Theoretical Computer Science

Course by Kappel David

Lukas Prokop

2nd march 2012

Contents

1 Intuitive and formal computational theory

First of all, we want to have a little introduction into theoretical computer science by understanding its basic tools and elements:

- Algorithms are performed on primitive, theoretical machines like register machines, turing machines
- We want to analyze algorithms by different parameters such as time and space
- We want to use proofs by reduction to show that two problems are based on the same theoretical problem
- Problems can be categorized into complexity classes such as N and NP and many more
- We want to operate (apply our algorithms) on fundamental structures such as graphs
- Finally we want to be able to write all those properties in a formal language

- We want to discuss those elements machine- and programming language independent.

1.1 The problem of computational complexity

Given is a problem defined by sorting a phone book. A motiviated software developer starts to implement the algorithm in a way, which is intuitive to him. But when he starts to run it, the program keeps running, running and running...

The question to ask is: "Is there any algorithm solving our problem more efficiently?". Obviously, the most intuitive algorithm is not always the most efficient one. In this course we want to discuss and understand tools to make such questions decidable.

1.2 Questions to theoretical computer science

- *What* is a problem?
- What is an *algorithm*?
- *Is there* an algorithm to solve problem XY?
- *Why* are there differences between algorithms and how can they be identified?
- Is the solvable problem *efficiently* solvable?
- How can we define *efficiently*?

1.3 The science of formal computation

1.3.1 Landau notation

$g(n)$ is "upper bound" of $f(n)$:

$$\mathcal{O}(g) = \{f : \mathbb{N} \mapsto \mathbb{R} : \exists c, n_0 : \forall n > n_0 : f(n) \leq c \cdot g(n)\}$$

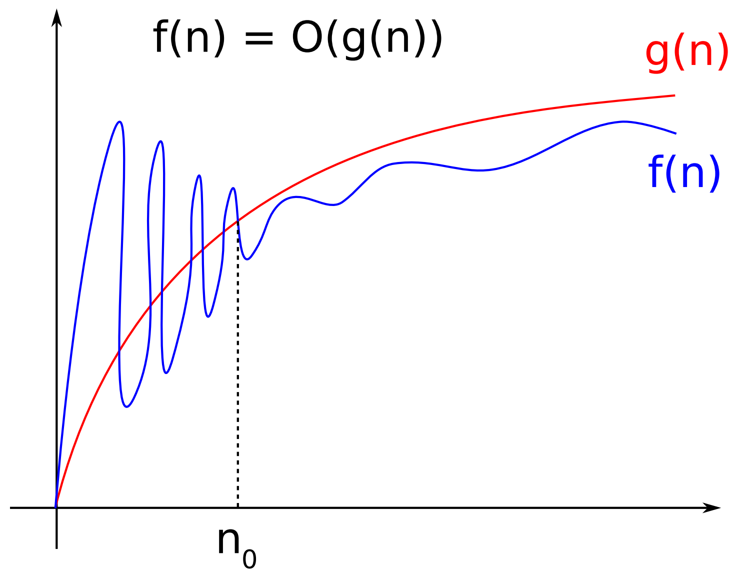
$g(n)$ is "exact bound" of $g(n)$:

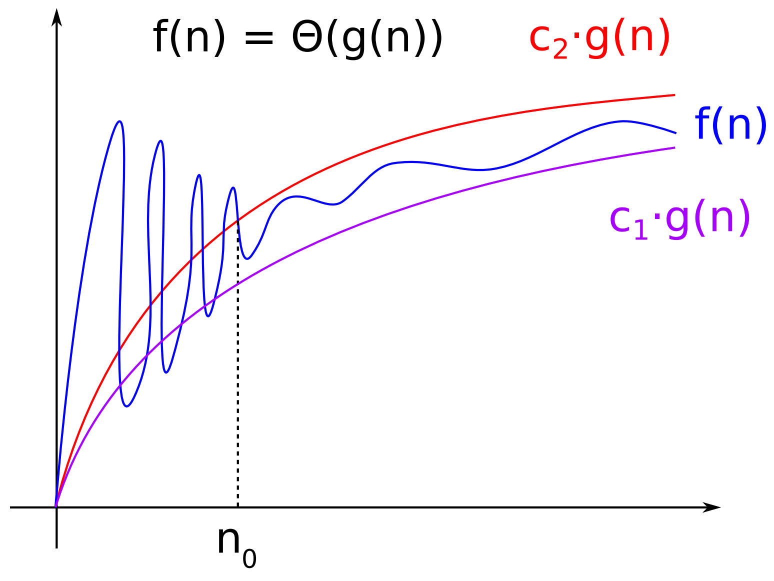
$$\Theta(g) = \{f : \mathbb{N} \mapsto \mathbb{R} : \exists c_1, c_2, n_0 : \forall n > n_0 : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\}$$

