Selected Topics of Theoretical Computer Science (Vybrané partie z teoretické informatiky, VPTI, 460-4115)

Petr Jančar
katedra informatiky FEI VŠB-TU Ostrava

6. listopadu 2014

Poznámka k textu. Jedná se o studijní oporu k uvedenému kurzu, speciálně zamýšlenou pro studenty kombinované formy studia. Text je psán v angličtině, což by v magisterském studiu informatiky neměl být pro studenty problém. Text má 5 částí (kapitol), které zhruba odpovídají plánovaným 5 tutoriálům v semestru.

Část 1

1 RSA-cryptosystem, and the underlying theory

The aim of this section is to recall the RSA cryptographic method and the necessary theory showing its correctness, computational complexity, etc.

For a more thorough study, one can use, e.g., [1] and other sources (including, of course, the Internet ...)

1.1 Creating an RSA system

RSA (Rivest, Shamir, Adleman) is a particular method which can be used for the public-key cryptography.

A concrete RSA system is created as follows:

1. Choose randomly two different prime numbers p, q whose binary length is, say, approx. 500 bits (for each of them).
2. Compute $n = pq$ and $\Phi(n) = (p-1)(q-1)$. (Notation $\Phi(n)$ is clarified later.)
3. Choose some (small, odd) e, $3 \leq e < \Phi(n)$, such that $\gcd(e, \Phi(n)) = 1$ (where gcd denotes the greatest common divisor).
4. Compute d, $0 < d < \Phi(n)$, such that $ed \equiv 1 \pmod{\Phi(n)}$.
5. Make (e, n) public (this pair is the public key). A (block of a) message is (viewed as) a number $M, 0 \leq M < n$. The encryption function is $enc(M) = M^e \bmod n$.
6. The secret key is (d, n). The decryption function is $dec(C) = C^d \bmod n$.

The following subsections serve for clarifying the notions, feasibility, correctness, etc.

1.2 Size of inputs and cost of arithmetic operations

When dealing with number a (in this text, by numbers we mean integers, elements of $\mathbb{Z}$, or nonnegative integeres, elements of $\mathbb{N}$, depending on the context), we assume that the size of its description is $\beta = \log a$ (the binary length; by log we mean the base-2 logarithm; we are aware of the necessary rounding to integers, which is not directly reflected in our notation).

Performing an arithmetic operation with two large numbers cannot be taken as a single step when analysing the computational complexity. It is necessary to estimate the number of *bit operations* in such a case.

Exercise:

- Show that adding (subtracting) two β-bit integers can be done in time $O(\beta)$.

- Show that multiplication is in $O(\beta^2)$.

- Show that the (integer) division and computing the remainder (the mod-function) are both in $O(\beta^2)$. (Give a pseudocode of an algorithm.)

Remark. By 'divide-and-conquer' one can get for multiplication $O(\beta^{\log_2 3})$; the (asymptotically) fastest known algorithm is in $O(\beta \log \beta \log\log \beta)$. Nevertheless, the classical algorithms are sufficient for practice.

Exercise (optional):

- Suppose that multiplication or division of integers can be performed in time $M(\beta)$. Show that the binary-to-decimal conversion can be done in time $O(M(\beta)\log\beta)$. (You can use a 'divide-and-conquer' approach.)

- Show a polynomial algorithm to decide if a given n is a nontrivial power, i.e., if $n = a^k$ for some $k > 1$.

1.3 Divisibility, primes, gcd, unique factorization (prime decomposition)

We define, for $a, b \in \mathbb{Z}$:

$a|b$ (a divides b, or b is divisible by a) iff $\exists c \in \mathbb{Z} : ca = b$.

E.g.: $6|18$, $-6|18$, $6|-18$, $6 \not| 14$, $\forall a \in \mathbb{Z} : a|0$ (also $0|0$).

Recall the notion of (nonnegative) *common divisors* of two (or more) numbers, and the notion of the *greatest common divisor*; we put $\gcd(0,0) = 0$.

Exercise. For $a = 50, b = 35$ find all elements of the set $\{ax + by \mid x, y \in \mathbb{Z}\}$ which belong to the interval $-20, -19, \dots, 19, 20$. Argue that you have really found all of them.

Theorem: When a, b are not both 0, $gcd(a,b)$ is the smallest positive in the set $\{ax + by \mid x, y \in \mathbb{Z}\}$ (of linear combinations of a, b).

Exercise. Prove the theorem. (Hint. Let $A = \{m \mid m > 0 \wedge \exists x, y \in \mathbb{Z} : m = ax + by\}$, and let $d = gcd(a,b)$. Since $\forall m \in A : d|m$, the smallest element of A can be written cd for $c \geq 1$, and all $ax + by$ ($x, y \in \mathbb{Z}$) can be written icd for some $i \in \mathbb{Z}$ (why?). Thus $cd|a, cd|b$ (why?).)

Theorem: If $\gcd(a,b) = \gcd(a,c) = 1$ then $\gcd(a,bc) = 1$.

Exercise. Prove the theorem. (Hint. By the previous theorem we have $ax_1 + by_1 = 1$, and $ax_2 + cy_2 = 1$. Multiply (the left- and the riht-hand sides of) these two equations.)

We recall that $p \geq 2$ is a *prime* iff there is no $a, 1 < a < p$, such that $a|p$ (there is no factor, i.e. a nontrivial divisor, of p).

Theorem. Every number $n \geq 2$ has a *unique factorization* (decomposition to primes), i.e., n can be uniquely presented as $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ for some primes $p_1 < p_2 < \cdots < p_k$ and exponents $e_i \geq 1$.

Exercise. Can you prove the theorem? (Hint. Derive from a previous theorem that for prime p we have: $p|ab \Rightarrow (p|a \vee p|b)$. Then, for the sake of contradiction, assume the least $m = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k} = q_1^{b_1} q_2^{b_2} \dots q_\ell^{b_\ell}$ with two different prime decompositions ...)

Remark. Despite intensive research effort, no efficient algorithms for factoring an integer (in particular for finding p, q when given their product n in the RSA system) have been found. Since no other methods for cracking (enhanced) RSA have been found either, this cryptographic system is considered secure. (But the system should satisfy further conditions, like not having p, q close to each other etc.; in practice, further improvements of the basic RSA method, and combinations with other methods, are used.)

Exercise.

- Define $\gcd(a, b)$ using the prime decompositions $a = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, $b = q_1^{e'_1} q_2^{e'_2} \cdots q_{k'}^{e'_{k'}}$.

- Similarly define the *least common multiple* $\mathrm{lcm}(a, b)$.

- Generalize the notions of gcd and lcm for pairs to finite sets of numbers.

- Show that if there is a polynomial algorithm for some NP-complete problem (i.e., if P=NP) then there is a polynomial algorithm for factoring integers. (Hint. Note that the problem "Given n, k, is there a factor of n which is less than k?" is in NP. We could thus use binary search for finding a factor.)

1.4 Modular arithmetic, Fermat's little theorem

We recall the operations $\div$ (integer division, *div*) and mod (remainder of division) as well as the well-known equation (for $a \in \mathbb{Z}$, $n > 0$):

$$a = ((a \div n) \cdot n) + (a \bmod n)$$

E.g.: for $a = -17, n = 5$ we get $-17 = -4 \cdot 5 + 3$; for $a = 17, n = 5$ we get $17 = 3 \cdot 5 + 2$. Recall that $0 \leq a \bmod n < n$.

We recall what

$$a \equiv b \pmod{n}$$

means (we have already used this in the description of RSA); it just means $a \bmod n = b \bmod n$. E.g. $-17, 3, 628$ all belong to the same equivalence class of $\equiv \pmod{5}$.

We define $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ for $n \geq 1$ (e.g., $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$) and recall that the operations of addition, subtraction, and multiplication are well defined on the equivalence classes of $\equiv \pmod{n}$ (the equivalence classes can be represented by the elements of $\mathbb{Z}_n$).

Exercise. Show that $ab \bmod n = ((a \bmod n) \cdot (b \bmod n)) \bmod n$.

Remark. Observe the problem with division, i.e. the problem that some nonzero elements $a \in \mathbb{Z}_n$ might not have a multiplicative inverse a^{-1}, where $aa^{-1} = 1$. This does not happen iff n is prime; saying algebraically: $\mathbb{Z}_n$ is a field iff n is prime.

Exercise:

- Suppose p is a prime and $0 < k < p$; show that $p \,|\, \binom{p}{k}$.

- Deduce that $(a + b)^p \equiv a^p + b^p \pmod{p}$ (where p is a prime).

- Then deduce that $a^p \equiv a \pmod{p}$ for all $a \in \mathbb{Z}_p$. (Use that $2^p = (1 + 1)^p$ etc.)

- Derive (the following version of) *Fermat's little theorem*:

$$\boxed{\text{if } p \text{ is prime then } a^{p-1} \equiv 1 \pmod{p} \text{ for every } a, 1 \leq a \leq p-1.}$$

- Note the other direction: it is impossible that $a^{n-1} = 1 + kn$ when $gcd(a, n) = d \geq 2$.

1.5 Modular exponentiation (by repeated squaring)

Exercise. Compute $7^{560} \bmod 561$, with an aim of devising a general efficient algorithm for computing $a^b \bmod n$. (After you are done, look at the following algorithm.)

One possibility of a general procedure

$Modular\!-\!Exponentiation(a, b, n)$ computing $a^b \bmod n$:

```
Suppose $b$ in binary is $b_k b_{k-1} \dots b_1 b_0$
$c := 0; d := 1;$
(* invariant: $d = a^c \mod n$; variable $c$ is used only for this invariant*)
for $i := k$ downto 0 do
($c := 2c; d := d^2 \bmod n$; if $b_i = 1$ then ($c := c + 1; d := d \cdot a \mod n$))
```

Exercise.

- What is the binary description of c after j runs of the cycle ?

- Show that the procedure performs $O(\beta)$ arithmetic operations, and that the overall number of bit operations is in $O(\beta^3)$.

Remark. Another possibility is to compute $a^1 \bmod n$, $a^2 \bmod n$, $a^4 \bmod n$, $a^8 \bmod n$, ...

(using, of course, $a^{2^{i+1}} \bmod n = ((a^{2^i} \bmod n) \cdot (a^{2^i} \bmod n)) \bmod n$)

and then multiply the computed $a^{2^j} \bmod n$ for all j such that $b_j = 1$ in $b_k b_{k-1} \dots b_1 b_0$.

Remark. We have thus shown that encrypting and decrypting in the RSA system can be performed efficiently.

1.6 Computing gcd, extended Euclid algorithm

Exercise.

- Show that $gcd(a, b) = gcd(b, a \bmod b)$ for $a \geq b > 0$.

- Recalling $gcd(a, 0) = a$, devise a recursive algorithm (Euclid's algorithm) for computing $\gcd(a, b)$ where $a, b \geq 0$.

- Suppose that we have $x, y \in \mathbb{Z}$ such that $bx + (a \bmod b)y = \gcd(b, a \bmod b)$ $(= \gcd(a, b))$. Find x', y' so that $ax' + by' = \gcd(a, b)$. (Recall that $a \bmod b = a - (a \div b) \cdot b$.)

- Enhance your algorithm (to the *extended Euclid's algorithm*) so that it outputs not only $\gcd(a, b)$ for given a, b but also $x, y \in \mathbb{Z}$ such that $ax + by = \gcd(a, b)$.

- Show that your algorithm runs in time $O(\beta^3)$.

Remark. By a more detailed analysis we could get $O(\beta^2)$.
It is also interesting to note that Fibonacci numbers $0, 1, 1, 2, 3, 5, 8, 13, \dots$ are the worst-case for Euclid's algorithm. F_k is approximately $\phi^k/\sqrt{5}$ where ϕ is the golden ratio (1.618...)

(Golden ratio (divine proportion): $A - - - B - -C : AC/AB = AB/BC = \phi$. Removing the maximal square from a golden rectangle $1 : \phi$ we get a smaller golden rectangle ...)

Exercise.

- Consider steps 3., 4. (choosing e and d) in creating an RSA-system. Suppose you have chosen e and $\mathit{Extended{-}Euclid}(e, \Phi(n))$ returned $gcd(e, \Phi(n)) = 1 = xe + y\Phi(n)$. How you now (immediately) get d (such that $de \equiv 1 \pmod{\Phi(n)}$) ?

- Devise a way of generating candidates for e which you can prove to guarantee finding some e with $gcd(e, \Phi(n)) = 1$ quickly.

