Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 2: Turing Machines

Aalto University
School of Science
Department of Computer Science

Spring 2017

Agenda

- Basic definitions of Turing Machines
- 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.
Noppa pages of T-79.1001, including the “Additional Reading” section

Turing Machines

- To discuss what can, and in particular what *cannot* be done by algorithms, one needs a mathematically precise definition of an algorithm. *Turing machines* are the standard choice.
- Turing machines look complicated, but are in fact really simple.
- They can simulate arbitrary algorithms with inconsequential loss of efficiency using an elementary data structure: a string of symbols.
- One may view the Turing machine formalism as a mathematically precise *assembly language* for a curious computer architecture. Thus, each Turing machine is analogous to an assembly language program.

1. Basic Definitions

Definition

A Turing machine is a quadruple $M = (K, \Sigma, \delta, s)$ where

- K is a finite set of *states* and $s \in K$ is a designated *initial state*,
- Σ is a finite set of symbols (the *alphabet* of M) so that $\sqcup, \triangleright \in \Sigma$,
- δ is a *transition function*:

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

where the *halting state* h, the *accepting state* “yes”, and the *rejecting state* “no” are not in K,
and the symbols $\to$ (right), $\leftarrow$ (left), and $-$ (stay) indicate *cursor directions*.

Transition functions

- The transition 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.
- It is required that $\triangleright$ always directs the cursor to the right and is never erased, i.e. formally:
  if for any states p and q, $\delta(q, \triangleright) = (p, \rho, D)$, then $\rho = \triangleright$ 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).

Starting and halting

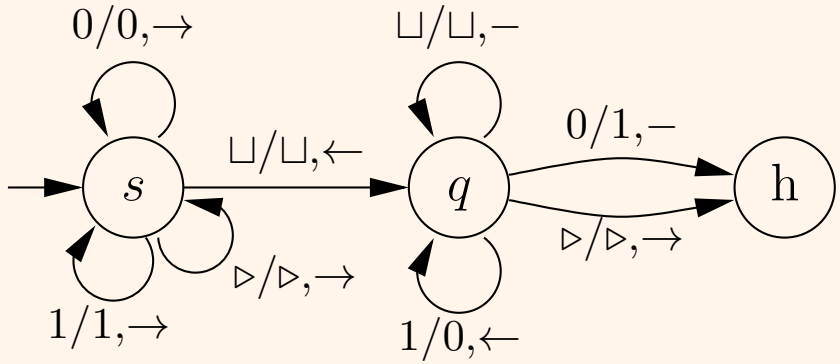
- The program starts with
  (i) initial state s,
  (ii) the string initialised to $\triangleright x$ where x is a finitely long string in $(\Sigma - \{\sqcup\})^*$ (x is the *input* of the machine) and
  (iii) the cursor pointing to $\triangleright$.
- A machine has *halted* if one of the three halting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

Example

Consider a Turing machine $M_{inc} = (K, \Sigma, \delta, s)$ with $K = \{s, q\}$, $\Sigma = \{0, 1, \sqcup, \triangleright\}$ and the following transition function 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	$\triangleright$	$(h, \triangleright, \rightarrow)$

Graphical presentation:



The machine is designed to compute the successor $n + 1$ of a natural number n given *in binary*.[a]

[a] Actually, the machine design is not quite correct. Can you spot the error?

Operational semantics

- A *configuration* is a triple (q, w, u), where
  - $q \in K$ is the current state,
  - $w \in \Sigma^+$ is the string to the left of the cursor, including the symbol scanned by the cursor, and
  - $u \in \Sigma^*$ is the string to the right of the cursor.