Simple form:

$$f(n) = \mathcal{O}(g(n))$$

$$f(n) = \Theta(g(n))$$


$f(n) = O(g(n))$
$g(n)$
$f(n)$
n_0


$f(n) = \Theta(g(n))$
$c_2 \cdot g(n)$
$f(n)$
$c_1 \cdot g(n)$
n_0

1.3.2 Graph theory

A graph G is a tuple of $\{V, E\}$ (vertices, edges). In a representational context, a graph is a set of interconnected objects.

We differ between various graphs by properties:

directed graph If vertex A can be reached by vertex B, it does not imply, that vertex B can be reached by vertex A

simple graph A simple graph is an undirected graph without loops and two vertices cannot be connected by more than one edge (”multiple edge”).

multigraph A multigraph might contain multiple edges and (optionally) loops.

loop An edge starting and ending on the same vertex

path A sequence of edges from vertex A to vertex B (no vertex shall be visited twice in this sequence)

cyclic A path with the starting and ending vertex and the number of visited vertices is greater equal 2

degree of a vertex number of edges incident to this vertex

clique A clique is a set of pairwise adjacent vertices. Therefore all vertices in a clique are connected to each other.

link An edge with different starting and ending vertex (opposite of loop)

All solvable decision problem can be represented by a directed graph. As far as all discussed problems in this course are decision problems, graphs become very important for us. A decision problem is a problem where ”yes” or ”no” are the valid answers.

2 Graph problems

2.1 REACH

Problem 1. *Given is a graph* $G(V, E)$ *with* $n = |V| \geq 2$. *Is there any path from vertex 1 to vertex n?*

We define a recursive algorithm called SEARCH

$\boxed{\text{TODO}}$ draw

1 -¿ 2 2 -¿ 3 3 -¿ 1 3 -¿ 5 4 -¿ 5 4 -¿ 3

Algorithm 1 recursive SEARCH algorithm

```
Require: $G = (V, E), E = 1..n$
  $S = \{1\}$
  $Mark(1)$
  while NOT empty($S$) do
    select $i \in S$ and remove $i$ from $S$
    add $j$ to $S$ and mark $j$
  end while
  if is_marked($n$) then
    return yes
  else
    return no
  end if
```

$$\begin{tikzcd}
 & 1 \arrow[rr] & & 2 \arrow[dl] \\
5 & & 3 \arrow[ul] \arrow[ll] & \\
 & 4 \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

Iteration	$S_i \subseteq V$	$M_i \subseteq V$
1	$S_1 = \{1\}$	$M_1 = \{1\}$
2	$S_2 = \{2, 3\}$	$M_2 = \{1, 2, 3\}$
3	$S_3 = \{3\}$	$M_3 = \{1, 2, 3\}$
4	$S_4 = \{5\}$	$M_4 = \{1, 2, 3, 5\}$
5	$S_5 = \{\}$	$M_5 = \{1, 2, 3, 5\}$

Solution: Yes

2.1.1 Complexity

Best case is $\boxed{\text{TODO}}$ Worst case is $\boxed{\text{TODO}}$

$$T(n) = \mathcal{O}(n^2)$$
$$S(n) = \mathcal{O}(|V|)$$

2.1.2 Proof of correctness

- Loop invariant: We want to define and prove our loop invariant
- transitive closure: Extension of a graph for indirect paths

$$s \to t \Leftrightarrow (s,t) \in E$$

Transitive closure:

$$\begin{aligned} s \xrightarrow[v]{t} t \Leftrightarrow \quad & s \to \quad t \vee \exists h \in V \\ & s \to \quad h \xrightarrow[v]{t} t \end{aligned}$$