1.7 The Chinese remainder theorem

Take $n = 4 \cdot 5 \cdot 9 = 180$. Note that $4, 5, 9$ are pairwise *relatively prime*, i.e., $\gcd(a, b) = 1$ for each pair where $a \neq b$. Consider the sets

$$\mathbb{Z}_{180} = \{0, 1, 2, \ldots, 179\}$$
$$\mathbb{Z}_4 \times \mathbb{Z}_5 \times \mathbb{Z}_9 = \{0, 1, 2, 3\} \times \{0, 1, 2, 3, 4\} \times \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$$

Any number $m \in \mathbb{Z}_{180}$ can be mapped to the triple $(m \bmod 4, m \bmod 5, m \bmod 9)$ in $\mathbb{Z}_4 \times \mathbb{Z}_5 \times \mathbb{Z}_9$. E.g., 54 is mapped to $(2, 4, 0)$. The important thing is that this mapping is one-to-one, and thus a bijection between $\mathbb{Z}_{180}$ and $\mathbb{Z}_4 \times \mathbb{Z}_5 \times \mathbb{Z}_9$:

Consider a triple $(a, b, c) \in \mathbb{Z}_4 \times \mathbb{Z}_5 \times \mathbb{Z}_9$. There are $5 \cdot 9 = 45$ numbers in $\mathbb{Z}_{180}$ which are $\equiv a \pmod 4$, namely $a, a+4, a+8, \ldots$. We note that $a, a+4, a+8, a+12, a+16$ are all different wrt $\equiv \pmod 5$, and that $a+i\cdot 4, a+i\cdot 4+20, a+i\cdot 4+2\cdot 20, a+i\cdot 4+3\cdot 20, \ldots$ are all equivalent modulo 5. We thus get precisely 9 numbers in $\mathbb{Z}_{180}$, located periodically with period $4\cdot 5 = 20$, which are $\equiv a \pmod 4$ and $\equiv b \pmod 5$. The difference $m' - m$ of two different numbers out of these 9 is $j \cdot 4 \cdot 5$ for $0 < j < 9$ (hence 9 does not divide the difference); the remainders $\bmod 9$ of these 9 numbers thus fill the whole $\mathbb{Z}_9$. So there is precisely one number $m \in \mathbb{Z}_{180}$ such that $m \bmod 4 = a$, $m \bmod 5 = b$, $m \bmod 9 = c$.

Theorem. (The Chinese remainder theorem) Let $n_1, n_2, \ldots, n_k$ be pairwise relatively prime, and $n = n_1 n_2 \ldots n_k$.
Then the following is a one-to-one correspondence between $\mathbb{Z}_n$ and $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \ldots \mathbb{Z}_{n_k}$:
$a \leftrightarrow (a_1, a_2, \ldots, a_k)$ where $a_i \equiv a \pmod{n_i}$.

Exercise.

- Prove the theorem by generalizing our considerations in the concrete example above. (Hint. Show this by induction on k, starting with the case $n = n_1 n_2$.)

- Define naturally (componentwise) addition and multiplication on the set $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \ldots \mathbb{Z}_{n_k}$, and show that the above correspondence between $\mathbb{Z}_n$ and $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times \ldots \mathbb{Z}_{n_k}$ is an isomorphism wrt addition and multiplication.

- Show the following corollaries of the theorem:

- If $n_1, n_2, \ldots, n_k$ are pairwise relatively prime and $n = n_1 n_2 \ldots n_k$ then the set of equations $x \equiv a_1 \pmod{n_1}$, $x \equiv a_2 \pmod{n_2}$, $\ldots$, $x \equiv a_k \pmod{n_k}$ has a unique solution modulo n.

- If $n_1, n_2, \ldots, n_k$ are pairwise relatively prime and $n = n_1 n_2 \ldots n_k$ then for all x, a we have $x \equiv a \pmod{n_1}$, $x \equiv a \pmod{n_2}$, $\ldots$, $x \equiv a \pmod{n_k}$ iff $x \equiv a \pmod{n}$.

We note in particular that if $n = pq$, where p, q are different primes, then for all $C \in \mathbb{Z}$: $C \equiv 1 \pmod{n}$ iff $C \equiv 1 \pmod{p}$ and $C \equiv 1 \pmod{q}$.

1.8 Correctness of encryption-decryption in RSA

We note that the basic requirements which a (commutative) encryption-decryption system has to satisfy follow for RSA from the equation

$$(M^e)^d \equiv M \pmod{n}$$

for our $n = pq$, $ed \equiv 1 \pmod{\Phi(n)}$ where $\Phi(n) = (p-1)(q-1)$.

We note that $ed = 1 + k\Phi(n) = 1 + k(p-1)(q-1)$ for some k.

We thus want to show that

$$M \cdot M^{k(p-1)(q-1)} \equiv M \pmod{pq}$$

By the Chinese remainder theorem it is sufficient to show both the following equivalences:

$$M \cdot M^{k(p-1)(q-1)} \equiv M \pmod{p}$$
$$M \cdot M^{k(p-1)(q-1)} \equiv M \pmod{q}$$

We look at the first one (the second being similar). For $M \equiv 0 \pmod{p}$ this obviously holds. For $M \not\equiv 0 \pmod{p}$ we have $M^{(p-1)} \equiv 1 \pmod{p}$ due to Fermat's little theorem; since $M^{k(p-1)(q-1)} = (M^{(p-1)})^{k(q-1)}$, we are done.

Exercise. Write the above proof in more detail.

1.9 Generating random primes, the prime number theorem

The remaining problem in creating an RSA system, is the first step, i.e. generating the (large) random primes. (These primes will be, of course, odd.)

Using some (pseudo)random generator we can generate a random sequence of (approx.) 500 bits (starting and finishing with 1).

Remark. We do not discuss the (pseudo)random generators (PRG) here, we only recall that the programming languages usually contain some procedures enabling to accomplish our generating task. Of course, one should take into account their varying quality and suitability for our aims, which is another problem to study. E.g. a class of PRG are linear congruential generators; such a generator, given a seed x_0, produces $x_1, x_2, x_3, \dots$ where $x_{i+1} = (ax_i + b) \bmod m$ for some constants a, b, m. A famous one was RANDU from 1960s [where $a = 2^{16} + 3$, $b = 0$, $m = 2^{31}$] which turned out to be very bad, in fact [due to tight correlations between the successively generated numbers]. There are also hardware generators which convert data from a physical process into a sequence of (hopefully) random numbers, but we stop discussing this here ...

Now we need an efficient algorithm testing if the generated number is a prime. If we have such an algorithm, we can repeatedly generate random numbers until we finally find a prime.

The fact that we should succeed in reasonable time follows from the sufficient density of primes in the set of numbers, which is captured by the following theorem. By $\pi(n)$ we denote the number of primes in the set $\{2, 3, \ldots, n\}$.

The prime number theorem:

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1$$

This was conjectured by Gauss but proved later (by involved proofs). It can be shown that $\pi(n)$ is really very close to $\frac{n}{\ln n}$.

Remark. For our applications, it is sufficient that there are 'enough' primes (and we thus have a sufficient probability to get a prime in a randomly generated sequence of integers). A corresponding weaker version of the prime number theorem, still sufficient for our aims, can be proved by a simpler combinatorial proof. Intuitively: each number can be represented by its prime factorization, and if the primes were essentially less dense than $n/\ln n$ then we could represent each integer so concisely that we would get a contradiction with simple facts about 'Kolmogorov complexity' of words in $\{0,1\}^*$. (We cannot represent each sequence of n bits by a sequence of k bits for $k < n$.) More details can be found, e.g., in [5].

Exercise. Estimate how many random 500-bit numbers we should generate so that the probability that a prime is among them is greater than 90%. (You might use that $\ln 2 \approx 0.69$.)

But do we have an efficient algorithm for primality testing?

Exercise. Explain why the straightforward algorithm testing if there is a (nontrivial) divisor of n (in the range $2, 3, \ldots, \sqrt{n}$) can not serve for us.

Recalling Fermat's little theorem, we can consider the following (pseudoprimality testing):

```
given $n$,
if $Modular{-}Exponentiation(2, n{-}1, n) \neq 1$ (* i.e., $2^{n-1} \not\equiv 1 \pmod{n}$ *)
then return "$n$ is COMPOSITE" (definitely)
else return "$n$ is PRIME" (we hope).
```

This test is surely efficient (why?) and it works surprisingly well for random numbers but can be 'fooled' by some concrete numbers.
We can, of course, add testing if $Modular{-}Exponentiation(3, n{-}1, n) \neq 1$ and/or if $Modular{-}Exponentiation(a, n{-}1, n) \neq 1$ for a randomly generated $a, 2 \leq a \leq n{-}1$ but this still might not help. (We return to this later.)

In what follows we show the Miller-Rabin randomized primality set which solves our problem completely. To understand why it works requires a bit deeper look at some results from the number theory, the group theory, etc. We start by showing the Miller-Rabin test and then we recall the theory, maybe in a slightly broader manner than is really necessary for showing 'Miller-Rabin' correctness.

1.10 The Miller-Rabin randomized primality test

The 'main procedure' is the following:

$Miller{-}Rabin(n, s)$

```
    for $j := 1$ to $s$ do
        ($a := Random(1, n-1)$;
        if $Witness(a, n)$ then return "$n$ is COMPOSITE");
    return "$n$ is PRIME"
```

$Witness(a, n)$ is a boolean procedure which tells us if a is a witness of compositeness (in the sense explained below) or not; the answer YES will really guarantee that n is composite. So 'Miller-Rabin' can only err by returning "n is PRIME" for a composite n.

Exercise. We shall later show that if n is composite then the number of witnesses in the set $\{1, 2, \ldots, n-1\}$ is at least $(n-1)/2$. Assuming this, show that 'Miller-Rabin' can be used to recognize primes beyond any real doubt.

Now we show a pseudocode of procedure Witness. It uses a modification of Modular-Exponentiation.

procedure $Witness(a, n)$ (* determines if a is a witness of compositeness of n *)

```
    $n - 1$ in binary: $b_k b_{k-1} \ldots b_1 b_0$;
    $d := 1$;
    for $i := k$ downto 0 do
        $y := d; d := d^2 \mod n$;
        if $d = 1$ and $y \neq 1$ and $y \neq n-1$ then return TRUE;
        if $b_i = 1$ then $d := d \cdot a \mod n$;
    if $d \neq 1$ then return TRUE else return FALSE
```

We note that $Witness(a, n)$ computes $a^{n-1} \bmod n$ but this can be interrupted when

a *nontrivial square root of 1 modulo n*

is discovered, i.e., when some number x is found for which $x^2 \equiv 1 \pmod{n}$ but $x \not\equiv 1 \pmod{n}$, $x \not\equiv -1 \pmod{n}$. As we show shortly, this justifies the answer that n is composite (and a is a witness in the sense that we discover a nontrivial square root of 1 modulo n when computing $a^{n-1} \bmod n$). If no nontrivial square root of 1 modulo n is found during the computation, the answer YES in the case $a^{n-1} \not\equiv 1 \pmod{n}$ is really justified as we already know (Fermat's little theorem).

Exercise. Show that $Miller{-}Rabin(n, s)$ performs $O(s\beta^3)$ bit operations (where, of course, β is the binary length of the input).

1.11 Polynomials; a witness of compositeness

Our 'pragmatic' aim in this subsection is to show that nontrivial square roots of 1 modulo n really witness compositeness of n, i.e., we want to prove the following theorem.

Theorem. If n is a prime then there is no x such that $x^2 \equiv 1 \pmod{n}$, $x \not\equiv 1 \pmod{n}$, $x \not\equiv -1 \pmod{n}$.

Exercise. Verify that there is no nontrivial square root of 1 modulo n for a few primes n. On the other hand find some nontrivial square roots of 1 modulo n for some composite n.

The theorem can be, as usual, proved in various ways. We use a more general one since it is useful to recall the notions of polynomials.

Exercise.

- Recall what a polynomial (in one variable) is. In particular consider polynomials over $\mathbb{Z}_p$ where p is a prime. (Remark. You might recall that the structure $(\mathbb{Z}_p, +, \cdot)$ is a (commutative) field.)

- Recall what is the *degree* of a polynomial $f(x)$ and what is a *root* of the polynomial, or so called *zero* of the respective polynomial function.
(Yes, it is a value x_0 such that $f(x_0) = 0$, i.e., $f(x_0) \equiv 0 \pmod p$ in our case.)

- Show that if $f(x)$ is a polynomial of degree $t \geq 1$ (with coefficients from $\mathbb{Z}_p$) and $a \in \mathbb{Z}_p$ is one of its roots (zeros) (i.e., $f(a) \equiv 0 \pmod p$) then $f(x)$ can be presented as the product

$$f(x) \equiv (x - a)g(x) \pmod p$$

for a polynomial $g(x)$ of degree $t - 1$.

- Deduce that

each nonzero polynomial (not identical with 0) of degree t has at most t distinct roots (also called 'zeros') modulo p (in the field $(\mathbb{Z}_p, +, \cdot)$ for p prime).

- Deduce the above Theorem (if n is a prime then there is no nontrivial square root of n).

1.12 Groups, subgroups, modular linear equations, Euler's phi function

To finish our complete understanding of (RSA and) Miller-Rabin primality testing, we should show that $Witness(a, n)$ really returns TRUE for at least one half of $a \in \{1, 2, \ldots, n-1\}$ when n is (an odd) composite. To achieve this, we 'submerge more deeply in algebra'.

Recall a definition of (the algebraic structure) *group*: it is a set G with a binary operation (often denoted by '+' or '$\cdot$') such that ...

Exercise. Write down a precise definition (consulting some relevant sources, on the Internet or elsewhere, if needed).

We note that $\mathbb{Z}_n$ is a group when we take (modular) addition as the group operation; this (additive) group is denoted by $(\mathbb{Z}_n, +_n)$, or simply by $\mathbb{Z}_n$ when no confusion can arise.

Exercise.

- Recall what is a subgroup of a given group. Construct the subgroup of $\mathbb{Z}_{15}$ generated by $a = 6$ (i.e., the least subgroup of $\mathbb{Z}_{15}$ containing 6). (You surely consider the sequence a, $a+a$, $a+a+a$, ..., i.e., a, $2a$, $3a$, ...)

