TURING MACHINES

- Turing machines are used as the formal model of algorithms.
- Turing machines can simulate arbitrary algorithms with inconsequential loss of efficiency using a single data structure: a string of symbols.

Definition. A Turing machine is a quadruple $M = (K, \Sigma, \delta, s)$:
K: a finite set of *states*
Σ: a finite set of symbols (*alphabet* of M) with $\sqcup, \triangleright \in \Sigma$
δ: $K \times \Sigma \mapsto (K \cup \{\mathrm{h}, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\rightarrow, \leftarrow, -\}$
(*transition function* of M)
h: halting state; “yes”: accepting state; “no”: rejecting state
cursor directions: $\rightarrow$ (move right), $\leftarrow$ (move left), $-$ (stay)

Example Turing machine $M = (K, \Sigma, \delta, s)$:

$K = \{s, q\}$

$\Sigma = \{0, 1, \sqcup, \triangleright\}$

δ :

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
$s,$	0	$(s, 0, \rightarrow)$
$s,$	1	$(s, 1, \rightarrow)$
$s,$	$\sqcup$	$(q, \sqcup, \leftarrow)$
$s,$	$\triangleright$	$(s, \triangleright, \rightarrow)$
$q,$	0	$(h, 1, -)$
$q,$	1	$(q, 0, \leftarrow)$
$q,$	$\sqcup$	$(q, \sqcup, -)$
$q,$	$\triangleright$	$(h, \triangleright, \rightarrow)$

- Function δ is the “program” of the machine.
- For current state $q \in K$ and current symbol $\sigma \in \Sigma$,
  $\delta(q, \sigma) = (p, \rho, D)$ where p is the new state,
  ρ is the symbol to be overwritten on σ and
  $D \in \{\rightarrow, \leftarrow, -\}$ is the direction in which the cursor will move.
- For any states p, q, $\delta(q, \triangleright) = (p, \rho, D)$ with $\rho = \triangleright$ and $D = \rightarrow$.
- If the machine moves off the right end of the string, it reads $\sqcup$ (the string becomes longer but it cannot become shorter).

- The program starts with (i) initial state s,
  (ii) the string initialized to $\triangleright x$ where x is a finitely long string in $(\Sigma - \{\sqcup\})^*$ (x is the *input* of the machine) and
  (iii) the cursor pointing to $\triangleright$.
- A machine has *halted* iff one of the 3 halting states ($\mathrm{h}, \text{“yes”}, \text{“no”}$) has been reached.
- If “yes” has been reached, the machine *accepts* the input.
  If “no” has been reached, the machine *rejects* the input.
- *Output* $M(x)$ of a machine M on input x:
  (i) If M accepts/rejects, then $M(x) = \text{“yes”}/\text{“no”}$.
  (ii) If h has been reached, $M(x) = y$
  where $\triangleright y \sqcup \sqcup \ldots$ is the string of M at the time of halting.
  (iii) If M never halts on input x, then $M(x) = \nearrow$

A formal model of the operation

- A configuration (q, w, u):

  $q \in K$ is the current state and $w, u \in \Sigma^*$ where
  (i) w is the string to the left of the cursor including the symbol scanned by the cursor and
  (ii) u is the string to the right of the cursor.

- The relation $\overset{M}{\to}$ (yields in one step): $(q, w, u) \overset{M}{\to} (q', w', u')$
  Let σ be the last symbol of w and $\delta(q, \sigma) = (p, \rho, D)$.
  Then $q' = p$, and w', u' are obtained according to (p, ρ, D).

  For example, if $D = \to$, then
  (i) w' is w with its last symbol replaced by ρ and the first symbol of u appended to it ($\sqcup$ if u is empty) and
  (ii) u' is u with the first removed (or empty, if u is empty).

- Yields in k steps: $(q, w, u) \overset{M^k}{\to} (q', w', u')$
  iff there are configurations $(q_i, w_i, u_i), i = 1, \ldots, k+1$ such that
  $(q, w, u) = (q_1, w_1, u_1)$
  $(q_i, w_i, u_i) \overset{M}{\to} (q_{i+1}, w_{i+1}, u_{i+1}), i = 1, \ldots, k$
  $(q', w', u') = (q_{k+1}, w_{k+1}, u_{k+1})$

- Yields: $(q, w, u) \overset{M^*}{\to} (q', w', u')$

  iff there is some $k \geq 0$ such that $(q, w, u) \overset{M^k}{\to} (q', w', u')$.

Turing machines as algorithms

Turing machines are natural for solving problems on strings:

- Let $L \subset (\Sigma - \{\sqcup\})^*$ be a language.

  A Turing machine M *decides* L iff for every string $x \in (\Sigma - \{\sqcup\})^*$, if $x \in L, M(x) =$ "yes" and if $x \notin L, M(x) =$ "no" .

- If L is decided by a Turing machine, L is a *recursive* language.

- A Turing machine M *accepts* L iff for every string $x \in (\Sigma - \{\sqcup\})^*$, if $x \in L$, then $M(x) =$ "yes" but if $x \notin L, M(x) = \nearrow$.

- If L is accepted by some Turing machine, L is a *recursively enumerable* language.

  Proposition. If L is recursive, then it is recursively enumerable.