Teflexive transitivity

$$s \xrightarrow[v]{*} t \Leftrightarrow s = t \vee s \xrightarrow[v]{t} t$$

Loop invariant of Search $p(i)$

$$1 \xrightarrow{*} n \Leftrightarrow \left[n \in M_i \vee \exists h \in S_i h \xrightarrow[V \backslash M_i]{t} n \right]$$

Proof by induction

Successor of S

$$X = \{t \in V \setminus M_i | s \to t\}$$
$$S_{i+1} = S_i \setminus \{S\} \cup X$$
$$M_{i+1} = M_i \cup X$$

Induction begin

$$i = 1, \quad M_1 = \{1\}, \quad S_1 = \{1\}$$
$$1 \xrightarrow{*} n \Leftrightarrow \left[n = 1 \vee 1 \xrightarrow[V \backslash \{1\}]{t} n \right]$$

Induction step We want to prove that $p(i) \Rightarrow p(i+1)$.

$$\begin{aligned} p(i) \Rightarrow 1 \xrightarrow[v]{*} n \quad \Leftrightarrow \quad & \left[n \in M_i \vee \exists h \in S_i : h \xrightarrow[V \backslash M_i]{t} n \right] \\ & \left[n \in M_i \vee \exists h \in S_i \setminus \{s\} : h \xrightarrow[V \backslash M_i]{t} n \vee s \xrightarrow[V \backslash M_i]{t} n \right] \\ & \left[\ldots \vee \ldots \vee \exists t \in V \setminus M_i : S \to t \xrightarrow[V \backslash M_i]{*} n \right] \\ & \left[\ldots \vee \ldots \vee \exists t \in X : t \xrightarrow[V \backslash M_{i+1}]{*} n \right] \\ & \textbf{Note. } V \setminus \{M_i \cup X\} = V \setminus M_{i+1} \\ & \left[\ldots \vee \ldots \vee n \in X \vee \exists t \in X : t \xrightarrow[V \backslash M_{i+1}]{t} n \right] \end{aligned}$$

Therefore this is our loop invariant.

3 Register machines

3.1 Algorithms

> ... Man soll ein Verfahren angeben, nach welchem sich mittels einer endlichen Anzahl von Operationen entscheiden lässt, ob die Gleichung in ganzen Zahlen lösbar ist.
> —David Hilbert (23 offene Probleme der Mathematik)

> Define a method to decide with a finite number of operations whether or not this equation is solvable with integers.
> —David Hilbert (23 offene Probleme der Mathematik)

When writing programs, we always use similar concepts which represent the fundamental parts of an algorithm:

- operators (eg. arithmetic)
- a finite, ordered number of statements / expressions
- memory access
- conditionals
- jumps
- termination statement

3.2 The idea of RAMs

We define the idea of RAMs, which allow us to execute those algorithms on a machine. The instruction set in a RAM is similar to the common subset of processor architectures.

program A program is a finite list of instances of the instruction set statements.

$$\mathcal{R} = (\pi_1, \dots, \pi_m)$$

assignments Assignments are a finite set of register-value pairs. All registers not listed have a value of 0.

$$R = \left\{(j_1, r_{j_1}), (r_2, r_{j_2}), \dots, (j_l, r_{j_l})\right\}$$

input An input for the register machine is a tuple of k numbers $x_1, x_2, \dots, x_k \in \mathbb{N}_0$

initialization

$$R[x_1, \dots, x_k] = \{(1, x_1), (2, x_2), \dots, (k, x_k)\}$$

configuration The configuration of an RAM is a tuple $\kappa = (b, R)$ with $b \in \mathbb{N}_0$ as next statement to be executed. $R \subset \mathbb{N}_0^2$ is the next register assignment.

configuration relation $\boxed{\text{TODO}}$ longrightarrow Is $\kappa = (b, R)$ and $\kappa' = (b', R')$ then $\kappa \underset{R}{\longrightarrow} \kappa'$ is true, if π_b is a jump statement to b' and $R' = R$ or $b' = b + 1$ and $R' = \{(j, x)\} \cup (R \setminus \{(i, y) \in \mathbb{N}_0^2 \mid i = j\})$ with j as the addressed register of π_b and x as its new computed value.