- Derive that if $d = gcd(a, n)$ then the subgroup generated by a is the set

$$\{0, d, 2d, 3d, \ldots, ((n/d)-1)d\}$$

of size n/d (where addition modulo n is the group operation).

- **Modular linear equations.** Devise a way to find solutions of $ax \equiv b \pmod{n}$.
Hint. Note that there are either $d = gcd(a,n)$ distinct solutions (in $\{0, 1, \ldots, n-1\}$), when $d|b$, or no solution. By Extended-Euclid: $d = gcd(a,n) = ax' + ny'$; the solutions are $x_0 = x'(b/d)$; $x_1 = x_0 + (n/d)$, $x_2 = x_0 + 2(n/d)$, ...

- Deduce that if $gcd(a,n) = 1$ then $ax \equiv b \pmod{n}$ has a unique solution.

When discussing RSA (namely computing e, d) we have already implicitly used the notion of the *multiplicative inverse*:

A (unique) solution of $ax \equiv 1 \pmod{n}$ exists iff $gcd(a,n) = 1$. (Extended-Euclid providing $1 = gcd(a,n) = ax + ny$ gives the multiplicative inverse x of a.)

Exercise. Explain now why $\mathbb{Z}_n$ is not a group (for $n \geq 2$) when we consider (modular) *multiplication* as the operation.

Recall that $a, b \geq 0$ are *relatively prime* iff $\gcd(a,b) = 1$. Given $n \geq 2$, we define the multiplicative group $(\mathbb{Z}_n^*, *_n)$, often denoted just $\mathbb{Z}_n^*$ for short, as the set of all $a \in \mathbb{Z}_n$ which are relatively prime to n; the group operation is (modular) multiplication.

Exercise. Verify that $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$ and that it is really a group.

Size of $\mathbb{Z}_n^*$ is given by Euler's phi function (Euler's totient function)

$$\Phi(n) = n \prod_{p|n} (1 - 1/p)$$

where p ranges over primes (which divide n).

E.g. $\Phi(15) = 15(1 - \frac{1}{3})(1 - \frac{1}{5}) = 15(\frac{2}{3})(\frac{4}{5}) = 8$,
$\Phi(45) = 45(1 - \frac{1}{3})(1 - \frac{1}{5}) = 45(\frac{2}{3})(\frac{4}{5}) = 24$.

We can thus see that n is prime iff $\Phi(n) = n - 1$.

Exercise. Show the validity of $\Phi(n)$ for our 'RSA case': namely that our $\Phi(n) = (p-1)(q-1)$ coincides with the above definition for $n = pq$ where p, q are different primes, and that $\Phi(n)$ really gives the size of $\mathbb{Z}_n^*$ in this case.
Then show that $|\mathbb{Z}_n^*| = \phi(n) = (p-1)p^{e-1}$ in the case $n = p^e$, for p being prime and $e \geq 1$.

Exercise. Prove the general formula for $\Phi(n)$ by the following application of the combinatorial approach of inclusion and exclusion (where $p_1, p_2, \ldots, p_t$ are the primes dividing n):

$\Phi(n) = n - \left(\frac{n}{p_1} + \frac{n}{p_2} + \cdots + \frac{n}{p_t}\right)$
$+\left(\frac{n}{p_1p_2} + \frac{n}{p_1p_3} + \cdots + \frac{n}{p_{t-1}p_t}\right)$ (for all pairs)
$-\left(\frac{n}{p_1p_2p_3} + \frac{n}{p_1p_2p_4} + \cdots + \frac{n}{p_{t-2}p_{t-1}p_t}\right)$ (for all triples)
$\cdots$
$+(-1)^t \frac{n}{p_1p_2\cdots p_t}$

$= \frac{n}{p_1p_2\cdots p_t}(p_1 - 1)(p_2 - 1)\cdots(p_t - 1)$.

1.13 Lagrange's theorem, Euler's theorem

Exercise. Consider $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$. Construct the least subset A of $\mathbb{Z}_{15}^*$ which contains elements $4, 8$ and is closed wrt the group operation (i.e., wrt multiplication mod 15). Recall the notion of subgroup of a group, and check if your set A is, in fact, a subgroup of

$\mathbb{Z}^*_{15}$.

Exercise. Prove the following theorem.

Theorem. A nonempty subset A of a finite group G which is closed wrt the group operation is a subgroup of G.

Exercise.

- Create all subgroups of $\mathbb{Z}^*_{15}$ which are generated by one element. (For each $a \in \mathbb{Z}^*_{15}$, construct the least subgroup containing a.)

- For the subgroup $H = \{1, 4\}$ construct the set $gH \subseteq \mathbb{Z}^*_{15}$ for each $g \in \mathbb{Z}^*_{15}$. (We define $gH = \{gh \mid h \in H\}$.) Note that for each $g_1, g_2 \in Z^*_{15}$ we have got either $g_1H = g_2H$ or $g_1H \cap g_2H = \emptyset$. $\{gH \mid g \in G\}$ is thus a partition on the set $\mathbb{Z}^*_{15}$.

Theorem. (Lagrange): Suppose S is a finite group and S' is a subgroup of S. Then $|S'|$ divides $|S|$. (This implies that if S' is a proper subgroup then $|S'| \leq |S|/2$.)

Proof. (More generally.) Let H be a subgroup of (even an infinite) G. Consider $\{gH \mid g \in G\}$. This is a partition of (the set) G. ($ah_1 = bh_2$ implies $ah = ah_1h_1^{-1}h = bh_2h_1^{-1}h \in bH$.)

The number $|\{gH \mid g \in G\}|$ is called the *index of subgroup* H in group G, denoted $[G : H]$.

Note that for each g we have $|H| = |gH|$ ($gh_1 = gh_2$ implies $g^{-1}gh_1 = g^{-1}gh_2$ and thus $h_1 = h_2$).

So $|G| = [G : H] \cdot |H|$. □

Recall how we constructed the subgroup of a finite group S generated by an element a (a, $a+a$, $a+a+a$, $\ldots$ in the additive notation, or a, aa, aaa, $\ldots$ in the multiplicative notation).

The *order* of a in S, denoted $ord(a)$, is the least $t \geq 1$ s.t. $a^t = e$ (or ta in the additive notation), where e is the unit. We can see that $ord(a)$ is equal to the size of the subgroup generated by a.

Exercise.

- Derive what is $a^{|S|}$ for an element a of a finite group S.

- Derive

Euler's theorem: for $a \in \mathbb{Z}^*_n$ we have $a^{\phi(n)} \equiv 1 \pmod{n}$.

- Note that Fermat's little theorem is, in fact, a corollary of Euler's theorem.

- Show how we can compute $a^{-1} \mod n$ (for $a \in \mathbb{Z}^*_n$) using Modular-Exponentiation, when we know $\phi(n)$. (Hint. Recall $a^{\Phi(n)} \equiv 1 \pmod{n}$.)

1.14 Enough witnesses in Miller-Rabin, Part 1

We recall that we aim at showing the following theorem.

Theorem: If n is an odd composite number then the number of witnesses (a, for which $Witness(a, n) = TRUE$) is at least $(n-1)/2$.

We show, in fact, that the number of nonwitnesses is at most $(n-1)/2$.

Exercise. Show that every nonwitness a must be an element of $\mathbb{Z}^*_n$ (i.e., $gcd(a, n) = 1$).

The overall strategy is to show that all nonwitnesses are contained in a *proper* subgroup B

of $\mathbb{Z}_n^*$; so (by Langrange's theorem) their number is less than $|\mathbb{Z}_n^*|/2$, i.e., less than $(n-1)/2$.
We are now able to prove easily one (in fact, substantial) subcase.
Case 1: Suppose that $\exists x \in \mathbb{Z}_n^*$ s.t. $x^{n-1} \not\equiv 1 \pmod n$. Since

$$B = \{b \in \mathbb{Z}_n^* \mid b^{n-1} \equiv 1 \pmod n\}$$

is (nonempty and) closed under multiplication, it is a proper subgroup containing all nonwitnesses.

Unfortunately, we are not done, since there exist so called *Carmichael numbers*, which satisfy

$$a^{n-1} \equiv 1 \pmod n \text{ for all } a \in \mathbb{Z}_n^*.$$

(They also show why our previous 'Fermat-based testing' is not sufficient for testing primality.) Though these numbers are very rare (561, 1105, 1729, ...), we have to count with them.
Note that $561 = 3 \cdot 11 \cdot 17$, and thus the size of $\mathbb{Z}_{561}^*$ is $\Phi(561) = 561(1-\frac{1}{3})(1-\frac{1}{11})(1-\frac{1}{17}) = 320$.

1.15 Cyclic groups

A group G is called *cyclic* if it is generated by one element $g \in G$, called a generator of G. Thus $G = \{g, g^2, g^3, \dots\}$ (in the multiplicative notation).

Exercise.

- What is the order of a generator g in G ?
- Find the orders of elements in $\mathbb{Z}_5^* = \{1,2,3,4\}$. Is $\mathbb{Z}_5^*$ cyclic?
- Recall that you constructed all subgroups of $\mathbb{Z}_{15}^*$ generated by one element. Is $\mathbb{Z}_{15}^*$ cyclic?
- Find $n \geq 4$ such that n is composite and $\mathbb{Z}_n^*$ is cyclic.

A theorem says:

$\mathbb{Z}_n^*$ $(n > 1)$ is cyclic precisely for those n whose values are 2, 4, p^e, or $2p^e$ where p is any odd prime and e is any positive integer.

We only show the following subcase: for p prime, $\mathbb{Z}_p^*$ is cyclic.

Proof.

Claim 1. For $x, y \in Z_p^*$, let $ord(x) = m_1$, $ord(y) = m_2$ where $\gcd(m_1, m_2) = 1$. Then $ord(xy) = m_1 m_2$.

> Let $ord(xy) = k$, so $(xy)^k \equiv 1 \pmod p$. Then $(xy)^{km_2} = x^{m_2 k} y^{m_2 k} \equiv 1 \pmod p$ and thus $x^{m_2 k} \equiv 1 \pmod p$ (since $y^{m_2} \equiv 1 \pmod p$). Hence $m_1 | m_2 k$ which means $m_1 | k$ (since $\gcd(m_1, m_2) = 1$). Similarly $m_2 | k$, and thus $m_1 m_2 | k$. So $k \geq m_1 m_2$. But $(xy)^{m_1 m_2} \equiv (x^{m_1})^{m_2} \cdot (y^{m_2})^{m_1} \equiv 1 \pmod p$, and thus $k = m_1 m_2$.

Claim 2. Let $ord(x) = m_1$, $ord(y) = m_2$. Then there is z such that $ord(z) = \mathrm{lcm}(m_1, m_2)$.

> Using prime factorizations of m_1, m_2 we can easily find a, b such that $a|m_1$, $b|m_2$, $\gcd(a,b) = 1$, and $\mathrm{lcm}(a,b) = ab = \mathrm{lcm}(m_1, m_2)$. Now $ord(x^{m_1/a}) = a$, $ord(y^{m_2/b}) = b$, so $ord(x^{m_1/a} y^{m_2/b}) = ab = \mathrm{lcm}(m_1, m_2)$ (using Claim 1).

So there is an element $z \in \mathbb{Z}_p$ such that $ord(z) = \text{lcm}\{ord(1), ord(2), \ldots, ord(p-1)\}$.
Thus all $1, 2, \ldots, p-1$ are roots (zeros) of the polynomial $x^{ord(z)} - 1$ (modulo p). This polynomial can have at most $ord(z)$ roots so necessarily $ord(z) = p-1$. □

Remark. The Claims do not use the assumption that p is prime, we need this for bounding the number of roots.
To complete our later reasoning we would need the generalization "for $n = p^e$ (p being an odd prime), the group $\mathbb{Z}_n^*$ is cyclic". (More precisely, it would suffice: if $n = p^e$, $e \geq 2$, then there is $a \in \mathbb{Z}_n^* : a^{n-1} \not\equiv 1 \pmod{n}$.)

Exercise.

- Find all generators of $\mathbb{Z}_5^*$. Check if those generators are also generators of $\mathbb{Z}_{25}^*$.

- Try to prove that if g is a generator of $\mathbb{Z}_p^*$ (p is prime) then g is also a generator for $\mathbb{Z}_{p^e}^*$ (for $e = 2, 3, \ldots$).

1.16 Discrete logarithm

Discrete logarithm (or index) $ind_{n,g}(a)$ is defined as the number z such that $g^z \equiv a \pmod{n}$, where g is a generator of $\mathbb{Z}_n^*$.

Discrete logarithm theorem:
if g is a generator of $\mathbb{Z}_n^*$, then $g^x \equiv g^y \pmod{n}$ iff $x \equiv y \pmod{\phi(n)}$.

Recall that we have shown that there is no (nontrivial square root of 1) $x^2 \equiv 1 \pmod{p}$, $x \not\equiv 1 \pmod{p}$, $x \not\equiv -1 \pmod{p}$ for p being prime. We illustrate a use of discrete logarithm on a stronger version:

Theorem: if p is an odd prime and $e \geq 1$, then the equation $x^2 \equiv 1 \pmod{p^e}$ has only two solutions, namely 1 and -1.

