on a different tape) a **path description** $i_1 i_2 \dots i_k$ for some k , with $1 \leq i_j \leq r$
 - M_d carries on k steps of the computation of M along the path given; it is **deterministic**
- Then M' will be a 3-tape machine; tape 1 contains input w and remains unchanged, tapes 2 and 3 are initially empty
 1. M' then generate the next path description on tape 3 in **lexicographic order**
 2. Then M' copies w on tape 2 and launches M_d
 3. If M_d is successful, then M' reports success; otherwise repeat from Step 1
- **Exponential slowdown**