$\boxed{\text{TODO}}$ missing some content

3.3 Complexity

uniform runtime Each statement takes time of 1. So the total runtime is the number of statements executed.

uniform memory usage The memory usage is the sum of used registers after execution.

The uniform runtime is unsatisfactory for many problems, we are looking at. That's why we want to discuss other machines later on.

Is $x \in \mathbb{N}_0$ then the logarithmic length of x is the number of bits of x in a binary representation.

Definition 1.

$$L(x) = \begin{cases} 1 & x = 0 \\ \lfloor \log x \rfloor + 1 & x \geq 1 \end{cases}$$

$\boxed{\text{TODO}}$ logarithmic time costs

Definition 2. *The logarithmic memory costs of a RAM $\mathcal{R}$ with input $x_1, \dots, x_k$ are defined with*

$$S_{\mathcal{R}}(x_1, \dots, x_k) = \sum_{j=0}^{\infty} \begin{cases} \max_{0 \leq i \leq N} L(r_{i,j}) & \exists 0 \leq i \leq N : r_{i,j} \neq 0 \\ 0 & \forall 0 \leq i \leq N : r_{i,j} = 0 \end{cases}$$

3.4 Another approach to complexity definitions

Definition 3. *Time complexity $T_{\mathcal{R}}(N)$ of a RAM $\mathcal{R}$ is defined in dependency of the logarithmic length n of an input.*

$\boxed{\text{TODO}}$ *subscript*

$$T_{\mathcal{R}}(N) = \max_{\dots} t_{\mathcal{R}}(x_1, \dots, x_k)$$

$\boxed{\text{TODO}}$ space complexity

3.4.1 Polynomial time limitations

$\boxed{\text{TODO}}$ polynomial / polynom? We define a register machine $\mathcal{R}$ to be polynomial runtime-limited if the cost function $T_{\mathcal{R}}$ is limited by a polynomial.

$$T_{\mathcal{R}}(n) = \mathcal{O}(\text{poly}(n))$$

So there is a polynomial with

$$T_{\mathcal{R}}(n) \le p(n) \quad \forall n \in \mathbb{N}$$

A problem is efficiently solvable if there is a polynomial-runtime-limited register machine $\mathcal{R}$ solving the problem.

4 Turing machines

Turing's idea of the turing machine is based on the work of a mathematician: The mathematician takes some input, processes this input according to a program, stores some values on a paper and presents his results. The mathematician is considered to be a finite state machine.

Turing designed a machine, which combines three components: Input, Output and storage are put together on an endless tape, which is capable of read and write operations.

A turing machine is able to determine whether or not it wants to accept a specific input word by entering one of the final states. This is the most important property for the turing machine, because it allows many theoretical scenarios we want to discuss in computation theory.

Definition 4. *A deterministic turing machine $\mathcal{T}$ is a seven-tuple*

$$\mathcal{T} = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$$

- *Q, is a set of states*
- *Σ is the input alphabet, $\square \notin \Sigma$*
- *Γ is the tape alphabet, $\Sigma \subseteq \Gamma$, blank symbol: $\square \in \Gamma$*

- $q_0 \in Q$ *is the begin state*
- $F \subset Q$ *is a set of end states*
- $\square$ *is the blank symbol (= no value)*
- $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\leftarrow, -, \rightarrow\}$, *is the transition function*

Exercise 1. *Write a binary incrementor for a turing machine.*

4.1 Turing computability

Definition 5. *A partial function* $f : \Sigma* \mapsto \Sigma*$ *is turing-computable if and only if*

$$\exists \, \mathrm{DTM}(\mathcal{T}) : f_{\mathcal{T}}(w) = f(w)$$

Exercise 2. *Write a palindrome decider.*

4.2 k-tape DTM on a 1-tape DTM

To simulate a k-tape DTM on a one-tape DTM, we put the tapes onto the single tape sequentially. The tape will be separated by a terminator. We also have to store the position of the cursor by replacing the symbol (the cursor is pointing to) with another special symbol.

$$\Gamma' = \Gamma \cup \underline{\Gamma} \cup \{<, >\}$$
$$|Q'| = |Q|^{\ k} + \ldots$$