Proof. Denote $n = p^e$ and let g be a generator of $\mathbb{Z}_n^*$.
We have $(g^{ind_{n,g}(x)})^2 \equiv g^{ind_{n,g}(1)} \pmod{n}$,
so $2 \cdot ind_{n,g}(x) \equiv 0 \pmod{\phi(n)}$ (where $\phi(n) = p^e(1 - 1/p) = p^e - p^{e-1} = p^{e-1}(p-1)$),
so $gcd(2, \phi(n)) = 2$, so $ind_{n,g}(x)$ can have exactly two values, so also $x^2 \equiv 1 \pmod{p^e}$ has exactly two solutions, namely $+1$ and -1. □

1.17 Enough witnesses in Miller-Rabin, Part 2

Case 2 (Carmichael numbers) Suppose that $\forall x \in \mathbb{Z}_n^* : x^{n-1} \equiv 1 \pmod{n}$.

We can not have $n = p^e$ for $e > 1$ where p is a prime. Otherwise $\mathbb{Z}_n^*$ contains a generator g s.t. $ord(g) = |\mathbb{Z}_n^*| = \phi(n) = (p-1)p^{e-1}$, and $g^{n-1} \equiv 1 \pmod{n}$. But then the discrete logarithm theorem implies $n-1 \equiv 0 \pmod{\phi(n)}$, i.e., $(p-1)p^{e-1} \mid p^e - 1$; but this is impossible for $e > 1$.

So we can write $n = n_1 n_2$ so that $n_1, n_2 > 1$ are relatively prime (and we recall the Chinese remainder theorem).

Let $n-1 = u2^t$ where u is odd ($n-1$ in binary finishes with t zeros). For any a (in $\{1, 2, \ldots, n-1\}$) consider the last $t+1$ steps of $Modular{-}Exponentiation(a, n-1, n)$, computing a^u, a^{2u}, $a^{2^2 u}$, …, $a^{2^t u}$ modulo n.

Note that $(-1)^{2^0 u} \equiv -1 \pmod{n}$; let $j \in \{0, 1, \ldots, t\}$ be the maximal such that there is v

s.t. $v^{2^j u} \equiv -1 \pmod{n}$; we fix such v.

Let $B = \{x \in \mathbb{Z}_n^* \mid x^{2^j u} \equiv 1 \pmod{n} \text{ or } x^{2^j u} \equiv -1 \pmod{n}\}$.

Exercise. Show that B is a subgroup of $\mathbb{Z}_n^*$ and that every nonwitness must be an element of B.

It is thus sufficient to show that there is some $w \in \mathbb{Z}_n^* - B$; we do this by an application of the Chinese remainder theorem to our $n = n_1 n_2$:

Since $v^{2^j u} \equiv -1 \pmod{n}$, we have $v^{2^j u} \equiv -1 \pmod{n_1}$.

There is some w satisfying

$w \equiv v \pmod{n_1}$ and

$w \equiv 1 \pmod{n_2}$.

Therefore

$w^{2^j u} \equiv -1 \pmod{n_1}$ and

$w^{2^j u} \equiv 1 \pmod{n_2}$.

This means that we can not have $w^{2^j u} \equiv 1 \pmod{n}$, neither $w^{2^j u} \equiv -1 \pmod{n}$ (by a corollary of the Chinese remainder theorem). Thus $w \notin B$.

Since $v \in \mathbb{Z}_n^*$ ($gcd(v, n) = 1$), we have $gcd(v, n_1) = 1$, so $gcd(w, n_1) = 1$ (and $gcd(w, n_2) = 1$), and thus $w \in \mathbb{Z}_n^*$. $\square$

Remark. By a more detailed analysis one can show that the number of nonwitnesses is at most $(n-1)/4$, which really happens for some numbers.

Část 2

2 A randomized communication protocol; fingerprinting

(Based on [5].)

Suppose two copies of the same database (e.g., containing genome sequences) are maintained at two distant computers C_1, C_2; the database is of size approx. 10^{16} bits.

We want to suggest a communication protocol between C_1 and C_2 which (from time to time) has to verify that the two database copies are really the same.

An obvious solution is that one computer, say C_1, sends all $n = 10^{16}$ bits to C_2 and C_2 verifies that the achieved bit sequence coincides with its copy.

Exercise

- Is this procedure a trivial task in practice?

- Do you think that we have some substantially better possibility when we want to be 100% sure that the copies are really the same?

In reality, we never need to be 100% sure, something like 99.999999999999% certainty is surely sufficient for us. (In fact, when sending $n = 10^{16}$ bits as suggested above, we cannot be 100% sure that they arrive safely (without flipping a bit) anyway ...)

But then the following simple protocol works for us:

- The content of the database at C_1 is $x = b_1 b_2 \dots b_n$, where b_i are bits (elements of $\{0, 1\}$); we suppose (i.e., easily arrange) that we always have $b_1 = 1$.
- C_1 chooses randomly a prime p from interval $2, 3, \dots, n^2$. (We recall that we have discussed how to generate random primes.)
- C_1 computes $s = Number(x) \bmod p$ (where $Number(x)$ is the number presented as $b_1 b_2 \dots b_n$ in binary).
- C_1 sends the pair (p, s) to C_2.
- C_2 computes $q = Number(y) \bmod p$ for its database copy y. If $s = q$ then C_2 announces "OK, our copies are the same", otherwise C_2 announces "Oops, our copies differ!".

Exercise
- Estimate the numbers of bits in binary presentation of p, s (and q), and compare the practical task of communicating p, s with the task of sending the whole $x = b_1 b_2 \dots b_n$.
- Why is the operation of choosing a random prime p and computing s manageable on nowadays machines? (Recall also the prime number theorem: $\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1$.)

We now want to verify that the probability of the right answer is high for every given input (not only for random inputs).

Exercise
- Characterize the situation when the protocol errs.
- Give an upper bound on the number of primes in the (prime) factorization of

$|Number(x) - Number(y)|$.
- Deduce (a bound on) the probability that the protocol errs, for general n, and for concrete $n = 10^{16}$.
- Suggest an improvement of the protocol if you want to still decrease the error-probability.

Fingerprinting

The previous example of comparing two databases (two long sequences of bits) can naturally remind us about fingerprints. (The complex objects, like people, can be sufficiently represented by small objects - fingerprints.)

Exercise. Try to formulate some form of a general method for comparing complex objects which is based on 'fingerprinting'.

As another example we consider comparing two polynomials. For simplicity we restrict ourselves to polynomials with one variable (a generalization for polynomials with more variables is in the next subsection).

So given two polynomials $p_1(x), p_2(x)$ (with integer coefficients), we should decide if they are the same, i.e. if their normal forms (see below) are the same (i.e., if the values $p_1(a), p_2(a)$ are the same for any given a in any interpretation). An example of polynomials appearing in the problem instances is
$p_1(x) = (2x^2 - x)^5 \cdot (x^4 + 3x^2 - 5)^{12} - (-3x^7 + 2x^5 - 4x + 17)^{24}$,
$p_2(x) = \dots$.
Important is that $p_1(x), p_2(x)$ are not required to be in normal form (which is a sum of monomials with different degrees $c_d x^d + c_{d-1}x^{d-1} + \cdots + c_2x^2 + c_1x + c_0$).

Exercise. Comparing two polynomials in normal form is easy (why?). But show that transforming a polynomial in general form (as exemplified above) into normal form can mean exponential computation.

There is a much faster method of comparing $p_1(x), p_2(x)$ than transforming them into normal form; roughly speaking, we can do the following:

> Take a random a and evaluate $p_1(a)$ and $p_2(a)$. If $p_1(a) \neq p_2(a)$ then return "p_1, p_2 are not the same", otherwise (when $p_1(a) = p_2(a)$) return "p_1, p_2 are the same".

From where we take the random a ? And how can we guarantee that the probability of an incorrect answer "p_1, p_2 are the same" is (very) small?

Exercise. If we want to have an efficient (i.e., polynomial probabilistic) algorithm to compare p_1, p_2, the (binary description of the) generated a has to have a polynomial size. On the other hand, it must be guaranteed that the probability of $p_1(a) = p_2(a)$ is very small when $p_1 \neq p_2$. Recall what we know about polynomials over finite fields (like $\mathbb{Z}_p$ where p is prime) and suggest how to solve our task by using this knowledge.
(Hint. Note that $p_1(a) \equiv p_2(a) \pmod{m}$ iff $q(a) \equiv 0 \pmod{m}$ for the polynomial $q = p_1 - p_2$. Compute an upper bound on the degree of q [in the size of the problem instance]; this gives an upper bound on the number of roots of q when m is prime. Choose a sufficiently large (but feasible) prime m ...)

Comparing polynomials with more variables (optional)

Imagine we compare polynomials $p_1(x_1, \ldots, x_n)$, $p_2(x_1, \ldots, x_n)$ over the finite field $\mathbb{Z}_p$, for p a prime. Let d be the maximum of degrees of particular variables in p_1, p_2.

So we ask if $Q(x_1, \ldots, x_n) = p_1(x_1, \ldots, x_n) - p_2(x_1, \ldots, x_n)$ is identical to the zero polynomial 0 (modulo p).

Supposing $Q \not\equiv 0$, we want to bound the number of roots $(a_1, \ldots, a_n) \in (\mathbb{Z}_p)^n$ of Q. When $n = 1$ (we have only one variable) then we know that there are at most d roots.

Now we try to arrive at an inductive formula $f(n, d, p)$ bounding the number of roots when $n > 1$. (We have shown $f(1, d, p) = d$.)

We note that $Q(x_1, \ldots, x_n)$ can be written in the form

$Q_0(x_2, \ldots, x_n) + x_1 \cdot Q_1(x_2, \ldots, x_n) + x_1^2 \cdot Q_2(x_2, \ldots, x_n) + \cdots + x_1^d \cdot Q_d(x_2, \ldots, x_n)$

for some polynomials $Q_i(x_2, \ldots, x_n)$, $i = 0, 1, 2, \ldots, d$, of degree $\leq d$ (for all particular variables).

We observe that $(a_1, \ldots, a_n)$ is a root of Q iff

- either (1): $(a_2, \ldots, a_n)$ is a root for all Q_i, $0 \leq i \leq d$,
- or (2): a_1 is a root of the one-variable nonzero polynomial

$Q_0(a_2, \ldots, a_n) + x_1 \cdot Q_1(a_2, \ldots, a_n) + x_1^2 \cdot Q_2(a_2, \ldots, a_n) + \cdots + x_1^d \cdot Q_d(a_2, \ldots, a_n)$.

Exercise.

- Show that there can be at most $p \cdot f(n-1, d, p)$ n-tuples satisfying (1).
- Show that there can be at most $d \cdot p^{n-1}$ n-tuples satisfying (2).
- Conclude that we can take $f(n, d, p) = d \cdot p^{n-1} + f(n-1, d, p) \cdot p$.
- From the sequence $f(1, d, p) = d$, $f(2, d, p) = d \cdot p + f(1, d, p) \cdot p$, $f(3, d, p) = d \cdot p^2 + f(2, d, p) \cdot p$, $\ldots$, derive a direct (nonrecursive) definition of $f(n, d, p)$.
(Result: $n \cdot d \cdot p^{n-1}$.)
- How big must p be wrt n, d when we want a guarantee that, for $Q \not\equiv 0$, the probability that $Q(a_1, \ldots, a_n) = 0$ for (uniformly) randomly chosen $(a_1, \ldots, a_n)$ is at most 1/2 ?

3 Interactive protocols

(For this topic, we mainly use [8].)

We recall that for any instance of a problem from NP there is a short proof ("witness", verifiable in polynomial time) showing that the answer is YES if it is really the case.

Exercise. Recall some NP-complete problems like SAT, ... and show how such witnesses look like in these concrete cases.

For the negative case, e.g. for showing that a cnf-boolean formula ϕ is unsatisfiable, the existence of such short witnesses is not clear at all. But one of interesting and maybe surprising results is that there are so called interactive protocols, between Prover and Verifier, that show that even for any problem from PSPACE there is a possibility how one can get convinced about the right answer for any given instance after doing only a polynomial computation.

(This result is known as IP=PSPACE.)

We shall first demonstrate the princip on the G-NONISO (the graph nonisomorpism problem), and then on the (counting) problem #SAT.

Recall that G-ISO (the graph isomorphism problem) is in NP but not known to be in P or to be NP-complete. It is obvious how somebody can convince you that two graphs G_1, G_2 (with thousands of nodes, say) are isomorphic: if you get a bijection between the sets of nodes (we can assume the set of nodes being $\{1, 2, \ldots, n\}$ in both cases, the bijection then being just a permutation of $\{1, 2, \ldots, n\}$) then you can easily verify (by a computer program, say) whether the bijection is an isomorphism ...

But how can somebody, named Prover, convince you that G_1, G_2 (both with the set $\{1, 2, \ldots, n\}$ of nodes) are *not* isomorphic? Imagine you secretly choose a random $i \in \{1, 2\}$, a random permutation $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ (how you can do this in your favourite programming language?) and you show $H = \pi(G_i)$ (i.e., the graph H arising from G_i by renaming its nodes according to π) to Prover. If he now rightly guesses your secret i, and he is able to successfully repeat such right guessing 50 times, say, you should be convinced that G_1, G_2 are not isomorphic. (Remark. It is not a matter of our discussion how Prover finds his answers; you are just supposed to be sure that he cannot see your secret actions.)

Exercise. Analyze and explain why you would be sure (more precisely, with what probability you can be fooled).

