TURING MACHINES

- Basics definitions
- Turing machines as algorithms
- Turing machines with multiple strings
- Linear speedup
- Space bounds

(C. Papadimitriou: *Computational complexity*, Chapters 2.1–2.5)

Additional references:

M. Sipser: *Introduction to the Theory of Computation*, Chapter 3.
P. Orponen: *Tietojenkäsittelyteorian perusteet*, Luku 4.

1. Basic Definitions

- 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 $M = (K, \Sigma, \delta, s)$ with

a finite set of *states* K,
a finite set of symbols Σ (*alphabet* of M) so that $\sqcup, \triangleright \in \Sigma$,
a *transition function* δ:

$$K \times \Sigma \rightarrow (K \cup \{\mathrm{h}, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\rightarrow, \leftarrow, -\},$$

a halting state h, an accepting state “yes”, a rejecting state “no”, and cursor directions: $\rightarrow$ (right), $\leftarrow$ (left), and $-$ (stay).

Example. Consider a Turing machine $M = (K, \Sigma, \delta, s)$ with $K = \{s, q\}$, $\Sigma = \{0, 1, \sqcup, \triangleright\}$ and a transition function δ defined as follows:

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

The machine computes $n+1$ for a natural number n in *binary*.

Transition functions

- Function δ is the “program” of the machine.
- For the current state $q \in K$ and the 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 and 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; thus it keeps track of the space used by the machine).

- 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 (h, "yes", "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 $M(x) =$ "yes" / "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 $M(x) = \nearrow$

Operational semantics

- 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 $\xrightarrow{M}$ (*yields* in one step): $(q, w, u) \xrightarrow{M} (q', w', u')$
  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).

Example. If $D = \rightarrow$, 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).

Configurations reached in several steps

- *Yields* in k steps: $(q, w, u) \xrightarrow{M^k} (q', w', u')$
  iff there are configurations $(q_i, w_i, u_i), i = 1, \ldots, k+1$ such that –
  $(q, w, u) = (q_1, w_1, u_1)$,
  – $(q_i, w_i, u_i) \xrightarrow{M} (q_{i+1}, w_{i+1}, u_{i+1}), i = 1, \ldots, k$, and
  – $(q', w', u') = (q_{k+1}, w_{k+1}, u_{k+1})$
- *Yields:* $(q, w, u) \xrightarrow{M^*} (q', w', u')$
  iff there is some $k \geq 0$ such that $(q, w, u) \xrightarrow{M^k} (q', w', u')$.
- Therefore $\xrightarrow{M^*}$ is the transitive and reflexive closure of $\xrightarrow{M}$.

2. 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 *computes* a (string) function $f : (\Sigma - \{\sqcup\})^* \rightarrow \Sigma^*$ iff for every string $x \in (\Sigma - \{\sqcup\})^*$, $M(x) = f(x)$.
- If such an M exists, f is called a *recursive function*.

Example. Transition function δ for checking even parity of $x \in \{0,1\}^*$:

$p \in K$	$\sigma \in \Sigma$	$\delta(p,\sigma)$	$p \in K$	$\sigma \in \Sigma$	$\delta(p,\sigma)$
s,	$\triangleright$	$(s, \triangleright, \rightarrow)$	t,	$\triangleright$	$(t, \triangleright, \rightarrow)$
s,	0	$(s, 0, \rightarrow)$	t,	0	$(t, 0, \rightarrow)$
s,	1	$(t, 1, \rightarrow)$	t,	1	$(s, 1, \rightarrow)$
s,	$\sqcup$	$(\text{"yes"}, \sqcup, -)$	t,	$\sqcup$	$(\text{"no"}, \sqcup, -)$

The respective Turing machine M decides $101 \in \{0,1\}^*$ as follows:

$$\begin{aligned}
(s, \triangleright, 101) &\xrightarrow{M} (s, \triangleright 1, 01) \\
&\xrightarrow{M} (t, \triangleright 10, 1) \\
&\xrightarrow{M} (t, \triangleright 101, \varepsilon) \\
&\xrightarrow{M} (s, \triangleright 101\sqcup, \varepsilon) \\
&\xrightarrow{M} (\text{"yes"}, \triangleright 101\sqcup, \varepsilon).
\end{aligned}$$

Recursively enumerable languages

- 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.
- We will later encounter examples of r.e. languages.