Example

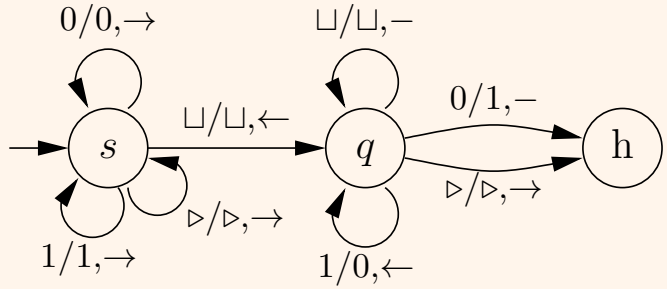
Consider the Turing machine M_{inc} introduced above. For the input "01", the initial configuration of the machine is $(s, \triangleright, 01)$.

- 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 instance, 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 symbol removed (or empty, if u is empty).

Example

Recall the Turing machine M_{inc}:



The initial configuration $(s, \triangleright, 01)$ yields in one step of computation the configuration $(s, \triangleright 0, 1)$, i.e.,

$$(s, \triangleright, 01) \overset{M_{inc}}{\rightarrow} (s, \triangleright 0, 1)$$

Then

$(s, \triangleright 0, 1) \overset{M_{inc}}{\rightarrow} (s, \triangleright 01, \varepsilon)$

$(s, \triangleright 01, \varepsilon) \overset{M_{inc}}{\rightarrow} (s, \triangleright 01\sqcup, \varepsilon)$

$(s, \triangleright 01\sqcup, \varepsilon) \overset{M_{inc}}{\rightarrow} (q, \triangleright 01, \sqcup)$

$(q, \triangleright 01, \sqcup) \overset{M_{inc}}{\rightarrow} (q, \triangleright 0, 0\sqcup)$

$(q, \triangleright 0, 0\sqcup) \overset{M_{inc}}{\rightarrow} (h, \triangleright 1, 0\sqcup)$

Hence, $M_{inc}(01) = 10$

Informal graphical notation:

$s, \triangleright \underline{0}1 \overset{M_{inc}}{\rightarrow} s, \triangleright 0\underline{1}$

$s, \triangleright 0\underline{1} \overset{M_{inc}}{\rightarrow} s, \triangleright 01\underline{\sqcup}$

$s, \triangleright 01\underline{\sqcup} \overset{M_{inc}}{\rightarrow} q, \triangleright 0\underline{1}\sqcup$

$q, \triangleright 0\underline{1}\sqcup \overset{M_{inc}}{\rightarrow} q, \triangleright \underline{0}0\sqcup$

$q, \triangleright \underline{0}0\sqcup \overset{M_{inc}}{\rightarrow} h, \triangleright \underline{1}0\sqcup$

Configurations reached in several steps

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

Example

$(s,\triangleright,01) \overset{M_{inc}}{\to}^{0} (s,\triangleright,01)$ and $(s,\triangleright,01) \overset{M_{inc}}{\to}^{3} (s,\triangleright 01\sqcup,\varepsilon)$

- The relation *yields:* $(q,w,u) \overset{M}{\to}^{*} (q',w',u')$ if there is some $k \geq 0$ such that $(q,w,u) \overset{M}{\to}^{k} (q',w',u')$.
  That is, $\overset{M}{\to}^{*}$ is the transitive and reflexive closure of $\overset{M}{\to}$.

Example

$(s,\triangleright,01) \overset{M_{inc}}{\to}^{*} (s,\triangleright 01\sqcup,\varepsilon)$, $(s,\triangleright,01) \overset{M_{inc}}{\to}^{*} (h,\triangleright 1,1\sqcup)$, and
$(s,\triangleright,111) \overset{M_{inc}}{\to}^{*} (h,\triangleright,000\sqcup)$

2. Turing Machines as Algorithms

Turing machines are natural devices for solving problems on strings:

- Let $L \subseteq (\Sigma - \{\sqcup\})^*$ be a language.
  A Turing machine M *decides* L if 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 *decidable* or *recursive* language.
- A Turing machine M *computes* a (string) function
  $f : (\Sigma - \{\sqcup\})^* \to \Sigma^*$ if for every string $x \in (\Sigma - \{\sqcup\})^*$,
  $M(x) = f(x)$.
- If such an M exists, f is called a *recursive function*.

