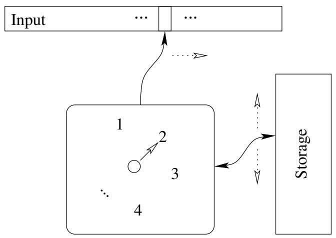
CS 455/555: Turing machines

Stefan D. Bruda

Fall 2020

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; $\triangleright, \# \in \Sigma$; $L, R \notin \Sigma$
  - $\sqcup$, $\leftarrow$, $\rightarrow$ also common instead of $\#$, L, R
- $H \subseteq K$ (halting states)
  - often $H = \{h\}$; more convenient that $h \notin K$
- $\delta : (K \backslash H) \times \Sigma \rightarrow K \times (\Sigma \cup \{L, R\})$ such that for all $q \in K \backslash H$:
  - $\delta(q, \triangleright) = (p, b)$ implies $b = R$
  - $\delta(q, a) = (p, b)$ implies $b \neq \triangleright$
- Configuration: $K \times \Sigma^* \times (\Sigma^*(\Sigma \backslash \{\#\}) \cup \{\varepsilon\})$
  - 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 $C_0 \vdash_M C_1 \vdash_M \cdots \vdash_M C_{n-1} \vdash_M C_n$

Compositional notation for Turing machines

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

$$M_1 \xrightarrow{a} M_2 \qquad M_1 \xrightarrow{\bar{a}} M_3$$

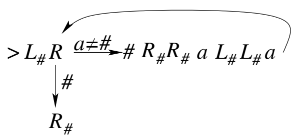
  M_1 halts and then either M_2 or M_3 start, depending on whether a is read (or not) by the head when M_1 halts
- Supplementary, handy notations:
  - $M \rightarrow N$ or just MN for M followed immediately by N
  - $M \xrightarrow{x=\alpha} M_x$
  - R_x for $R \circlearrowleft_{\bar{x}}$; $R_{\bar{x}}$ for $R \circlearrowleft_x$; similar for L_x, $L_{\bar{x}}$

Sample machines

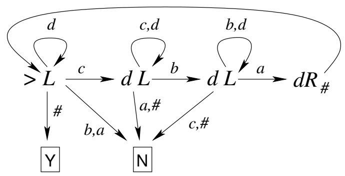
$$\begin{array}{rcl} (s,\#) & = & (q_2, L) \\ (q_2, l) & = & (q_2, \#) \\ (q_2, \#) & = & (q_3, L) \\ (q_3, l) & = & (q_3, L) \\ (q_3, \#) & = & (q_4, l) \\ (q_4, l) & = & (q_4, R) \\ (q_4, \#) & = & (h, \#) \end{array}$$

More concisely:
$> L\# L_{\bar{l}} l R_{\bar{l}}$

A copy machine:



TM accepting $\{a^n b^n c^n : n \geqslant 0\}$:



Recursive languages and functions

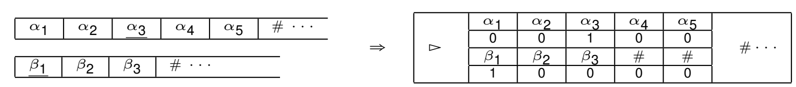
- Two variants of accepting a language: we always halt and produce a positive or negative answer, or we either halt or not halt
- Recursive languages: Languages decided by Turing machines
  - two halting states, one accepting the other rejecting or
  - one halting state, writes "Y" or "N" on the tape
  - decides = always halts
- 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)
  - $f : \Sigma^* \to \Sigma^*$ 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 $\{0, 1, ;\} \subseteq \Sigma$ and then M computes $f : \mathbb{N}^k \to \mathbb{N}$ iff $M(w_1; w_2; \cdots w_k) = f(w_1, w_2, \ldots, w_k)$ for all $w_1, w_2, \ldots, w_k \in \{0\} \cup \{1\}\{0, 1\}^*$

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
  - $M' = M \xrightarrow{\mathrm{N}} \mathrm{N} \circlearrowleft$ 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)
  $\delta : (K \setminus H) \times \Sigma^k \to K \times (\Sigma \cup \{L, R\})^k$
  - We put all the tapes as tracks on a single tape



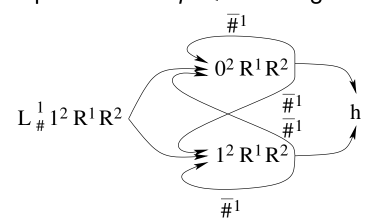
  - On every move we must scan the whole non-blank tape ⇒ quadratic slowdown
- Two-way infinite tape
  - We pick a point and we fold the tape at that point into a two-track tape
  - Every state q is replaced by two states $q^\uparrow$ and $q^\downarrow$
  - $q^\uparrow$ behaves like the original q and 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 $\Delta \subseteq (K \backslash H) \times \Sigma \times K \times (\Sigma \cup \{L, R\})$
- Configuration, $\vdash_M$, etc. identical, but now a configuration can yield in one step more than one configuration
- M accepts w iff $(s, \#w\underline{\#}) \vdash_M^* (h, u\underline{a}v)$ 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 $(s, \#w\underline{\#}) \vdash_M^N C$ (M always halts)
  2. $w \in L$ iff $(s, \#w\underline{\#}) \vdash_M^* (h, u\underline{a}v)$ for some $u, v \in \Sigma^*$, $a \in \Sigma$ (some accepting computation, others may be rejecting)
- M computes f iff $\forall\, w \in \Sigma : (s, \#w\underline{\#}) \vdash_M^* (h, \#f(w)\underline{\#})$ and Item 1 above applies

Example of nondeterministic computation: Composite numbers

- Language: $\{x \in \{0\} \cup \{1\}\{0,1\}^* : \exists\, p, q \in \{1\}\{0,1\}\{0,1\}^* : x = p \times q\}$
- 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 $p \leqslant x$ and again a number $q \leqslant x$



  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 $C \vdash_M C_1$, $C \vdash_M C_2$, ..., $C \vdash_M C_n$. Is there an upper bound for n?
  - Sure: $n \leqslant r = |K| \times (|\Sigma| + 2)$
- 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 $1 \leqslant i_j \leqslant r$
  - 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