Now we come to a more difficult problem #SAT; this problem can be defined as the language

$$\#SAT = \{(\phi, k) \mid \phi \text{ is a cnf-boolean formula with precisely } k \text{ satisfying assignments}\}.$$

We can imagine that Prover has, e.g., the exponentially big truth-table at his disposal (as a proof that ϕ has precisely k satisfying assignments) but our task is to suggest a communication protocol between Prover and Verifier during which Verifier performs only a polynomial computation and will be finally convinced that (ϕ, k) is in #SAT, though Prover can give him only very limited (polynomial) information. Of course, by 'convinced' we mean 'convinced by objective arguments', not e.g. by trusting Prover. So a malicious Prover who tries to persuade Verifier about a wrong answer must not succeed.

Remark. We do not care how Prover creates his messages, similarly as we did not care how a witness (satisfying assignment) for SAT can be found when we discussed its existence. (In this sense, Prover can be viewed as having unlimited computational power, not only exponential.) Another remark is that probability will again play a crucial role. It is sufficient that Verifier gets convinced with 99.99999% certainty ...

An interactive protocol for #SAT

Prover (P) and Verifier (V) get a cnf-boolean formula $\phi(x_1, \ldots, x_m)$; as a running example we consider

$\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3)$.

Exercise. Show that there are precisely 4 assignments $(a_1, a_2, a_3) \in \{0, 1\}^3$ (for boolean variables x_1, x_2, x_3) which satisfy ϕ.

Let $f_i(a_1, \ldots, a_i)$ $(0 \leq i \leq m)$ denote the number of satisfying assignments of ϕ which must assign $x_1 := a_1, \ldots, x_i := a_i$.

In our example: $f_0() = 4$, $f_1(0) = 1$, $f_1(1) = 3$, $f_2(1,0) = 1$, $f_3(0,1,0) = 0$, $f_3(1,0,1) = 1, \ldots$.

Exercise.

- Compute all values $f_i(a_1, \ldots, a_i)$ for our ϕ.

- Note that generally (for $i < m$): $f_i(a_1, \ldots, a_i) = f_{i+1}(a_1, \ldots, a_i, 0) + f_{i+1}(a_1, \ldots, a_i, 1)$.

With our example ϕ we associate the polynomial

$p(x_1, x_2, x_3) = (1 - (1 - x_1)(1 - (1 - x_2))) \cdot (1 - (1 - x_2)(1 - x_3))$.

Exercise.

- Verify that for boolean arguments a_1, a_2, a_3 we have $p(a_1, a_2, a_3) = 1$ iff (a_1, a_2, a_3) satisfies ϕ and $p(a_1, a_2, a_3) = 0$ otherwise.

- Along the lines of the example, suggest a general procedure constructing a polynomial p_ϕ to a boolean formula ϕ which has this "boolean values property".

- Moreover, check that the polynomial p_ϕ has degree $\leq n$ where $n = \mathit{length}(\phi)$ (by the degree we mean the maximum of exponents of each individual variable, in the normal form).

- Check also that p_ϕ has the "boolean values property" also when interpreted over a finite field $\mathbb{Z}_q$, for q being a prime.

- Verify that the normal form of our example polynomial (presented as the sum of different monomials) is

$p(x_1, x_2, x_3) = -x_1x_2^2x_3 + x_1x_2x_3 + x_1x_2^2 + x_2^2x_3 - x_2^2 - 2x_2x_3 + x_2 + x_3$.

We now view the above functions f_i as polynomials $f_i(x_1, x_2, \ldots, x_i)$:

$$f_i(x_1, \ldots, x_i) = \sum_{a_{i+1}, \ldots, a_m \in \{0,1\}} p(x_1, \ldots, x_i, a_{i+1} \ldots, a_m) .$$

Exercise.

- Check that we have $f_i(r_1, \ldots, r_i) = f_{i+1}(r_1, \ldots, r_i, 0) + f_{i+1}(r_1, \ldots, r_i, 1)$ for all numbers $r_1, \ldots, r_i$.

-Check that in our example

- $f_0() = 4$,
- $f_1(x_1) = 2x_1 + 1$,
- $f_2(x_1, x_2) = x_1x_2^2 + x_1x_2 - x_2^2 + 1$,
- $f_3(x_1, x_2, x_3) = p(x_1, x_2, x_3) = -x_1x_2^2x_3 + x_1x_2x_3 + x_1x_2^2 + x_2^2x_3 - x_2^2 - 2x_2x_3 + x_2 + x_3$.

Now we show a protocol where Prover aims at convincing Verifier that $\phi(x_1, \ldots, x_m)$ has precisely k satisfying assignments. We shall also demonstrate this protocol on our concrete instance $\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3)$ and $k = 4$.

Let us recall that Verifier can only perform a polynomial work (thus also the communicated messages between P and V can only be of polynomial size), and the protocol should be *complete* (this means that P has a possibility to convince V with 100% certainty when his claim is indeed correct) and *sound* (which means that P can fool V, i.e. convince him that an

incorrect answer is correct, only with negligible probability.)
(Remark. Later we return to these definitions in more detail.)

We note that the normal form of $p(x_1, x_2, \ldots, x_n)$, can be of exponential size (can you show an example?), and we thus cannot ask P to send, e.g., the normal form of polynomials $f_0()$, $f_1(x_1)$, $f_2(x_1, x_2)$, $\ldots$ to V. The idea is to ask P to send successively the normal forms of *one-variable polynomials* $f_0()$, $f_1(x_1)$, $f_2(r_1, x_2)$, $f_3(r_1, r_2, x_3)$, $\ldots$, $f_n(r_1, r_2, \ldots, r_{n-1}, x_n)$ where the random numbers $r_1, r_2, \ldots$ will be chosen by V. Of course, these numbers will be chosen from a finite domain, namely from a field $\mathbb{Z}_q$ for a (sufficiently large but feasible) prime q. But let us look at the protocol:

- P sends to V some prime $q > 2^n$ (of polynomial size in $n = \mathit{length}(\phi)$) and a value (which he claims to be) $f_0()$.

  V checks that really $q > 2^n$ and q is a prime, and that $f_0() = k$; he rejects if something fails. (Remark. Verifying that q is prime can be done by a randomized polynomial algorithm [or a less efficient deterministic polynomial algorithm]. Another possibility would be to require that Prover sends a (short) witness that q is a prime [stemming from a former proof (by Pratt) that the primality problem is in NP].)

  > In our example, assuming $\mathit{length}(\phi) = 7$, P sends, say, $q = 131$ and (claims) $f_0() = 4$.

- P sends V (the coefficients of the one-variable) polynomial $f_1(z)$. V checks that the degree is $\leq n$ and evaluates $f_1(0), f_1(1)$ and checks that $f_0() = f_1(0) + f_1(1)$. If OK, V chooses a random $r_1 \in \mathbb{Z}_q$ and sends to P. (For later use) he also computes $f_1(r_1)$.

  > In our example, P sends $f_1(z) = 2z + 1$. V evaluates $f_1(0) = 1$, $f_1(1) = 3$, and finds that really $f_0() = 4 = f_1(0) + f_1(1)$. He chooses, say, $r_1 = 38$. He computes $f_1(38) = 77$.

- P sends V (the coefficients of the one-variable) polynomial $f_2(r_1, z)$. V checks that the degree is $\leq n$ and evaluates $f_2(r_1, 0), f_2(r_1, 1)$ and checks that $f_1(r_1) = f_2(r_1, 0) + f_2(r_1, 1)$. Then V chooses a random $r_2 \in \mathbb{Z}_q$ and sends to P. He also computes $f_2(r_1, r_2)$.

  > In our example, P sends $f_2(38, z) = 38z^2 + 38z - z^2 + 1 = 37z^2 + 38z + 1$. V evaluates $f_2(38, 0) = 1$, $f_2(38, 1) = 76$, and finds that really $f_1(38) = 77 = f_2(38, 0) + f_2(38, 1)$. He then chooses, say, $r_2 = 105$. He computes $f_2(38, 105) = 37 \cdot 105^2 + 38 \cdot 105 + 1 = 52$ (modulo 131).

- P sends V (the coefficients of the one-variable) polynomial $f_3(r_1, r_2, z)$. V checks that the degree is $\leq n$ and evaluates $f_3(r_1, r_2, 0), f_3(r_1, r_2, 1)$ and checks that $f_2(r_1, r_2) = f_3(r_1, r_2, 0) + f_3(r_1, r_2, 1)$. Then V chooses a random $r_3 \in \mathbb{Z}_q$ and sends to P. He also computes $f_3(r_1, r_2, r_3)$.

  > In our example, P sends
  > $f_3(38, 105, z) = -38 \cdot 105^2 z + 38 \cdot 105 z + 38 \cdot 105^2 + 105^2 z - 105^2 - 2 \cdot 105 z + 105 + z = 122z + 96$ (modulo 131).

V evaluates $f_3(38, 105, 0) = 96$, $f_3(38, 105, 1) = 87$, and finds that really $f_2(38, 105) = 52 = f_3(38, 105, 0) + f_3(38, 105, 1)$.
He then chooses, say, $r_3 = 17$. He computes $f_3(38, 105, 17) = 122 \cdot 17 + 96 = 74$ (modulo 131).

- . . .
- The last phase follows after $r_1, r_2, \ldots, r_m$ have been chosen. V compares $f_m(r_1, r_2, \ldots, r_m)$ with $p_\phi(r_1, r_2, \ldots, r_m)$; he rejects when the values differ but he is convinced that $(\phi, k) \in \#SAT$ when the values are the same. (Remark. Note that V constructs $p_\phi(x_1, \ldots, x_m)$ quickly from $\phi(x_1, \ldots, x_m)$. He does not need to transform p_ϕ into the normal form; this might be useful for P but we do not care how P computes his messages.)

  In our example ($m = 3$) V has computed $f_3(38, 105, 17) = 74$, and now constructs $p(x_1, x_2, x_3) = (1 - (1 - x_1)(1 - (1 - x_2))) \cdot (1 - (1 - x_2)(1 - x_3))$ and then checks $p(38, 105, 17) = 74$.

Exercise.

- Check that Verifier performs only polynomial computation (in the size of the input (ϕ, k)).

- Show that if $(\phi, k) \in \#SAT$ then there is a (strategy of the) Prover which results in convincing Verifier.

- Now suppose that a (malicious) Prover wants to convince Verifier that $(\phi, k) \in \#SAT$ though it is not true. It is clear that P must start with sending (the claim that) $f_0() = k$ though it is not true (for 'the correct' f_0 determined by ϕ); let us say that P thus sends a wrong $\bar{f}_0()$. Later P must send (the coefficients of) polynomial $f_1(z)$; if he sends the correct $f_1(z)$ then V finds $f_1(0) + f_1(1) \neq k$ though P claimed $\underline{f}_0() = k$. So P must continue lying and sends some wrong $\bar{f}_1(z)$ (which satisfies $\bar{f}_1(0) + \bar{f}_1(1) = k$) instead. (Note that this means that $f_1(r) = \bar{f}_1(r)$ for at most n elements out of $\{0, 1, 2, \ldots, q{-}1\}$ [which contains more than 2^n elements].) If V now (randomly) chooses r_1 such that $f_1(r_1) \neq \bar{f}_1(r_1)$ then P must continue lying when sending $f_2(r_1, z)$: if he sends the correct $f_2(r_1, z)$, V finds that the value $f_2(r_1, 0) + f_2(r_1, 1)$ (which is the correct $f_1(r_1)$) differs from the previously computed $\bar{f}_1(r_1)$. So P must send some wrong $\bar{f}_2(r_1, z)$ instead . . . Continue in this reasoning and derive a bound on the probability that P convinces V that $(\phi, k) \in \#SAT$ though it is not the case.

Část 3

4 Zero-knowledge proofs

(E.g. in [7], [4].)

When we prove a (mathematical) assertion, the proof usually contains more information (and provides a deeper insight) than the plain fact that the assertion is true; this is also (usually) desirable. But there are contexts where a party, say Peggy [or Prover], should show to another party, say Victor [Verifier], that she has a secret piece of knowledge (like a password allowing an access to a source) but she does not want to disclose any additional information to him. (V should only get convinced that P is entitled to the access but he should not learn anything about P's password ...)

The described situation is modeled by special Interactive Protocols, called Zero-Knowledge (ZK) Interactive Protocols (together with the fact that the assertion holds, V gets the zero additional knowledge ...)

Graph isomorphism

A famous example is a ZK-protocol for problem GI (Graph-Isomorphism) (which is a problem in NP, with an unclear complexity status).

Exercise. Define GI precisely and show that it belongs to NPTIME.

Suppose P and V are given two graphs G_1, G_2 (both with n nodes and m edges ...), and P also knows an isomorphism $b : G_1 \to G_2$; we can view b as a permutation on $\{1, 2, \dots, n\}$ (why?). P wants to convince V that G_1, G_2 are isomorphic but without disclosing any information about a respective isomorphism. They perform the following protocol:

- P randomly chooses $i \in \{1, 2\}$ and a permutation π of $\{1, 2, \dots, n\}$, computes the graph $H = \pi(G_i)$ and sends H to V.
- V (randomly) selects $j \in \{1, 2\}$ and sends j to P.
- P determines a permutation σ such that $\sigma(G_j) = H$ and sends σ to V.
- V checks if really $\sigma(G_j) = H$; in the positive case V accepts, and otherwise rejects.

Exercise.

- How can P compute σ? Note that this can be done in polynomial time when P knows $b : G_1 \to G_2$.

- Check **completeness** of the protocol, which means that V surely accepts when $G_1 \overset{iso}{=} G_2$ and P follows (correctly) the protocol.