5 Multi-tape turing machines

Thesis 1. *Church-turing thesis*

Init: Copy input data to register tape (State Q_0)
Algorithm: Stored in program $Q_1 \ldots Q_p$
Output: Q_{p+1} (store r_0 to output)

Structure of register tape:

$$\#\#\# \underbrace{0}_{\text{key}} \# \underbrace{\text{bin}(r_0)}_{\text{value}} \#\# \underbrace{1}_{\text{key}} \#\# \underbrace{\text{bin}(r_1)}_{\text{value}} \#\# \ldots$$

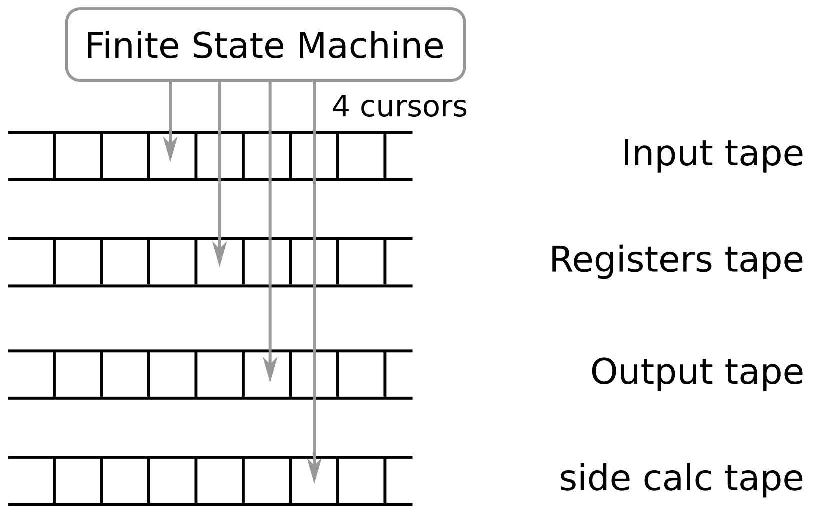


Figure 1: Multitape turing machine

5.1 Instruction *24

1. indirect address
   - Search for r_{24} at register tape
   - Copy r_{24} to tape 4 (accumulator)
2. Search for $r_{r_{24}}$
3. Write $r_{r_{24}}$ to accumulator
4. Search for r_0
5. Add values at accumulator

6 Simulate Turingmachine at register machine

1. Init
   - position = 3 ($r_1 = 3$)
   - state = q_0 ($r_2 = 0$)

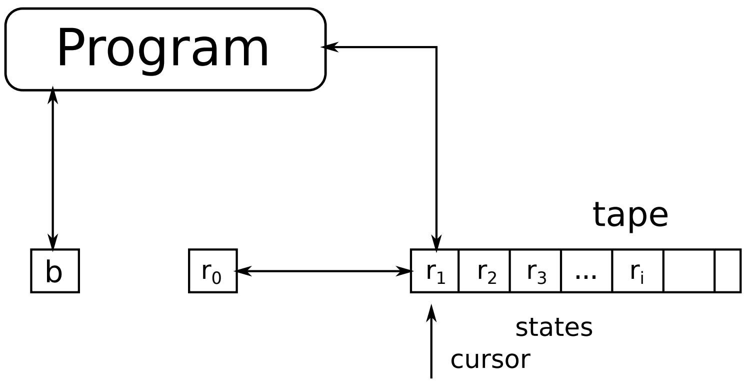


Figure 2: Turingmachine simulated at register machine

2. Loop

```
if state == q_0 and tape_q == b_1:
    state = q  (r_2 = i)
    tape_q = b  (*r_1 = j)
    position = position + 1
if state == ...
    ...
```

7 Another approach to simulate a turing machine at a register machine

Take the architecture from figure 2, but reduce the number of registers to $(\mathfrak{r}_0, \mathfrak{r}_1, \mathfrak{r}_2, \mathfrak{r}_3)$ and define $\mathfrak{r}_2$ und $\mathfrak{r}_3$ to be stacks. The result is a turing-complete process model.

8 Formal languages

Each decision problem can be formulated as word problem of formal languages.