☞ *The term "recursive" refers historically to Kurt Gödel's formulation of computation in terms of recursively defined integer functions. The formalism was shown to be equivalent to Turing machines, and the term has extended its meaning also to this context.*

Example

Consider a Turing machine $M = (K, \Sigma, \delta, s)$ deciding the language $L = \{x \in \{0,1\}^* \mid$ the number of symbols "1" in x is even $\}$ where $K = \{s,t\}$, $\Sigma = \{0,1,\sqcup,\triangleright\}$ and the transition function δ:

$p \in K$	$\sigma \in \Sigma$	$\delta(p,\sigma)$
s,	$\triangleright$	$(s,\triangleright,\rightarrow)$
s,	0	$(s,0,\rightarrow)$
s,	1	$(t,1,\rightarrow)$
s,	$\sqcup$	("yes", $\sqcup$, $-$)

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

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

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

Recursively enumerable languages

- A Turing machine M *accepts* or *semidecides* L if 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 *semidecidable* or *recursively enumerable* language.
- We will later encounter examples of r.e. languages.

Proposition

If L is recursive, then it is recursively enumerable.

☞ *Historically, a "recursively enumerable set" (or language) meant a set which has a recursive (in the sense of Gödel) enumerating function. This concept was shown to be equivalent to semidecidability by Turing machines, and now the two terms are used interchangeably.*

Solving problems using Turing machines

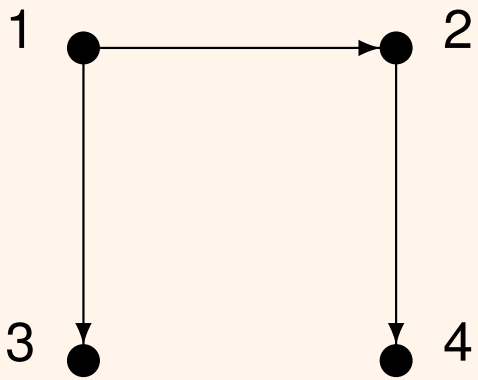
- Instances of a problem are encoded into strings.
- Solving a decision problem amounts to deciding the language that consists of the “yes” instances of the problem.
- An optimisation problem is solved by a Turing machine that computes the appropriate function from strings to strings (where the output 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, then representation B of the same instance has length at most $p(n)$ for some polynomial p.
- An 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 (or "tapes") and associated cursors are more convenient from the programmer's point of view.
- They can be simulated by ordinary Turing machines with an inconsequential loss of efficiency.
- A *k-string Turing machine*, for some integer $k \geq 1$, is a quadruple $M = (K, \Sigma, \delta, s)$ where the transition function δ has been generalised to handle *k strings simultaneously*:

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

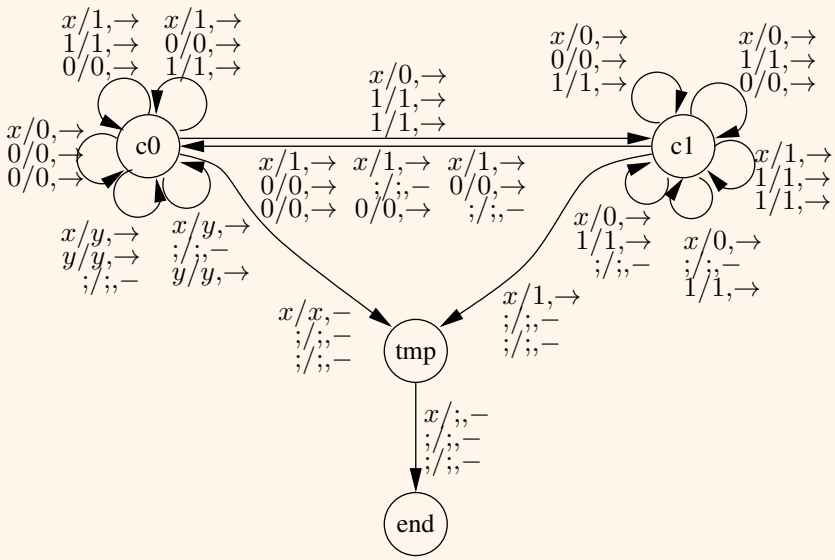
- This definition yields an ordinary Turing machine when $k = 1$.

