Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 3: More about Turing Machines

Aalto University
School of Science
Department of Computer Science

Spring 2017

Agenda

- Nondeterministic Turing machines
- Turing machines and random access machines

(C. Papadimitriou: *Computational Complexity*, Chapters 2.6–2.7)

1. Nondeterministic Turing Machines

- A nondeterministic Turing machine may have several alternative transitions at each configuration. The machine accepts its input if *some* sequence of choices leads to acceptance.
- This is an unrealistic model of actual computation, but a very convenient mathematical concept, and also provides a formulation for certain types of computational *problems*. (The problem class $\mathbf{NP}$.)
- Nondeterministic TMs can be simulated by deterministic TMs with an exponential loss of efficiency.
- Open question: is there a polynomial-time simulation method? (Is $\mathbf{P} = \mathbf{NP}$?)

Transition relation

Definition

A nondeterministic Turing machine (NTM) is a quadruple $N = (K, \Sigma, \Delta, s)$ like the ordinary Turing machine except that Δ is a *transition relation* (rather than a transition function):

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

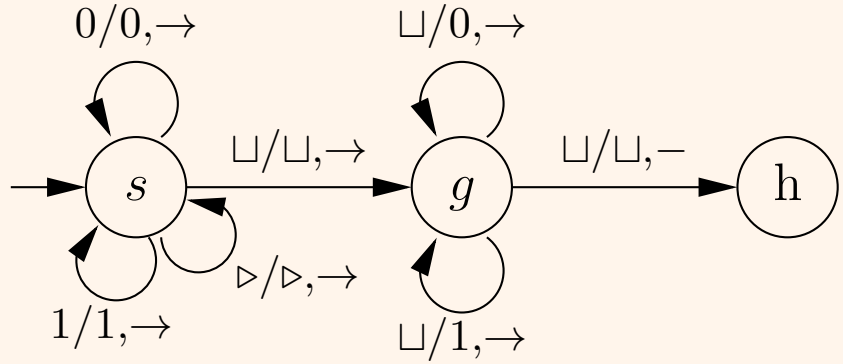
- Configurations are defined as before but “yields” is a relation (rather than a function) for a NTM N: $(q, w, u) \overset{N}{\rightarrow} (q', w', u')$ if there is a tuple in Δ that makes this a legal transition.

Example

Consider the nondeterministic Turing machine $N_g = (K, \Sigma, \Delta, s)$ which generates an arbitrary binary sequence after its input where $K = \{s, g\}$, $\Sigma = \{0, 1, \sqcup, \triangleright\}$ and the transition relation Δ:

p	σ	p'	σ'	D
s	0	s	0	$\rightarrow$
s	1	s	1	$\rightarrow$
s	$\sqcup$	g	$\sqcup$	$\rightarrow$
s	$\triangleright$	s	$\triangleright$	$\rightarrow$
g	$\sqcup$	g	1	$\rightarrow$
g	$\sqcup$	g	0	$\rightarrow$
g	$\sqcup$	h	$\sqcup$	$-$

graphical representation:



Given input “1”, one possible computation is:

$$(s, \triangleright, 1) \overset{N_g}{\rightarrow} (s, \triangleright 1, \varepsilon) \overset{N_g}{\rightarrow} (s, \triangleright 1 \sqcup, \varepsilon) \overset{N_g}{\rightarrow} (g, \triangleright 1 \sqcup \sqcup, \varepsilon) \overset{N_g}{\rightarrow} (h, \triangleright 1 \sqcup \sqcup, \varepsilon)$$

Notice that:

$(g, \triangleright 1 \sqcup \sqcup, \varepsilon) \xrightarrow{N_g} (g, \triangleright 1 \sqcup 1 \sqcup, \varepsilon)$

$(g, \triangleright 1 \sqcup \sqcup, \varepsilon) \xrightarrow{N_g} (g, \triangleright 1 \sqcup 0 \sqcup, \varepsilon)$

Hence,

- $(g, \triangleright 1 \sqcup \sqcup, \varepsilon) \xrightarrow{N_g}{}^{*} (h, \triangleright 1 \sqcup 000000 \sqcup, \varepsilon)$
- $(g, \triangleright 1 \sqcup \sqcup, \varepsilon) \xrightarrow{N_g}{}^{*} (h, \triangleright 1 \sqcup 10101010101 \sqcup, \varepsilon)$
- $\ldots$
- $(g, \triangleright 1 \sqcup \sqcup, \varepsilon) \xrightarrow{N_g}{}^{*} (h, \triangleright 1 \sqcup 11111111111111111111 \sqcup, \varepsilon)$
- $\ldots$