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

☞ *The terms recursive and recursively enumerable suggest that Turing machines are equivalent in power with arbitrarily general (recursive) computer programs.*

Solving problems using Turing machines

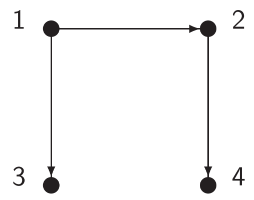
- Instances of the problem need to be represented by strings.
- Solving a decision problem amounts to deciding the language consisting of the encodings of the "yes" instances of the problem.
- An optimization problem is solved by a Turing machine that computes the appropriate function from strings to 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.
- A reasonably succinct input representation is assumed.

  In particular, numbers are always represented in binary.

3. Turing Machines with Multiple Strings

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

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

- This definition yields an ordinary Turing machine when $k = 1$.

Generalized transitions

- Transitions are determined by $\delta(q, \sigma_1, \ldots, \sigma_k) = (p, \rho_1, D_1, \ldots, \rho_k, D_k)$.

  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 is defined as a $2k+1$-tuple $(q, w_1, u_1, \ldots, w_k, u_k)$.
- A k-string machine with input x starts from the configuration

$$(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon).$$

- Relations $\xrightarrow{M}$, $\xrightarrow{M^t}$, $\xrightarrow{M^*}$ are defined in analogy to ordinary machines.

- Output is defined as for ordinary machines:

  If $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{M^*} (\text{"yes"}, w_1, u_1, \ldots, w_k, u_k)$, then $M(x) = \text{"yes"}$.

  If $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{M^*} (\text{"no"}, w_1, u_1, \ldots, w_k, u_k)$, then $M(x) = \text{"no"}$.

  If $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{M^*} (\mathrm{h}, w_1, u_1, \ldots, w_k, u_k)$, then $M(x) = y$ where y is $w_k u_k$ with the leading $\triangleright$ and trailing $\sqcup$s removed. (*Output* read from the *last (kth) string*.)
- The *time required* by M on input x is t iff $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{M^t} (H, w_1, u_1, \ldots, w_k, u_k)$ 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 decided by a multistring Turing machine operating within time $f(n)$.
- Notice that *worst-case inputs* are taken into account.

Multiple strings vs. a single string

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

Proof sketch:

- M' is based on an extended alphabet $\Sigma' = \Sigma \cup \underline{\Sigma} \cup \{\triangleright', \triangleleft\}$.
- M' represents a configuration of M by concatenation

$$(q, w_1, u_1, \ldots, w_k, u_k) \mapsto (q, \triangleright, w_1' u_1 \triangleleft w_2' u_2 \triangleleft \ldots w_k' u_k \triangleleft \triangleleft)$$

  where each w_i' is w_i with the leading $\triangleright$ replaced by $\triangleright'$ and the last symbol σ_i by $\underline{\sigma_i}$ to keep track of cursor positions.
- Initial configuration: $(s, \triangleright, \underline{\triangleright'} x \triangleleft \underline{\triangleright'} \triangleleft \ldots \underline{\triangleright'} \triangleleft \triangleleft)$

- The simulation of a step of M by M' takes place as follows:
  1. pass: symbols underlined (scanned) on the k strings
  2. pass: change in the underlined (scanned) symbols
- The strings of M have a total length of $O(kf(n))$.
  To simulate one step of M, M' needs $O(k^2 f(n))$ steps.
- Since M makes at most $f(n)$ steps, M' makes $O(f(n)^2)$ steps (k is fixed and independent of x).

☞ Thesis: *No conceivable "realistic" improvement on the Turing machine will increase the domain of the language such machines decide, or will affect their speed more than polynomially.*

4. Linear Speedup

- When using Turing machines, the rate of growth of the time/space requirements is important but the precise multiplicative and additive constants are not.
- In practice this also holds to some extent because of continuously improving computer hardware.

Theorem. Let $L \in \mathbf{TIME}(f(n))$. Then for any $\varepsilon > 0$, $L \in \mathbf{TIME}(f'(n))$ where $f'(n) = \varepsilon f(n) + n + 2$.