Generalised 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.

Example

- Compute the sum of binary numbers on strings 1 and 2 to string 3
- Numbers are presented in “least significant bit first” order and terminated with a semicolon
- $\Sigma = \{0, 1, ;, \sqcup, \triangleright\}$, $x \in \Sigma$ and $y \in \{0, 1\}$.
  Parameterised edges represent multiple transitions



$$\left(\begin{array}{ll} \text{c0}, & \triangleright\underline{\sqcup} \\ & \triangleright\underline{0}1; \\ & \triangleright\underline{1}11; \end{array}\right) \xrightarrow{M} \left(\begin{array}{ll} \text{c0}, & \triangleright 1\underline{\sqcup} \\ & \triangleright\underline{0}1; \\ & \triangleright 1\underline{1}1; \end{array}\right) \xrightarrow{M}$$

$$\left(\begin{array}{ll} \text{c1}, & \triangleright 10\underline{\sqcup} \\ & \triangleright\underline{0}1; \\ & \triangleright 11\underline{1}; \end{array}\right) \xrightarrow{M} \left(\begin{array}{ll} \text{c1}, & \triangleright 100\underline{\sqcup} \\ & \triangleright\underline{0}1; \\ & \triangleright 11\underline{1}; \end{array}\right) \xrightarrow{M}$$

$$\left(\begin{array}{ll} \text{tmp}, & \triangleright 1001\underline{\sqcup} \\ & \triangleright\underline{0}1; \\ & \triangleright 11\underline{1}; \end{array}\right) \xrightarrow{M} \left(\begin{array}{ll} \text{end}, & \triangleright 1001; \\ & \triangleright\underline{0}1; \\ & \triangleright 11\underline{1}; \end{array}\right)$$

- *Output* is defined as for ordinary machines:
  - if $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \overset{M}{\to}^{*} (\text{“yes”}, w_1, u_1, \ldots, w_k, u_k)$, then $M(x) = \text{“yes”}$
  - if $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \overset{M}{\to}^{*} (\text{“no”}, w_1, u_1, \ldots, w_k, u_k)$, then $M(x) = \text{“no”}$.
  - if $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \overset{M}{\to}^{*} (\mathrm{h}, w_1, u_1, \ldots, w_k, u_k)$, then $M(x) = y$ where y is $w_k u_k$ with the leading $\triangleright$ and trailing $\sqcup$s removed (i.e., *output* is read from the *last (kth) string*.)
- The *time required* by M on input x is t if

$$(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \overset{M}{\to}^{t} (H, w_1, u_1, \ldots, w_k, u_k)$$

where $H \in \{\mathrm{h}, \text{“yes”}, \text{“no”}\}$.
If $M(x) = \nearrow$, then the time required is thought to be ∞.

Complexity classes

- Turing machine performance is measured by the amount of time (or space) required on instances of size n, as 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 $f(|x|)$ where $|x|$ is the size of the input x.
- In this case we also say that $f(n)$ is an *(upper) time bound* for M, and that the language L decided by M belongs to the *time complexity class* $\mathbf{TIME}(f(n))$.
- Notice that time bounds are determined by the *worst-case inputs* for each input size n.
- To recap the key definition:

$$\mathbf{TIME}(f(n)) = \{L : L \text{ is decided by some multistring Turing machine operating within time } f(n)\}.$$

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 $\Sigma' = \Sigma \cup \underline{\Sigma} \cup \{\triangleright', \underline{\triangleright'}, \triangleleft\}$.
- 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: $(s, \triangleright, \underline{\triangleright'} x \triangleleft \underline{\triangleright'} \triangleleft \ldots \underline{\triangleright'} \triangleleft \triangleleft)$

Proof cont'd

- The simulation of a step of M by M' takes place in two passes (sweeps) across the whole string of M' as follows:

  1st pass: gather information on the symbols underlined (scanned) on the k strings;

  2nd pass: change the underlined (scanned) symbols and move the simulated cursors (underlines) appropriately.