- String functions $f : (\Sigma - \{\sqcup\})^* \mapsto \Sigma$

  A Turing machine M *computes* f iff for every string $x \in (\Sigma - \{\sqcup\})^*$, $M(x) = f(x)$.

- If such an M exists, f is called a *recursive function*.

Solving problems using Turing machines

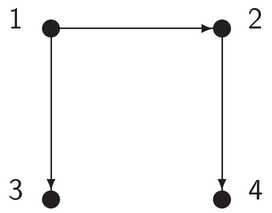
- Instances of the problem need to be represented by strings.
- Solving a decision problem amounts to deciding the language consisting of the encodings of the "yes" instances of the problem.
- An optimization problem is solved by a Turing machine that computes the appropriate function from strings to strings (where the output is similarly represented as a string).

How does representation affect solvability?

- Any "finite" mathematical object can be represented by a finite string over an appropriate alphabet.

Example.

Graph:



Representations as a string:

"$\{(1,10),(1,11),(10,100)\}$"

"$(0110,0001,0000,0000)$"

Representation vs. solvability?

- All acceptable encodings are related polynomially:

  If A and B are both "reasonable" representations of the same set of instances, and representation A of an instance is a string with n symbols, the representation B of the same instance has length at most $p(n)$ for some polynomial p.
- Exception: unary representation of numbers requires exponentially more symbols than the binary representation.
- Reasonable succinct input representation is assumed.

  In particular, numbers are always represented in binary.

Turing machines with multiple strings

- Turing machines with multiple strings and associated cursors are more convenient.
- They can be simulated by an ordinary Turing machine with an inconsequential loss of efficiency.
- A k-string Turing machine, where $k \geq 1$ is an integer, is a quadruple $M = (K, \Sigma, \delta, s)$ where δ has been generalized:

  $\delta\colon\ K \times \Sigma^k \mapsto (K \cup \{\mathrm{h}, \text{"yes"}, \text{"no"}\}) \times (\Sigma \times \{\rightarrow, \leftarrow, -\})^k$

- $\delta(q, \sigma_1, \ldots, \sigma_k) = (p, \rho_1, D_1, \ldots, \rho_k, D_k)$
  If M is in the state q, the cursor of the first string is scanning σ_1, that of the second σ_2 and so on, then the next state is p, the first cursor will write ρ_1 and move D_1 and so on.
- A configuration: $(q, w_1, u_1, \ldots, w_k, u_k)$
- $\xrightarrow{M}$ defined similarly as for the ordinary machine.
- A k-string machine starts from the configuration $(s, \triangleright, x, \triangleright, \epsilon, \ldots, \triangleright, \epsilon)$

- Output is defined as for ordinary machines:
  If $(s, \triangleright, x, \triangleright, \epsilon, \ldots, \triangleright, \epsilon) \xrightarrow{M^*}$ ("yes", $w_1, u_1, \ldots, w_k, u_k$), then $M(x) =$ "yes".
  If $(s, \triangleright, x, \triangleright, \epsilon, \ldots, \triangleright, \epsilon) \xrightarrow{M^*}$ ("no", $w_1, u_1, \ldots, w_k, u_k$), then $M(x) =$ "no".
  If $(s, \triangleright, x, \triangleright, \epsilon, \ldots, \triangleright, \epsilon) \xrightarrow{M^*} (\mathrm{h}, w_1, u_1, \ldots, w_k, u_k)$, then $M(x) = y$ where y is $w_k u_k$ with the leading $\triangleright$ and trailing $\sqcup$s removed. (Note: output read from the last (kth) string.)
- The *time required* by M on input x is t iff $(s, \triangleright, x, \triangleright, \epsilon, \ldots, \triangleright, \epsilon) \xrightarrow{M^t} (H, w_1, u_1, \ldots, w_k, u_k)$ where $H \in \{\mathrm{h}, \text{"yes"}, \text{"no"}\}$.
  If $M(x) = \nearrow$, then the time required is thought to be ∞.

Complexity classes

- Performance measured by the amount of time (or space) required on instances of size n using a function of n.
- Machine M *operates within time* $f(n)$ if for any input string x, the time required by M on x is at most $f(|x|)$.
- Function $f(n)$ is a *time bound* for M.
- A *complexity class* $\mathbf{TIME}(f(n))$ is a set of *languages* L such that L is decided by a multi string Turing machine operating within time $f(n)$.
- Notice that *worst-case inputs* are taken into account.

Multi string vs. single string machines

Theorem. Given any k-string Turing machine M operating within time $f(n)$, we can construct a Turing machine M' operating within time $\mathcal{O}(f(n)^2)$ and such that for any input x, $M(x) = M'(x)$.

Proof sketch:

- M' simulates M using one string by concatenating the strings of M: $(q, w_1, u_1, \ldots, w_k, u_k) \mapsto (q, \triangleright, w_1' u_1 \triangleleft w_2' u_2 \triangleleft \ldots w_k' u_k \triangleleft \triangleleft)$
- The strings of M have a total length of $\mathcal{O}(kf(n))$.
  To simulate one step of M, M' needs $\mathcal{O}(k^2 f(n))$ steps.
- M makes at most $f(n)$ steps.
  M' makes $\mathcal{O}(f(n)^2)$ steps (k is fixed and independent of x).