Proof sketch

- Let $M = (K, \Sigma, \delta, s)$ be a k-string machine deciding L in time $f(n)$.
  We construct a k'-string machine $M' = (K', \Sigma', \delta', s')$ operating within time bound $f'(n)$ and simulating M.
  (If $k > 1$, $k' = k$ and if $k = 1$, then $k' = 2$).
- Performance savings are obtained by adding word length:
  Each symbol of M' encodes several symbols of M and each move of M' several moves of M.
- Given M and ε we take some integer m and use m-tuples of symbols of M in M'.
- The linear term $(n+2)$ in the theorem is due to condensing input.

Proof sketch — cont'd

- M' simulates m steps of M in at most a constant (6) number of steps in a stage.
- In such a stage M' reads the adjacent symbols (m-tuples) on both sides of the cursors (this takes 4 steps).
  The state of M' records all symbols at or next to all cursors.
  Now M' can predict the next m moves of M which can be implemented in 2 steps.
- The time spent by M' on input x is $|x| + 2 + 6\lceil f(|x|)/m \rceil$.
- The speedup is obtained if $m = \lceil 6/\varepsilon \rceil$.
- Notice that a lot of new states have to be added: $|K| * m^k |\Sigma|^{3mk}$.

Consequences of the linear speedup theorem

- It holds for any time bound $f(n)$ such that $f(n) \geq n$,
  (i) if $f(n) = cn$, then $f'(n) \approx n$ and
  (ii) if $f(n)$ is superlinear, e.g., $f(n) = 20n^2 + 11n$, then $f'(n) \approx n^2$
  (*arbitrary linear speedup*).
- If L is polynomially decidable, then $L \in \mathbf{TIME}(n^k)$ for some integer $k > 0$.

Definition. The set of all languages decidable by Turing machines in polynomial time $\mathbf{P}$ is defined as the union

$$\bigcup_{k>0} \mathbf{TIME}(n^k)$$

5. Space bounds

- Strings cannot become shorter during computation.
- Thus the sum of lengths of the final strings provides a preliminary definition of the space consumed by a computation.
- There is an overcharge: sublinear space bounds are not covered!
  Example. The language of palindromes can be decided by a 3-string Turing machine in logarithmic space.
- This suggests us to exclude the effects of reading the input and writing the output as regards the consumption of space.

Turing machines with input and output

Definition. A k-string Turing machine ($k > 2$) with *input and output* is an ordinary k-string Turing machine with the following restrictions on the program δ:

If $\delta(q, \sigma_1, \ldots, \sigma_k) = (p, \rho_1, D_1, \ldots, \rho_k, D_k)$, then

(a) $\rho_1 = \sigma_1$ (read-only input string),

(b) $D_k \neq \leftarrow$ (write-only output string), and

(c) if $\sigma_1 = \sqcup$, then $D_1 = \leftarrow$ (end of input respected).

Proposition. For any k-string Turing machine M operating within time bound $f(n)$ there is a $(k+2)$-string Turing machine M' with input and output which operates within time bound $O(f(n))$.

Space consumption

Definition. Suppose that for a k-string Turing machine M and an input x, $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{M^*} (H, w_1, u_1, \ldots, w_k, u_k)$ where $H \in \{\text{“yes”}, \text{“no”}, \mathrm{h}\}$ is a halting state.

Then the space required by M on input x is $\sum_{i=1}^{k} |w_i u_i|$.

If M is a Turing machine *with input and output*, then the space required by M on input x is $\sum_{i=2}^{k-1} |w_i u_i|$.

Let $f : \mathbf{N} \mapsto \mathbf{N}$.
Turing machine M operates within space bound $f(n)$ if for any input x, M requires space at most $f(|x|)$.

Space complexity classes

Definition. A *space complexity class* $\mathbf{SPACE}(f(n))$ is a set of *languages* L decidable by a Turing machine with input and output operating within space bound $f(n)$.

Definition. The class $\mathbf{SPACE}(\log(n))$ is denoted by $\mathbf{L}$.

Example. The language of palindromes belongs to $\mathbf{L}$.

Theorem. Let $L \in \mathbf{SPACE}(f(n))$. Then for any $\varepsilon > 0$, $L \in \mathbf{SPACE}(2 + \varepsilon f(n))$.

☞ Constants do not count for space as well.

Learning Objectives

- ➤ A deeper understanding why (k-string) Turing machines make a reasonable model of computation.
- ➤ You should know how time/space complexity classes are derived using bounds on computations.
- ➤ The idea that multiplicative/additive constants do not count.
- ➤ The definitions and background of complexity classes $\mathbf{P}$ and $\mathbf{L}$.