Term. "dt. rekursiver Einschluss"

Type 0:	Turingmachine, REACH	→left/right
Type 1:	Turingmachine, REACH	→left/right
Type 2:	Push-Down-Automaton, PALINDROME	→Push / Pop
Type 3:	Finite state automaton	→Pop

$$\text{Type 0} := \{\text{recursively enumerable grammar}, \text{recursive languages}\}$$

9 Complexity class P

Algorithm. REACH
Algorithm. RELPRIME

Algorithm 2 Euclid(x, y)

```
a ←gcd(x, y)
if a = 1 then
  RELPRIM
else
  return false
end if
```

Algorithm 3 Greatest common divisor(x, y)

```
while $y > 0$ do
  $r \leftarrow x \bmod y$
  $x \leftarrow y$
  $y \leftarrow r$
end while
```

Prove that the number of iterations of the gcd algorithm is limited to $\max(x, y)$.

$$r := x \bmod y \Rightarrow r < y$$

Case 1: $\frac{x}{2} \geq y$

$$r := x \bmod y \Rightarrow r < y < \frac{x}{2} \checkmark$$

Case 2: $\frac{x}{2} < y$

$$r := x \bmod y = x - y < \frac{x}{2} \checkmark$$

Algorithm. CIRCUIT VALUE PROBLEM

10 Non-determinism

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

```
      _ 0,1
     | |  1    0,1    0,1
  -> q_1 -> q_2 -> q_3 -> (q_4)
```

10.1 Travelling Salesman Problem

$$\underbrace{\#\text{wien}\#\text{graz}\#\text{linz}\#}_{\text{cities}}\#\underbrace{\#\text{wien}\#\text{graz}\#101..\#\#..\#}_{\text{costs}}\#\#\underbrace{10010}_{\text{budget}}\#\#\#$$

1. Guess route (non-deterministic)
   - #wien#linz#graz#
   - #graz
   - #linz
   - ...
2. Calculate costs (Copy one city to end of tape 2)
3. Decide: If costs < budget, accept. Reject otherwise.

The complexity of the three steps can be approximated by $\mathcal{O}(n^2)+\mathcal{O}(n^2)+\mathcal{O}(c) = \mathcal{O}(n^2)$.

10.2 Cliquenproblem

Clique is certificate.

10.3 CoNP

$$L \subseteq \Sigma^*$$

Complementary of a language. Diverging paths.

L is a language in P. Therefore $\hat{L}$ (complementary language of L) is also in P.

Proof. $\tau = (Q, \Sigma, \Gamma, \Delta, q_0, \Box, F)$ decides L in polynomial time.

Construct $\tau' \in \tau$ which computes $\hat{L}$ in polynomial time.

$$T' = \left\{Q', \Sigma, \Gamma, Delta\right\}$$

11 Hierarchy of complexity classes

Is $f_1, f_2 : \mathbb{N} \to \mathbb{N}$ with $f_1(n) \in \mathcal{O}(f_2(n))$ (and f_2 is time/space constructable), therefore:

$$\text{(N/D) SPACE}(f_1(n)) \subsetneq \text{(N/D) SPACE}(f_2(n))$$

$$\text{(N/D) SPACE}(f_1(n)) \subsetneq \text{SPACE}(f_2(n))$$

11.1 Time depends on Space

A deterministic turing machine requires $\mathcal{O}(f(n))$ space. More space requires more time to process. Therefore time is limited with $\mathcal{O}(2^{f(n)})$ accordingly. There are two approaches to prove this:

1. Additional tape: non-deterministic decisions
   The length of the tape is limited with $\mathcal{O}(2^{f(n)})$
2. Configuration of a turing machine
   $\Gamma(Q) + \#$tape squares $= \mathcal{O}(f(n))$

11.2 CANYIELD algorithm

12 Proof by reduction

12.1 Matrixinversion as example of an efficient algorithm

Solve the linear equation system by matrix inversion.

$$A \cdot x = b$$

$$x = A^{-1} \cdot b$$

12.2 Matrix multiplication

A, B	n×n matrix
I	n×n identity matrix
0	n×n zero matrix

$$\begin{pmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix}^{-1} = \begin{pmatrix} I & -A & A \cdot B \\ 0 & I & -B \\ 0 & 0 & I \end{pmatrix}$$