- Imagine G_1, G_2 are not isomorphic but a malicious P' tries to convince V that they are. Give a bound on the probability that P' succeeds.

- Use the standard trick (amplification by repeating) guaranteeing **soundness**, i.e., that any (malicious) P' can convince V about a wrong answer only with negligible probability.

Remark. It is not necessary to assume that P possesses $b : G_1 \to G_2$; P has generally unbounded computational power and she could thus compute such isomorphism b. But the assumption is useful here when we have practical implementations in mind - the Prover's work is then also polynomial.

The above given (P,V)-protocol for GI satisfies the following (strong) version of (a formal attempt to define) the zero-knowledge property:

Definition. A (P,V)-protocol (for showing $A \in IP$) is a *zero-knowledge protocol* if there is a polynomial-time probabilistic Turing machine M which for any input $x \in A$ outputs a tuple $(y_1, y_2, \ldots, y_k)$ of possible messages appearing on the communication channel when P and V work on x, and in such a way that the (probabilistic) distribution of the tuples in $M(x)$ (i.e. of the possible outputs of M working on x) is the same as the distribution of the tuples really appearing in the communication when P and V work on x.

In other words, if V gets convinced that $x \in A$ (by the correct P) then the information he gets from the interchanged messages is the same as would be provided by running 'the simulator' M on x.

Exercise. Suggest a simulator M which proves that the above protocol for GI is a zero-knowledge protocol.

Remark. In fact, the requirement that the probabilistic distributions (of the P-V communications and of the outputs of M) are the same is often weakened to a sort of computational indistinguishability.

Quadratic residues

(Based on an Internet text by Boaz Barak ...)

Here we show another example of a ZK-protocol.

For $n \in \mathbb{N}$, $x \in \mathbb{Z}_n^*$ is a *quadratic residue modulo* n if there is s $(0 \leq s < n)$ such that $s^2 \equiv x \pmod{n}$.

Exercise.

- Show that $s \in \mathbb{Z}_n^*$ when $s^2 \equiv x \pmod{n}$ for $x \in \mathbb{Z}_n^*$.
- Show that the set QR_n of the quadratic residues modulo n is a subgroup of $\mathbb{Z}_n^*$.
- Deduce that if x is a (fixed) residue and y a random residue then xy is a random residue.

Protocol QR:

- P and V share an input x, n, $gcd(x, n) = 1$,
  and P also knows (private) w such that $w^2 \equiv x \pmod{n}$.
- P chooses random $u \in \mathbb{Z}_n^*$ and sends $y = u^2 \bmod n$ to V.
- V chooses random $b \in \{0, 1\}$ and sends to P.
- If $b = 0$ then P sends $z = u$, otherwise P sends $z = wu \bmod n$.
- If $b = 0$ then V accepts iff $z^2 \equiv y \pmod{n}$, and if $b = 1$ then V accepts iff $z^2 \equiv xy \pmod{n}$.

The analysis is left (as an option) for the reader.

We just note that computing a square root s for a given x modulo n is (tightly related to factoring integers and thus also) believed to be computationally intractable (i.e., it is believed that every probabilistic polynomial time algorithm would succeed only with negligible probability).

Hamiltonian cycle

Imagine now that P and V have a graph G, with nodes $\{1, 2, \ldots, n\}$ and with m edges, and P wants to convince V that G has a Hamiltonian cycle. We can imagine that P knows the apropriate permutation $(i_1, i_2, \ldots, i_n)$ of $\{1, 2, \ldots, n\}$
(where all pairs $(i_1, i_2), (i_2, i_3), \ldots, (i_{n-1}, i_n), (i_n, i_1)$ are edges of G).

Consider the following ('physical') protocol:

- P randomly chooses a permutation π of $\{1, 2, \ldots, n\}$, computes the graph $H = \pi(G)$, and puts m paper pieces with edges of H upside down on the table. V can verify that the number of pieces is m but he does not see what is written on the pieces. (We say that P has made a *commitment*, she cannot change the pieces now.)
- V (randomly) selects one of two requests: "Show me that H is isomorphic to G" or "Show me that H has a Hamiltonian cycle".
- In the first case P turns over all pieces and shows the permutation π so that V can verify that H really is isomorphic to G. In the second case P turns over only pieces with edges $(\pi(i_1), \pi(i_2)), (\pi(i_2), \pi(i_3)), \ldots, (\pi(i_{n-1}), \pi(i_n)), (\pi(i_n), \pi(i_1))$ so that V can verify that H really contains a (simple) cycle with n nodes. (If V finds the P's answer correct, he accepts, otherwise he rejects.)

Exercise.
- Check the completeness and the soundness of the protocol (when repeated sufficiently many times).
- Argue why it is a zero-knowledge protocol.
- In fact, this protocol would not obey a formal definition defining P and V as communicating Turing machines. Suppose we want to implement it in such a way; specify where a substantial problem arises.

Before returning to the problem you noticed, we look more carefully at the notion of soundness of an Interactive Protocol. We have defined it as the property that P can convince V that $x \in A$ though, in fact, $x \notin A$ only with negligible probability. This is made more precise by the following definition:

A (P,V)-protocol for a set A ($A \in IP$) must satisfy

- *completeness*: for every $x \in A$, V always accepts when interacting with (the correct strategy) P;
- *soundness*: there is a polynomial p such that for every $x \notin A$ and every (potential, even malicious, strategy) P', V rejects with probability at least $1/p(|x|)$ (when interacting with P'). (We say that V rejects with non-negligible probability.)

Exercise.

- Why is such a definition of soundness sufficient, i.e., how can we then achieve that the probability of accepting a wrong input is really negligible? (Recall that we have to maintain that V performs only polynomial computation.) (We call a function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ negligible if for every polynomial $p(y)$ there is x_0 such that $\forall x \geq x_0 : f(x) < 1/p(x)$.)

- Recalling that the problem HC (Hamiltonian cycle) is NP-complete, it should be clear that the problem 3-COL (is a given graph 3-colourable?) [which is in NP, why?] has a ('physical') zero-knowledge protocol. Explain.

- Suggest a direct simple ('physical') zero-knowledge protocol for 3-COL. (Recall that it is sufficient to get the soundness in the above defined sense.)

Commitment schemes; one-way functions

(See, e.g. [2].)

Now we come to the problem you have certainly noticed above, namely to the problem of *commitment schemes*. We have easily implemented them in 'physical' protocols (where we can have 'sealed non-transparent envelopes') but it is not trivial to implement them in 'electronic communication setting' (i.e., in the model when P and V are (probabilistic) Turing machines communicating by messages on a special tape).

We can easily see that the basic problem can be formulated as the *bit commitment* problem: we need a way how P can choose a bit $b \in \{0, 1\}$ and be further bound to that choice (she can not change it later), so she surely must send a message C to V at that point; the message (commitment) C must determine b but V must not be able to learn b from C. Later (when required by the protocol) P can send a key which enables V to open (the 'locked box') C and read b.

To really work, such a scheme must satisfy several properties. We just touch on this subject in the following exercise.

Exercise.

- How would you formulate the *binding* property? (P is bound by her choice ...). You can confine yourself to the so called unconditional (perfect) binding. [There is also a version of computational binding.]

- How would you formulate the *hiding* property? (The value chosen by P is hidden so that V cannot learn it ...) Do it in both the unconditional sense and the computational sense.

- Why we can not have a scheme which is both unconditionally biding and unconditionally hiding?

- Suggest a way how you can use RSA to create a bit commitment scheme which is unconditionally binding and (believed to be) computationally hiding. (You can use the [nonobvious] fact that computing the least significant bit of m from the value $m^e \bmod n$ (in the RSA scheme) is 'as hard as' computing m.)

- Find definitions of so-called one-way functions and discuss their relations to commitment schemes.

5 Probabilistically Checkable Proofs (PCP)

(E.g., in [9], [6].)

Now we look at special interactive protocols, which play an important role in showing lower bounds for approximation algorithms (discussed later in our course).

We start with an important example. Recall that 3SAT is in NP (is NP-complete, in fact). Given a boolean formula $\phi(x_1, x_2, \ldots, x_n)$, in cnf with 3 literals in each clause, we can use (a description of) a truth assignment (to the variables $x_1, x_2, \ldots, x_n$) which makes ϕ true as a proof of its satisfiability. A standard way to verify the correctness of this proof is to read the whole proof, substitute the given values into ϕ and verify that ϕ is true for this assignment.

Consider now a special subproblem, for some (fixed) $\varepsilon > 0$:

> Problem: ε-robust 3SAT
>
> Instance: a formula $\phi(x_1, x_2, \ldots, x_n)$ as in 3SAT, with m clauses, which is either satisfiable or each nonsatisfying assignment does not satisfy at least εm clauses.
>
> Question: Is ϕ satisfiable?

Remark. We do not care now how to guarantee that the instances have the mentioned property; we just have the task to suggest a way of proof-verifying which will work for such instances.

Again, a truth assignment (for variables $x_1, x_2, \ldots, x_n$) can serve as a proof. But it is now not necessary to read the whole proof:

Verifier can choose a clause randomly (for which $O(\log m)$ bits are sufficent) and then ask the values of 3 bits in the proof, namely the truth values of the variables in the selected clause. If the clause is not satisfied, V rejects; if the clause is satisfied, V repeats choosing a clause randomly, etc. ... when he finds $\log_{1-\varepsilon} \frac{1}{2}$ times that the chosen clause is satisfied, V accepts.

Remark. V can thus wrongly accept only with probability $\leq \frac{1}{2}$. As usual, repeating the whole process, say, 50 times, we get this probability below $\frac{1}{2^{50}}$.

We have thus shown that the problem ε-robust 3SAT has a probabilistically checkable proof where the Verifier uses $O(\log n)$ random bits (here n means the size of input) and asks for $O(1)$ (i.e., a constant number of) bits from a proof. In a succint notation, ε-robust 3SAT belongs to $PCP(\log n, 1)$.

Generally, $PCP(r(n), q(n))$ denotes the class of problems (sets, languages) where for each L in the class there is an interactive protocol which can be presented as follows:

For each $x \in L$, P can be viewed as a string (a proof of the fact that $x \in L$) which determines the answers to Verifier's queries. Verifier can only use $O(r(size(x)))$ random bits and ask $O(q(size(x)))$ queries (and altogether works in polynomial time). (This allows to put an upper bound $2^{O(r(size(x))} \cdot O(q(size(x)))$ on the length of the proof.)

As usual, we require that for any $x \in L$ there is P such that (P,V) always accepts x (i.e., with probability 1), and for $x \notin L$ and any P' the protocol (P',V) accepts x with probability $\leq \frac{1}{2}$.

From the definition we easily get:

$PCP(0, 0) = P$

$PCP(poly(n), 0) = co{-}RP$

$PCP(0, poly(n)) = NP$

Remark. By $PCP(poly(n), 0)$ we mean the union of all $PCP(r(n), 0)$ where r is a polynomial function, etc.
Recall also that RP is the class of problems (languages) such that for each $L \in RP$ there is a polynomial time randomized (probabilistic) algorithm such that $x \in L$ is accepted with probability $\geq \frac{1}{2}$ and $x \notin L$ is rejected with probability 1.

The main results regarding PCP were shown during 1990s. One of them says that

$PCP(poly(n), poly(n)) = NEXPTIME$.

Exercise. Show that $PCP(r(n), q(n)) \subseteq NTIME(poly(n) \cdot 2^{O(r(n))})$, and derive that $PCP(\log n, poly(n)) = NP$ and $PCP(\log n, 1) \subseteq NP$.

The following result, also known as the PCP-theorem, came as a surprise and has a nontrivial technical proof:

$NP = PCP(\log n, 1)$.

Exercise. Prove the PCP theorem when assuming that ε-robust 3SAT is NP-complete (for some $\varepsilon > 0$).

In fact, ε-robust 3SAT really is NP-complete but the known proof of this fact uses the PCP theorem (which is thus proved in another way, of course).

Ideas of a proof of a weaker version of the PCP theorem

The PCP theorem says that $NP = PCP(\log n, 1)$. We sketch the ideas for the following weaker version: $NP \subseteq PCP(n^3, 1)$; this is equivalent to the following claim.

Claim. 3-SAT $\in PCP(n^3, 1)$.

We start with the arithmetization (of formulas = instances of 3-SAT) illustrated on the following example:

To the formula $\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$
(with clauses $c_1, c_2, \ldots, c_m$ for $m = 2$, and variables $x_1, x_2, \ldots, x_n$ for $n = 4$)

we construct the polynomial $p_\phi = p_1 + p_2 = (1 - x_1) \cdot x_2 \cdot (1 - x_3) + x_1 \cdot x_2 \cdot (1 - x_4)$

(which is equivalent to
$x_2 - x_1x_2 - x_2x_3 + x_1x_2x_3 + x_1x_2 - x_1x_2x_4 = x_1x_2x_3 - x_1x_2x_4 - x_2x_3 + x_2$).

Note that generally the degree of p is 3 (here the degree of a term is the sum of the degrees of its variables) and there are $m = O(n^3)$ clauses (and thus $m = O(n^3)$ polynomials p_i) when we have n variables $x_1, x_2, \ldots, x_n$ (we do not allow repeating of clauses).

We will be computing in the field $\mathbb{Z}_2 = (\{0, 1\}, +_{\text{ mod } 2}, \cdot_{\text{ mod } 2})$; then, e.g., our p_ϕ above is equivalent to $x_1x_2x_3 + x_1x_2x_4 + x_2x_3 + x_2$.