In fact, for any finite sequence B of symbols $0, 1$:

$(g, \triangleright 1 \sqcup \sqcup, \varepsilon) \xrightarrow{N_g}{}^{*} (h, \triangleright 1 \sqcup B \sqcup, \varepsilon)$

and hence, $(s, \triangleright, 1) \xrightarrow{N_g}{}^{*} (h, \triangleright 1 \sqcup B \sqcup, \varepsilon)$

NTMs deciding languages

The power of nondeterminism is characterised by the nature of the input-output relation induced by NTMs.

Definition

A nondeterministic Turing machine N decides a language L if for any $x \in \Sigma^*$, the following holds:

$$x \in L \text{ iff } (s, \triangleright, x) \overset{N}{\rightarrow}{}^{*} (\text{“yes”}, w, u) \text{ for some strings } w \text{ and } u.$$

Thus:

(i) An input is accepted if there is some sequence of nondeterministic choices that results in the accepting state “yes”.

(ii) The input is rejected if no sequence of nondeterministic choices leads to acceptance.

Example

Consider the language TSP(D) consisting of pairs (M, B) such that M is an $n \times n$ distance matrix and B is an integer and there is a tour of n cities whose length is at most B.

The following nondeterministic Turing machine N decides TSP(D):

- N generates after its input x nondeterministically an arbitrary sequence of $0, 1, /$ no longer than the input (cf. the machine N_g).
- After this, N operates completely deterministically. It first checks whether the input x is well-formed, i.e., consists of a pair (M, B) with M an $n \times n$ distance matrix and B a binary number, and whether the generated sequence is of the form $/c_1/c_2/\ldots/c_n/$ where each c_j is a binary number.
- If these checks fail, N halts in the rejecting state; otherwise it continues to determine whether the generated sequence $/c_1/c_2/\ldots/c_n/$ presents a tour of the cities of length at most B.
- If this is the case, N halts in the accepting state, otherwise in the rejecting state.

- Notice that checks whether the input and the generated sequence are well-formed can be done in a polynomial number of steps.
- Similarly, it can be determined whether the generated sequence presents a tour of the cities of length at most B using a polynomial number of steps.
- Hence, each computation of N halts in the accepting or rejecting state after a polynomial number of steps.
- Moreover, N decides nondeterministically TSP(D):
  If $x \in$ TSP(D), then the input x is of the form (M, B), and N has a computation where a sequence presenting a tour of the cities of length at most B is generated, and then after the checks N halts in the accepting state.
  If $x \notin$ TSP(D), N halts in the rejecting state no matter what the generated sequence is, because one of the checks fails.
  Hence, $x \in$ TSP(D) iff $(s, \triangleright, x) \overset{N}{\rightarrow}{}^{*}$ ("yes", w, u) for some w and u.

Nondeterministic time complexity classes

Definition

A nondeterministic Turing machine N decides a language L in time $f(n)$ if N decides L and for any $x \in \Sigma^*$,

$$\text{if } (s, \triangleright, x) \overset{N}{\to}{}^{k} (q, w, u), \text{ then } k \leq f(|x|).$$

☞ All computation paths should have length at most $f(|x|)$.

Definition

The time complexity class $\mathbf{NTIME}(f(n))$ comprises the family of languages L that can be decided by nondeterministic Turing machines in time $f(n)$.

Nondeterministic time complexity classes—cont'd

Definition

The set $\mathbf{NP}$ of all languages decidable by nondeterministic Turing machines in polynomial time is defined as:

$$\mathbf{NP} = \bigcup_{k>0} \mathbf{NTIME}(n^k)$$

☞ Note that $\mathbf{P} \subseteq \mathbf{NP}$, as TMs are also NTMs.

Example

Consider the nondeterministic machine N introduced above for TSP(D): (i) it decides TSP(D) and (ii) there is some k such that for each input x every computation path is of length at most $c|x|^k$.

☞ TSP(D) $\in \mathbf{NP}$.