- The strings of M have a total length of $O(kf(n))$.
  To simulate one step of M, M' needs $O(kf(n))$ steps for the 1st pass and $O(k^2 f(n))$ steps for the 2nd pass. (The latter can also be reduced to $O(kf(n))$ by clever programming.)
- Since M makes at most $f(n)$ steps, M' makes $O(f(n)^2)$ steps. (Note that k is fixed and independent of x.)

☞ *No known "realistic" improvement on the Turing machine model will extend the family of languages such machines decide, or will affect their speed more than polynomially.*

4. Linear Speedup

- When using Turing machines, the rates of growth of the time/space requirements are important but the precise multiplicative and additive constants are not.
- This "linear speedup" of TM computations is achieved by simply using a bigger alphabet to encode blocks of symbols, and making transitions to cover a whole block at a time.
- Hence this is just a technical trick, but a very convenient one.

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 increasing the "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 set $m = \lceil 6/\varepsilon \rceil$ and use m-tuples of symbols of M as symbols in M'.
- The term $(n+2)$ in the time bound for M' arises from the time needed to compress the input string into the m-tuple encoding.

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 desired speedup is obtained as $m = \lceil 6/\varepsilon \rceil$.
- Notice that a lot of new states have to be added: $|K| * m^k |\Sigma|^{3mk}$.

Consequences of the linear speedup theorem

- In the linear speedup theorem we can achieve:
  (i) if $f(n) = \Theta(n)$, then $f'(n) \leq (1+\varepsilon)n+2$ for any $\varepsilon > 0$,
  (ii) if $f(n) = \Theta(g(n))$ for some superlinear $g(n)$, then $f'(n) \leq g(n)$,
  (iii) in particular if $f(n) = cn^k + o(n^k)$ for some $c > 0$ and $k > 1$, then $f'(n) \leq n^k$.
- *Corollary.* If L is polynomial-time decidable, then $L \in \mathbf{TIME}(n^k)$ for some integer $k > 0$.

Definition (The class $\mathbf{P}$)

The set of all languages decidable by (deterministic!) Turing machines in polynomial time is defined as:

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

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 used by a computation.
- However because the input string is included, sublinear space bounds cannot be discussed using this definition.
- Hence we exclude the effects of reading the input and writing the output as regards the usage 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 transition function 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 usage

- Suppose that for a k-string Turing machine M and an input x,

$$(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \overset{M}{\rightarrow}^{*} (H, w_1, u_1, \ldots, w_k, u_k)$$

  where $H \in \{\text{“yes”}, \text{“no”}, \mathrm{h}\}$ is a halting state.
- Then the space used by M on input x is $\Sigma_{i=1}^{k} |w_i u_i|$.
- If M is a Turing machine *with input and output*, then the space used by M on input x is $\Sigma_{i=2}^{k-1} |w_i u_i|$.
- Let $f : \mathbf{N} \mapsto \mathbf{R}^{+}$. Turing machine M *operates within space bound* $f(n)$ if for any input x, M uses space at most $f(|x|)$.

Space complexity classes

Definition

The *space complexity class* $\mathbf{SPACE}(f(n))$ comprises the family of languages L that can be decided by Turing machines with input and output operating within space bound $f(n)$.

Definition (The class $\mathbf{L}$)

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

An analog for the linear speedup theorem can be shown for space, too, by increasing the "word length":

Theorem

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

☞ Constants do not count for space either.

Learning Objectives

- Understanding why (k-string) Turing machines make a reasonable model of computation.
- Defining time/space complexity classes according to resource bounds on computations.
- The idea that multiplicative/additive constants are not relevant in Turing machine complexity.
- The definitions and background of complexity classes $\mathbf{P}$ and $\mathbf{L}$.