Suppose now a (truth) assignment $a = (a_1, a_2, \ldots, a_n) \in \{0, 1\}^n$. The following exercise presents an important idea.

Exercise. Show that for a random $r = \{r_1, r_2, \ldots, r_m\} \in \{0, 1\}^m$ we have that the probability $Prob(r \cdot p(a) = r_1p_1(a) + r_2p_2(a) + \cdots + r_mp_m(a) = 0)$ is 1 when a satisfies ϕ and 1/2 when a does not satisfy ϕ.

This suggest a way for V (Verifier) how to convince himself that a satisfies ϕ (if it is the case) but he needs to know the whole $a = (a_1, a_2, \ldots, a_n)$ for evaluation. We now look for an encoding of a which enables the needed evaluation by just reading a few bits of the encoding (i.e, of the proof).

For $a = (a_1, a_2, \ldots, a_n)$ we define the following linear functions

$L_1^a : \mathbb{Z}_2^n \to \mathbb{Z}_2$; $L_1^a(y_1, \ldots, y_n) = \sum_{1 \le i \le n} a_i y_i$

$L_2^a : \mathbb{Z}_2^{n^2} \to \mathbb{Z}_2$; $L_2^a(y_{1,1}, \ldots, y_{n,n}) = \sum_{1 \le i,j \le n} a_i a_j y_{i,j}$

$L_3^a : \mathbb{Z}_2^{n^3} \to \mathbb{Z}_2$; $L_3^a(y_{1,1,1}, \ldots, y_{n,n,n}) = \sum_{1 \le i,j,k \le n} a_i a_j a_k y_{i,j,k}$

(Example. In the case $n = 3$, $a = (1, 0, 1)$ we get

$L_1^a(y_1, y_2, y_3) = y_1 + y_3$
$L_2^a(y_{1,1}, \ldots, y_{3,3}) = y_{1,1} + y_{1,3} + y_{3,1} + y_{3,3}$
$L_3^a(y_{1,1,1}, \ldots, y_{3,3,3}) = y_{1,1,1} + y_{1,1,3} + \cdots)$

The desired proof (encoding) for a will consist of the function tables for L_1^a, L_2^a, L_3^a (and will thus have length $2^n + 2^{n^2} + 2^{n^3}$).

Now, when V wants to compute $r \cdot p(a)$, he can transform the polynomial $q = r \cdot p$ into the normal form,

in our example, for $r = (1, 1)$ we get $q_{11} = x_1x_2x_3 + x_1x_2x_4 + x_2x_3 + x_2$, for $r = (0, 1)$ we get $q_{01} = x_1 \cdot x_2 \cdot (1 - x_4) = x_1x_2 + x_1x_2x_4$ (recall that we work over $\mathbb{Z}_2$), etc.,

and computes the appropriate characteristic vectors of the subpolynomials of degrees 0,1,2,3, denoted $v_0 \in \mathbb{Z}_2$, $v_1 \in \mathbb{Z}_2^n$, $v_2 \in \mathbb{Z}_2^{n^2}$, $v_3 \in \mathbb{Z}_2^{n^3}$; in the case of the above q_{11} we get $v_0 = 0$, $v_1 = (0, 1, 0, 0)$, since the subpolynomial of degree 1 is x_2, v_2 contains one 1 at position $(2, 3)$, since the subpolynomial of degree 2 is x_2x_3, and v_3 contains two 1's at positions $(1, 2, 3)$, $(1, 2, 4)$ since the subpolynomial of degree 3 is $x_1x_2x_3 + x_1x_2x_4$.

Exercise. Formalize this process and show that V thus can use just three bits of the proof for computing $q(a)$, since $q(a) = v_0 + L_1^a(v_1) + L_2^a(v_2) + L_3^a(v_3)$.

In fact, the work of V must be enhanced to cope with wrong proofs. He has to verify (with sufficient certainty) that the given proof really contains tables of linear functions (which satisfy $f(a + b) = f(a) + f(b)$) and that they are consistent (i.e., contain tables $L_1^a, L_2^{a'}, L_3^{a''}$ for the same $a = a' = a''$). There is also a subtle point that the tested L can be only "almost linear" and V does not find this; he handles this by a certain "correcting the proof" … One has to do some technical work, involving a detailed computation of probabilities, to show that these tasks are really realizable and thus that there really is an appropriate protocol using $O(n^3)$ random bits and a constant number of proof-queries.

Remark There would be a lot of work to finish our proof and then to strengthen it to really yield the PCP theorem. Let us just note that a more direct proof has been found by Irit Dinur [3].

Část 4

6 Approximation algorithms

Optimization problems

We recall what is an *algorithmic problem* (a *problem* for short): it is defined by

- a description of the set of allowable inputs, with their representation as finite sequences over a finite alphabet (sequences of bits, i.e., elements of $\{0,1\}^*$ in particular),
- a description of a function that maps each (allowable) input to a non-empty set of correct outputs (answers, results, solutions), each of which is also a finite sequence over a finite alphabet (like $\{0,1\}$).

In *decision problems* each input is mapped to an element of $\{0,1\}$ (i.e. $\{false, true\}$). In *optimization problems* the desired outputs satisfy some optimality criterion (an example is searching for a *shortest* path between two nodes in a graph), in *approximation problems* the outputs can just approximate an optimal solution.

An algorithm solving a particular problem is required to output just one of the prescribed solutions (answers), not to list all of them.

Remark. We can also speak about *evaluation problems* when the required output is the value of a cost function (like the length of a shortest path), etc.

MinSpanTree is in P (=PTIME) [or in PO]

Exercise. Recall the problem MinSpanTree (the task is to construct a minimal spanning tree for an edge-labelled graph), and a polynomial algorithm for solving the problem. (A greedy approach works in this case. Can you prove this?)

Remark. We thus say that MinSpanTree is in PTIME, though we usually view PTIME as a set of decision problems (i.e., a set of languages). More precise is to use the notation like PO (the class of optimization problems solvable in polynomial time) or so ...

TSP is NP-equivalent

Recall the problem TSP (Travelling Salesman Problem), which is also an optimization problem where a minimal solution (a solution with the minimal value of a cost function) is searched. (You can construct an example showing that the greedy approach does not work in this case.)

Recall the class NP (i.e., NPTIME) and the notions of NP-hardness and NP-completeness. In the definitions, the notion of *polynomial reduction* $\leq_P$ is used.

Exercise. E.g., recall the construction showing Sat $\leq_P$ IndSet where Sat is the problem of satisfiability of boolean formulas (in cnf, conjunctive normal form) and IndSet is the decision variant of the optimization problem MaxIndSet of searching a maximal independent set (anticlique) in a graph.

IndSet can be further reduced to the problem HamCycle (Hamiltonian cycle in undirected graphs). (It is useful to use the analogous problem for directed graphs as an intermediate step.) HamCycle can be then used to show that TSP$_{dec}$, the decision version of TSP, is NP-complete.

Polynomial reductions ($\leq_P$) are a special case of more general *Turing reductions* ($\leq_T$). We have $P_1 \leq_T P_2$ if there is a polynomial algorithm which solves P_1 by using some calls of an *oracle* (a hypothetical procedure) for P_2; the procedure for P_2 is viewed as returning the answer immediately, i.e., its call costs just one unit of time.

Remark. In fact, Turing reductions are generally used in computability theory without the polynomiality assumption. If a confusion might arise, we can write $\leq_T^p$ to stress this polynomiality assumption. Here we implicitly assume polynomiality.

We can easily observe that $\leq_P$ is indeed a special case of $\leq_T$: $P_1 \leq_P P_2$ iff there is a polynomial algorithm solving P_1 which calls (an oracle for) P_2 just once, at the end, and returns the answer provided by this call of P_2. Polynomial reductions are useful for getting a finer view of complexity classes but it is sometimes sufficient and convenient to consider (the more general) Turing reductions.

When we say that a general problem (e.g., an optimization problem) O is $\mathcal{C}$-hard (where $\mathcal{C}$ is a class of problems, like, e.g., NP) we mean $\forall P \in \mathcal{C} : P \leq_T O$. O is $\mathcal{C}$-easy when $\exists P \in \mathcal{C} : O \leq_T P$. O is $\mathcal{C}$-equivalent when O is $\mathcal{C}$-easy and $\mathcal{C}$-hard.

Remark. Thus, e.g., UnSat (the complement of Sat) is NP-equivalent when we assume Turing reductions, since every decision problem (i.e., every language) is Turing-reducible to its complement. (How?) Of course, when we say that UnSat is coNP-complete, we implicitly refer to polynomial reductions. (The type of considered reductions must be specified when a misunderstanding can arise.)

Exercise. Explain why is TSP, i.e. TSP$_{opt}$ (the optimization variant), NP-equivalent. (Recall that HamCycle $\leq_P$ TSP$_{dec}$.)

TSP is not in APX (even not in APX(2^n))

If we have an optimization problem which is NP-hard (often NP-equivalent), a polynomial algorithm can only approximate an optimal solution, when we assume the general (though unproven) belief that P≠NP.

Remark. In fact, we can also think about polynomial *probabilistic* algorithms for approximation. (Later we show an example.) But this does not seem to help substantially: the class BPP (the set of problems solvable by probabilistic polynomial algorithms with bounded error probability), which is a 'still practical' superset of P, is also believed not to contain any NP-hard problem.

Let us have an optimization problem P, which attaches a set of feasible solutions $FS(x)$ to each input x and specifies a cost function $v(x, s)$ (where $s \in FS(x)$) which should be minimized (or maximized). We say that A is an approximation algorithm for P if it is a (deterministic) polynomial algorithm and for each input x returns a solution $s_A(x) \in FS(x)$. We say that A has the *approximation ratio* (at most) $r_A(n)$ (where $r_A : \mathbb{N} \to \mathbb{N}$) if

- $\frac{v(x,s_A(x))}{v_{opt}(x)} \leq r(size(x))$ in the case of minimization, and

- $\frac{v_{opt}(x)}{v(x,s_A(x))} \leq r(size(x))$ in the case of maximization.

(As expected, $v_{opt}(x)$ denotes the value $v(x,s)$ of an optimal solution s.)

By

$$\mathrm{APX}(r(n))$$

we denote the class of optimization problems for which there are approximation algorithms with the approximation ratio $\leq r(n)$. By

$$\mathrm{APX}$$

we denote the union of all classes APX(c) where $c \in \mathbb{N}$ (the approximation ratio is bounded by a constant). The next claim implies that TSP is not in APX if P$\neq$NP.

Claim. If P$\neq$NP then there is no (polynomial) approximation algorithm for TSP with the approximation ratio $\leq 2^n$.

This is shown by using a *gap technique*:

Given the NP-complete problem HamCycle, we want to show a polynomial algorithm which, given an instance G of HamCycle, constructs an instance G' of TSP such that:

- if there is a Hamiltonian cycle in G then the (value of the) optimal solution in G' is 'small' ($\leq n$ in our case, where n is the number of vertices of G'),
- if there is no Hamiltonian cycle in G then the (value of the) optimal solution in G' is 'large' ($> n2^n$ in our case).

An approximation algorithm with ratio $\leq 2^n$ could thus be used to decide HamCycle (how?).

The construction with the above properties is simple: G' arises from G by giving each edge value 1, and by completing with the edges nonpresent in G, these with value $n2^n$.

This rather negative result does not hold for TSP$^{\Delta}$, the (frequent) subproblem of TSP where the distances satisfy the triangle inequality. We can easily show that TSP$^{\Delta}$ is in APX(2): we construct a minimal spanning tree and return the nodes in the order in which they are visited by the depth-first search of this tree. (In fact, it is known that TSP$^{\Delta}$ belongs to APX($\frac{3}{2}$).)

A use of (de)randomization; Max-3-Sat is in APX(8/7)

Let us consider the problem Max-3-Sat. An instance is a boolean formula $\phi(x_1,\dots,x_n)$ in cnf, where each clause contains 3 literals; the task is to find a truth-assignment (for variables $x_1,\dots,x_n$) which satisfies the maximal number of clauses. We assume $\phi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$; so m denotes the number of clauses.

Exercise. Show that Max-3-Sat is NP-equivalent. (NP-hardness can be easily shown by Sat $\leq_T$ Max-3-Sat; but can you also show Max-3-Sat $\leq_T$ Sat ?)

There is a simple approximation algorithm A for Max-3-Sat with $r_A \leq 2$ (so Max-3-Sat belongs to the class APX(2)):

> Evaluate the number of satisfied clauses for the truth assignments $(0,0,\dots,0)$ and $(1,1,\dots,1)$ and return the one which satisfies more.

The desired property follows from the observation that if clause c_i is not satisfied by $(0, 0, \ldots, 0)$ then it is surely satisfied by $(1, 1, \ldots, 1)$; hence one of the assignments satisfies at least $\lceil \frac{m}{2} \rceil$ clauses, which guarantees

$\lceil \frac{m}{2} \rceil \leq v(\phi, s_A(\phi)) \leq v_{opt}(\phi) \leq m$, and thus $\frac{v_{opt}(\phi)}{v(\phi, s_A(\phi))} \leq 2$.

Remark. We have thus also illustrated a frequent scheme for estimating approximation ratios. In the case of maximization problems, we try to find a suitable upper bound for the optimal value (since we seldom are able to express this value exactly) and a lower bound for the computed approximation value. In the minimization problems, we look for a suitable lower bound for the optimum and an upper bound for the approximation.