Nondeterministic time complexity classes—cont'd

- Can nondeterministic Turing machines be simulated by deterministic Turing machines?
- Answer: yes they can, but in general only exponential-time simulations are known for turning a nondeterministic machine to a deterministic one!

Theorem

Suppose that a language L is decided by an NTM N in time $f(n)$. Then it is decided by a 3-string deterministic TM M in time $O(c^{f(n)})$, where $c > 1$ is some constant depending on N. Thus,

$$\mathbf{NTIME}(f(n)) \subseteq \bigcup_{c>1} \mathbf{TIME}(c^{f(n)}).$$

Proof sketch

- Let $N = (K, \Sigma, \Delta, s)$ be a NTM.
- Let d be the degree of nondeterminism (maximal number of possible moves for any state-symbol pair in Δ) for N.
- Any computation of N is a sequence of nondeterministic choices. A sequence of t choices is a sequence of t integers from $0, 1, \ldots, d-1$.
- The simulating machine M considers all such sequences of choices *in order of increasing length* and simulates N on each.
- While considering a sequence $(c_1, c_2, \ldots, c_t)$, M maintains the sequence as its second string.
- With sequence $(c_1, c_2, \ldots, c_t)$ M simulates the actions that N would have taken if N had taken choice c_i at step i for its first t steps.

Proof sketch—cont'd

- If a sequence leads N to halting with "yes", then M does, too. Otherwise it considers the next sequence.
- Generating the next sequence (on the second string) is like calculating the next integer in base d.
- When should M reject?
  (Note that the bound $f(n)$ is not available to M!)
- Machine M rejects its input whenever it has simulated all sequences of choices of length t and finds among them *no continuing computation*.
- The time bound $O(c^{f(n)})$ is then established as the product of
  – the number of sequences $\Sigma_{t=1}^{f(n)} d^t = O(d^{f(n)+1})$ and
  – the cost of generating and simulating each sequence which is clearly below $O(2^{f(n)})$.

Space complexity

- For discussing space complexity, a nondeterministic Turing machine model with input and output is needed.
- Given a k-string NTM N with input and output, we say that N decides language L *within space* $f(n)$ if N decides L and for any $x \in (\Sigma - \{\sqcup\})^*$, if $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon,) \xrightarrow{N}{}^{*} (q, w_1, u_1, \ldots, w_k, u_k)$, then $\Sigma_{i=2}^{k-1} |w_i u_i| \leq f(|x|)$.

Example

- The language of REACHABILITY consists of triples (G, v, u) where G is a graph (say given in the form (n, M) where n is the number of vertices in binary and M the adjacency matrix) and v and u are two binary numbers such that there is a path from the vertex number v to the vertex number u in the graph G.
- REACHABILITY is nondeterministically decidable within space $O(\log n)$
- A nondeterministic Turing machine N deciding REACHABILITY nondeterministically within space $O(\log n)$ uses two strings S_1 and S_2 besides the input string and works as follows:

- The nondeterministic Turing machine N:
  1. When starting N writes v to S_1 and nondeterministically a binary number having at most as many bits as n to string S_2.
  2. If S_1 and S_2 are legal vertex numbers and there is an edge from the vertex in S_1 to the vertex in S_2 in G, then N goes to step 3 otherwise it rejects.
  3. If S_2 contains the vertex u, the machine accepts else it moves the vertex in S_2 to S_1, generates nondeterministically a binary number having at most as many bits as n to string S_2 and goes back to step 2.
- N works within $2 \times \lceil \log n \rceil$ space
- N decides REACHABILITY because it has an accepting computation exactly when there is a simple path from v to u in G.

2. Random Access Machines

- Claim: Turing machines can implement arbitrary algorithms, even reasonably efficiently.
- The "Strong Church-Turing thesis":
  "Any reasonable attempt to model mathematically computer algorithms and their time performance ends up with a model of computation and associated time cost that is equivalent to Turing machines within a polynomial."
- Further evidence: Turing machines can simulate *random access machines* (RAMs), which model a simple but typical "standard" computer architecture. The loss of efficiency is only cubic, despite the RAM model operating on arbitrarily large integers in a single step.

Example (for observing the relevance of RAMs)

C-code compiled into i486 architecture assembly language:

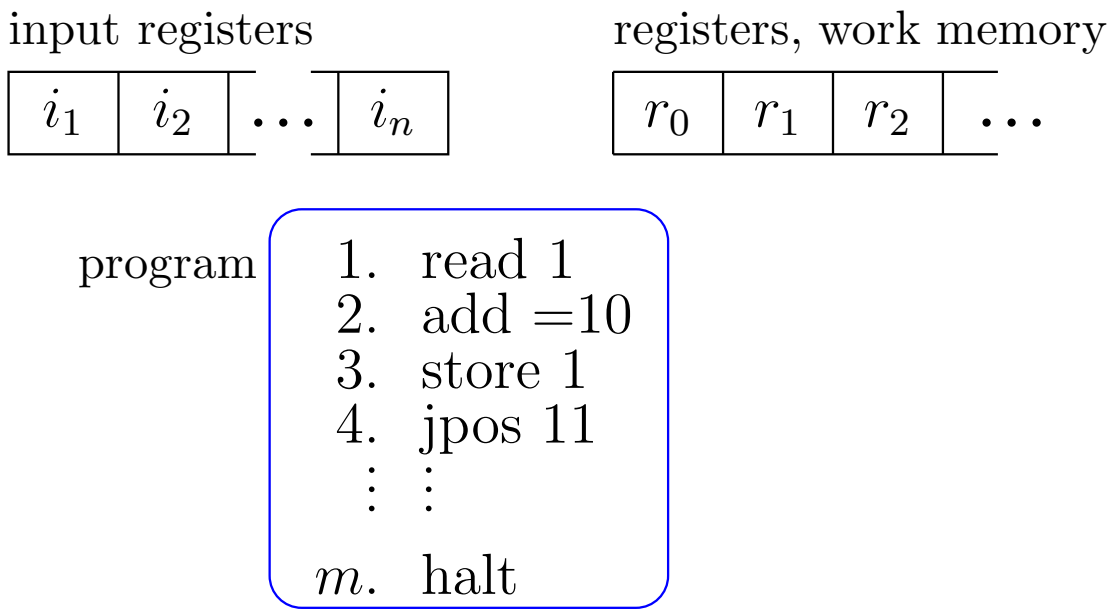
```
typedef struct_intlist {
    int value;
    struct _intlist* next;
} intlist;

int find_smallest(intlist* list) {
    int smallest = INT_MAX;
    while(list != NULL) {
        if(list->value < smallest)
            smallest = list->value;
        list = list->next;
    }
    return smallest;
}
```

```
find_smallest:
        pushl %ebp
        movl %esp, %ebp
        subl $16, %esp
        movl $2147483647, -4(%ebp)
        jmp .L2
.L4:    movl 8(%ebp), %eax
        movl (%eax), %eax
        cmpl -4(%ebp), %eax
        jge .L3
        movl 8(%ebp), %eax
        movl (%eax), %eax
        movl %eax, -4(%ebp)
.L3:    movl 8(%ebp), %eax
        movl 4(%eax), %eax
        movl %eax, 8(%ebp)
.L2:    cmpl $0, 8(%ebp)
        jne .L4
        movl -4(%ebp), %eax
        leave
        ret
```

Basic definitions of RAMs

- Data structure: an array of registers, each capable of containing an arbitrarily large integer, possibly negative.
- A RAM program $\Pi = (\pi_1, \pi_2, \ldots, \pi_m)$ is a finite sequence of *instructions* (of assembler language type).
- Register 0 serves as an accumulator
- Three modes of addressing: 'j', '$\uparrow j$', and '$= j$'
- Input is contained in a finite array of input registers $I = (i_1, \ldots, i_n)$.



Instruction set

Instruction	Op	Semantics
READ	j	$r_0 := i_j$
READ	$\uparrow j$	$r_0 := i_{r_j}$
STORE	j	$r_j := r_0$
STORE	$\uparrow j$	$r_{r_j} := r_0$
LOAD	x	$r_0 := x'$
ADD	x	$r_0 := r_0 + x'$
SUB	x	$r_0 := r_0 - x'$
HALF		$r_0 := \lfloor \frac{r_0}{2} \rfloor$

Instruction	Op	Semantics
JUMP	j	$\kappa := j$
JPOS	j	if $r_0 > 0$, $\kappa := j$
JZERO	j	if $r_0 = 0$, $\kappa := j$
JNEG	j	if $r_0 < 0$, $\kappa := j$
HALT		$\kappa := 0$

where x (resp. x') is one of
'j' / '$\uparrow j$' / '$= j$'
(resp. r_j / r_{r_j} / j)

Operational semantics

- A configuration is a pair $C = (\kappa, R)$ where κ is the number of the instruction to be executed and $R = \{(j_1, r_{j_1}), (j_2, r_{j_2}), \ldots, (j_k, r_{j_k})\}$ is a finite set of register-value pairs.
  The initial configuration: $(1, \emptyset)$.
- For a RAM program Π and an input $I = (i_1, \ldots, i_n)$, the relation $(\kappa, R) \overset{\Pi, I}{\to} (\kappa', R')$ (yields in one step) is defined as follows:
  – κ' is the new value of κ after executing the κth instruction of Π,
  – R' is R with possibly some pair (j, x) deleted and (j', x') added according to the κth instruction of Π.
- The relation $\overset{\Pi, I}{\to}$ induces $\overset{\Pi, I^k}{\to}$ and $\overset{\Pi, I^*}{\to}$ as previously.

Definition

Let D be a set of finite sequences of integers. A RAM Π *computes* $\phi : D \to \mathbf{Z}$ if for each $I \in D$, $(1, \emptyset) \overset{\Pi, I^*}{\to} (0, R)$ so that $(0, \phi(I)) \in R$.

Example

A RAM program computing $\phi(x,y) = |x-y|$

Input:

$I = (6,10)$

$\phi(I) = 4$

Program:

1. READ 2
2. STORE 2
3. READ 1
4. STORE 1
5. SUB 2
6. JNEG 8
7. HALT
8. LOAD 2
9. SUB 1
10. HALT

Configurations:

(1, {})
(2, {(0,10)})
(3, {(0,10), (2,10)})
(4, {(0,6), (2,10)})
(5, {(0,6), (2,10), (1,6)})
(6, {(0,-4), (2,10), (1,6)})

(8, {(0,-4), (2,10), (1,6)})
(9, {(0,10), (2,10), (1,6)})
(10, {(0,4), (2,10), (1,6)})
(0, {(0,4), (2,10), (1,6)})

Counting time and space

- The execution of each RAM instruction counts as one time step.
  - Addition of large integers takes place in one step.
  - Multiplication is *not included* in the instruction set.
- The size of the input is computed in terms of logarithms:
  - For an integer i, $\mathrm{b}(i)$ is its binary representation with no redundant leading 0s and with a minus sign in front if negative.
  - The length of integer i, $l(i) = |\mathrm{b}(i)|$.
  - For a sequence of integers $I = (i_1, \ldots, i_n)$, $l(I) = \Sigma_{j=1}^{n} l(i_j)$.

Time bounds for RAMs

Definition

Suppose that a RAM program Π computes a function $\phi : D \to \mathbf{Z}$ and let $f : \mathbf{N}^+ \to \mathbf{N}^+$.

The program Π computes ϕ *in time* $f(n)$ if

$$\text{for any } I \in D,\ (1,\emptyset) \overset{\Pi,I^k}{\to} (0,R) \text{ so that } k \leq f(l(I)).$$

Example

The multiplication of arbitrarily large integers is accomplished by a RAM in linear number of steps (i.e., the number of steps is proportional to the logarithm of the input integers).

☞ RAM programs are powerful.

Example

A RAM for solving REACHABILITY can be found in the textbook.

Simulating TMs with RAMs

- The simulation of a Turing machine having an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is possible with a linear loss of efficiency.
- The domain of inputs for the simulating RAM is $D_\Sigma = \{(i_1, \ldots, i_n, 0) \mid n \geq 0,\ 1 \leq i_j \leq k,\ j = 1, \ldots, n\}$.
- For a language $L \subseteq (\Sigma - \{\sqcup\})^*$, define $\phi_L : D_\Sigma \longrightarrow \{0,1\}$ so that

$$\phi_L(i_1, \ldots, i_n, 0) = 1 \text{ iff } \sigma_{i_1} \cdots \sigma_{i_n} \in L.$$

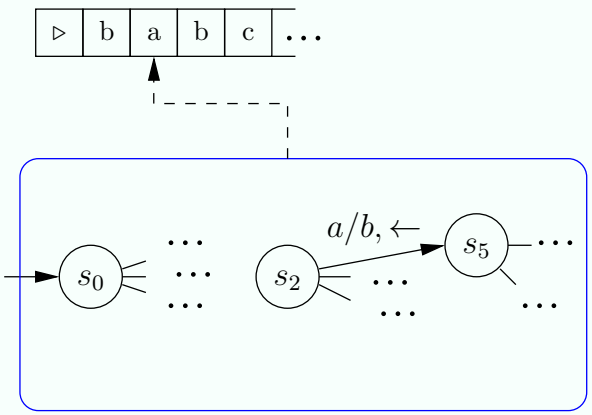
☞ Deciding L is equivalent to computing ϕ_L.

Theorem

Let $L \in \mathbf{TIME}(f(n))$. Then there is a RAM program which computes the function ϕ_L in time $O(f(n))$.

Proof sketch

Given a Turing machine M deciding L construct for each state of M a subroutine (a RAM program) which simulates the state.



input registers:
original input

$\hat{a}$	$\hat{b}$	$\hat{a}$

```
1. copy input to
   work registers
s2, a?: load ↑1
        sub =â
        jzero s2, a
        jump s2, b?
s2, a:  load =b̂
        store ↑1
        load 1
        add =−1
        store 1
        jump s5, ▷?
s2, b?: load ↑1
        :

reject: load =0
        halt
accept: load =1
        halt
```

registers, work memory:
current TM string contents

	6			▷̂	$\hat{b}$	$\hat{a}$	$\hat{b}$	$\hat{c}$	...

r_0 r_4

read/write head pointer

Simulating RAMs with TMs

- Any RAM can be simulated by a Turing machine with only a polynomial loss of efficiency.
- The binary representation of a sequence $I = (i_1, \ldots, i_n)$ of integers, denoted by $\mathrm{b}(I)$, is the string $\mathrm{b}(i_1); \ldots; \mathrm{b}(i_n)$.

Definition

Let D be a set of finite sequences of integers and $\phi : D \to \mathbf{Z}$. A Turing machine M computes ϕ if for any $I \in D$,

$$M(\mathrm{b}(I)) = \mathrm{b}(\phi(I)).$$

Simulating RAMs with TMs

Theorem

If a RAM program computes ϕ in time $f(n)$, then there is a 7-string Turing machine M which computes ϕ in time $O(f(n)^3)$.

Proof sketch

The strings of the machine serve the following purposes:

1. Input
2. Representation of register contents $\ldots; \mathrm{b}(i) : \mathrm{b}(r_i); \ldots \triangleleft$
   (update: erase old value by $\sqcup$s and add new value to the right)
3. Program counter
4. Register address currently sought

5.–7. Extra space reserved for the execution of instructions

Proof sketch—cont'd

- Each instruction of the RAM program Π is implemented by a group of states of M.
- Simulating an instruction of Π on M takes $O(f(n)l)$ steps where l is the size of the largest integer in the registers (as there are $O(f(n))$ pairs on string 2).
- Simulating Π on M takes $O(f(n)^2 l)$ steps.
- It remains to establish that $l = O(f(n))$.

Claim: After the tth step of a RAM program Π computation on input I, the contents of any register have length at most $t + l(I) + l(B)$ where B is the largest integer referred to in an instruction of Π.

Proof sketch—cont'd

Inductive proof of the claim

- Base case: the claim is true when $t = 0$.
- Induction hypothesis: the claim is true after the (t-1)th step.
- Case analysis over instruction types of the tth instruction:
  Most of the instructions do no create new values (jumps, HALT, LOAD, STORE, READ). For these the claim continues to hold directly after the execution of the instruction.
  Consider arithmetic, say ADD, involving two integers i and j.
  The length of the result is one plus the length of the longest operand which is by induction hypothesis at most $t - 1 + l(I) + l(B)$.
  Thus, the result has length at most $t + l(I) + l(B)$.
- Hence, the claim holds.

Learning Objectives

- Basic understanding of differences between deterministic and nondeterministic Turing machines.
- The definitions and background of complexity class $\mathbf{NP}$ and the problem whether $\mathbf{P} = \mathbf{NP}$ or not.
- Understanding why Turing machines constitute a general-purpose and reasonably efficient model of algorithms/computation.