Let us note that if we take a (uniformly distributed) *random truth assignment* (for $x_1, x_2, \ldots, x_n$) in ϕ, where we assume that all three variables in each clause are different, then the expected value $E(Y)$ of (the random variable counting) the number of satisfied clauses is $\frac{7}{8}m$ (why?). (Hint. Recall that the expected value is linear, so $E(Y) = E(Y_1 + Y_2 + \cdots + Y_m) = E(Y_1) + E(Y_2) + \cdots + E(Y_m)$ where $E(Y_i)$ is the expected value of the random variable which is 1 if the clause c_i is satisfied and 0 if c_i is not satisfied.)

Exercise. Try to derandomize the probabilistic algorithm based on this observation, i.e., suggest a deterministic polynomial algorithm approximating MAX-3-SAT with the ratio $\leq \frac{8}{7}$. (Hint. Use the fact that $E(Y) = \frac{1}{2} \cdot E(Y|x_1 = 0) + \frac{1}{2} \cdot E(Y|x_1 = 1)$, and thus necessarily at least one of the values $E(Y|x_1 = 0)$, $E(Y|x_1 = 1)$ is $\geq \frac{7}{8}m$.)

MinVertexCover is in APX(2)

Another NP-equivalent problem is the vertex cover problem MINVERTEXCOVER: given an (undirected) graph $G = (V, E)$, find a minimal set $C \subseteq V$ of vertices such that each edge $(v, v') \in E$ is 'covered', i.e., has at least one vertex in C ($\{v, v'\} \cap C \neq \emptyset$).

Remark. Note that $\bar{C} = V - C$ is an independent set (i.e., an anticlique; there is no edge between two nodes in $\bar{C}$). For later use you can show that INDSET $\leq_p$ VERTEXCOVER (where VERTEXCOVER is the decision version of MINVERTEXCOVER).

Exercise. Show that the following greedy approach for MINVERTEXCOVER, "repeatedly choose a vertex which covers the maximal number of sofar uncovered edges", does not have the approximation ratio ≤ 2; you can even show that the ratio is not $\leq c$ for any $c \in \mathbb{N}$.

There is the following 2-approximation algorithm A for MINVERTEXCOVER:

> construct a maximal matching, i.e. a maximal set of pairwise disjoint edges; the set of vertices incident with those edges constitute a vertex cover which is at most 2 times bigger than a minimal one. (Why? You will probably show that $z \leq v_{opt}(G) \leq v(G, s_A(G)) \leq 2z$ for some z.)

MinSetCover is in APX$(1 + \ln n)$

The set covering problem MINSETCOVER is formulated as follows.

Informally: we have a set of skills to be covered, and a set of people, each of which has some skills. Find a minimal 'committee' covering all skills (supposing there is some).

More formally, we have a set X (of 'skills'), a set $\mathcal{F}$ of subsets of X (elements of $\mathcal{F}$ are 'people') such that the union of all elements of $\mathcal{F}$ is X.

Exercise. Show that MinSetCover can be easily viewed as a generalization of MinVertexCover but we have no good analogue of the above 'maximal matching' approximation algorithm in this case.

The greedy heuristics,

> repeatedly select a set (element of $\mathcal{F}$) covering the maximum number of the sofar uncovered elements of X

gives an approximation ratio $O(\log n)$.

Exercise. Show this claim; you can elaborate the approach hinted below.
The approximation ratio can be shown bounded by $H(max\{|S|; S \in \mathcal{F}\})$, where $H(n) = \sum_{i=1}^{n} \frac{1}{i}$ is the n-th harmonic number.
(Recall that $H(n) < 1 + \ln n$ [using the fact $\int \frac{1}{x} dx = \ln x$].)
The greedy algorithm constructs a cover $\mathcal{C} = \{S_1, S_2, \dots\}$; each newly added set can be viewed as adding cost 1 to the solution (whose cost is $|\mathcal{C}|$). We spread this cost on the newly covered elements: each element x which is first covered by S_i has cost $c_x = 1/|\, S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})\,|$.

So $|\mathcal{C}| = \sum_{x\in X} c_x$; this is $\leq \sum_{S\in\mathcal{C}^*} \sum_{x\in S} c_x$, where $\mathcal{C}^*$ is an optimal solution.

So we will be done when we show that $\forall S \in \mathcal{F} : \sum_{x\in S} c_x \leq H(|S|)$
(since then $|\mathcal{C}| \leq |\mathcal{C}^*| \cdot H(max\{|S|; S \in \mathcal{F}\})$).

So fix some $S \in \mathcal{F}$ and let $u_i = |\, S - (S_1 \cup S_2 \cup \cdots \cup S_i)\,|$.

Due to the greedeness: $u_{i-1} \leq |\, S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})\,|$.

Let k be the smallest s.t. $u_k = 0$.

$\sum_{x\in S} c_x = \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{|\, S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})\,|} \leq \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{u_{i-1}}$.

Since $(b-a) \cdot \frac{1}{b} \leq \sum_{i=a+1}^{b} \frac{1}{i} = H(b) - H(a)$, we have $\sum_{x\in S} c_x \leq \sum_{i=1}^{k} (H(u_{i-1}) - H(u_i))$.
Telescoping sum $\ldots = H(u_0) - H(0) = H(u_0) = H(|S|)$.

(The above proof is a straightforward but a bit technical calculation. Less technical can be to proceed by induction on $|S|$ [or by supposing a counterexample which has the minimal $|S|$]; it can be also useful to strengthen the claim: $\forall S \in \mathcal{F} : \forall S' \subseteq S : \sum_{x\in S'} c_x \leq H(|S'|)$ [where we put $H(0) = 0$].)

A use of linear programming; MinWeightVertexCover is in APX(2)

Given $G = (V, E)$ and a weight function $w : V \to \mathbb{N}_+$, we search for a vertex cover $C \subseteq V$ such that $\sum_{v\in C} w(v)$ is minimal. This is a generalization of MinVertexCover (why?), let us denote this problem as MinWeightVertexCover.

Exercise. Show that an approach analogous to the previous 2-approximation algorithm for MinVertexCover can give results which are far from optimal. (It is also not clear how a randomization could help.)

Let us reformulate our problem. In fact, we are searching for a certain function (or a $|V|$-dimensional vector) $x : V \to \{0, 1\}$, defining a set $C \subseteq V$ by $v \in C \Leftrightarrow x(v) = 1$.

We are thus solving the following integer linear programming problem:

we search for an *integer* solution x satisfying

$\forall v \in V : 0 \leq x(v) \wedge x(v) \leq 1$, and

$\forall (v, v') \in E : x(v) + x(v') \geq 1$

which minimizes $\sum_{v \in V} x(v)w(v)$.

Of course, the IntLinProg (integer linear programming) problem is also NP-hard (NP-equivalent, in fact) but when we relax the 'integer' requirement, i.e., we search for an optimal real (in fact, rational) solution, we get an instance of LinProg (linear programming) which is solvable in polynomial time.

Remark. The algorithms for LinProg used in practice are based on the so-called simplex method, and have exponential worst-case complexity (for artificially composed instances, not for those arising in practice). The question of polynomiality of LinProg was a long-standing open problem until answered positively by Khachiyan in 1979.

Our algorithm thus finds an optimal solution $\bar{x}$ (of the LinProg-instance), and then it rounds the components to 0,1, i.e., puts $v \in V$ into C iff $\bar{x} \geq 1/2$.

Exercise. Show that the resulting algorithm is indeed a 2-approximation algorithm for MinWeightVertexCover.

Remark. (The nontrivial result on polynomiality of) LinProg is a convenient tool to show that MinWeightVertexCover is in APX(2). Nevertheless, one can suggest direct 2-approximation algorithms, without using LinProg.
Student Martin Šustek reported [in Nov 2009] the following algorithm.
Process all edges (of the given graph $G = (V, E)$) successively: to each edge e attach the maximal weight $w'(e)$ so that you maintain the invariant that $w(v)$, for each $v \in V$, is greater or equal to the sum of weights of the (processed) edges incident with v. Finally return the set V' of the "saturated" vertices (i.e., those $v \in V$ for which $w(v)$ is equal to the sum of weights of the incident edges).

Fully polynomial approximation scheme for MaxSubsetSum

Recall the problem MaxKnapsack: given a set $\{1, 2, \ldots, n\}$ of objects, each i with utility $u(i)$ and weight $w(i)$, and a weight limit W, find a subset $I \subseteq \{1, 2, \ldots, n\}$ of objects so that $\sum_{i \in I} u(i)$ is maximal on condition that $\sum_{i \in I} w(i)$ does not exceed W.

Exercise. Show a *pseudopolynomial algorithm* solving this problem. I.e., given a polynomial $p(n)$, suggest a polynomial algorithm working on instances x in which the (values of the) given numbers are bounded by $p(size(x))$.

SubsetSum is a special case of the decision version of Knapsack: given positive numbers $a_1, a_2, \ldots, a_n$ and a limit t, we ask if there is a subset $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} a_i = t$.

Exercise. Show that SubsetSum is NP-complete. (You can use 3-Sat for reduction.)

Remark. Note that the numbers in the SubsetSum-instances you constructed when showing 3-Sat $\leq_p$ SubsetSum are not polynomial (in the size of instance). (Otherwise you would have shown P=NP. Why?) The problems which are NP-complete even when restricted to instances with polynomially bounded numbers are sometimes called *strongly NP-complete.* Thus SubsetSum (or Knapsack) are not stronly NP-complete; on the other hand, e.g., TSP is strongly NP-complete. (Why?)

We now show that MaxSubsetSum (an optimization version of SubsetSum; we search for the largest sum $\sum_{i\in I} a_i$ not exceeding t) belongs to APX, but not only that: we show that MaxSubsetSum belongs to APX(c) for all $c > 1$ (i.e., to the class sometimes denoted APX*). In fact, our result will be still stronger: we show that MaxSubsetSum belongs to the class

$$\text{PTAS}\,(\subseteq \text{APX}^*)$$

of the optimization problems which have *polynomial-time approximation schemes* (*ptas* for short).

A *ptas* for an optimization problem O is an algorithm A which gets as input an instance x of O *and* a rational $\varepsilon > 0$; it computes a (feasible) solution $s_A(x, \varepsilon)$ guaranteeing the approximation ratio $\leq (1 + \varepsilon)$.
(I.e., $\frac{v(x,s_A(x,\varepsilon))}{v_{opt}(x)} \leq (1 + \varepsilon)$ in the case of a minimization problem, and $\frac{v_{opt}(x)}{v(x,s_A(x,\varepsilon))} \leq (1 + \varepsilon)$ in the case of a maximization problem.)
Moreover, for any fixed ε, (the resulting restriction of) A must be polynomial (in $size(x)$).

To broaden our view, we will show a ptas for the following problem which is tightly related to MaxSubsetSum. We denote it MinLoadScheduling (for 2 identical processors): an instance x is a list of tasks $1, 2, \ldots, n$ with times (durations) $d_1, d_2, \ldots, d_n$ and we want to distribute them to 2 processors so that the computation time is minimal. In other words, we search for a subset $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i\in I} d_i$ is minimal but satisfies $\sum_{i\in I} d_i \geq \frac{\ell}{2}$ where (the overall load) $\ell = d_1 + d_2 + \cdots + d_n$. (The tasks from I are then scheduled for one processor, the rest is scheduled for the other processor.)

Exercise. Formulate precisely how is this problem related to MaxSubsetSum.

We suggest the following ptas A for MinLoadScheduling. The algorithm A, given $d_1, d_2, \ldots, d_n$ and ε, classifies all tasks with $d_i \geq \varepsilon\ell$ as *large* (where $\ell = d_1 + d_2 + \cdots + d_n$); the tasks with $d_i < \varepsilon\ell$ are *small.* We note that there are at most $\lfloor\frac{1}{\varepsilon}\rfloor$ large tasks.

A then checks successively each of the at most $2^{\lfloor\frac{1}{\varepsilon}\rfloor}$ distributions of large tasks (to 2 processors); each particular distribution is completed by distributing small tasks 'greedily', i.e., each small task is then successively scheduled for the currently less loaded processor. A best solution is finally returned as the result.

Exercise. Check that the complexity of A is $O(2^{\lfloor\frac{1}{\varepsilon}\rfloor} \cdot n)$, and that the approximation ratio is $\leq (1+\varepsilon)$. (Hint. An optimal solution distributes large tasks in a certain way, and this way was also considered by A. Either all small tasks are in that case scheduled for one processor whose load is not greater than the load of the other, and A has thus found an optimal solution, or $\frac{\ell}{2} \leq v_{opt}(x) \leq v(x, s_A(x, \varepsilon)) \leq \frac{\ell}{2} + \frac{\varepsilon\ell}{2} = (1 + \varepsilon)\frac{\ell}{2}$.)

The appearance of $\frac{1}{\varepsilon}$ in an exponent is unpleasant. Can we do better? In this case yes; we switch back to MaxSubsetSum and show that the problem belongs even to

$$\text{FPTAS},$$

the class of the optimization problems which have *fully polynomial-time approximation schemes* (*fptas* for short).

An *fptas* is a ptas whose complexity is polynomial both in the size of the input ($size(x)$) and in (the value) $\lfloor\frac{1}{\varepsilon}\rfloor$. (E.g., the case $O((\frac{1}{\varepsilon})^2(size(x))^3)$ falls in this category.)

To show that MaxSubsetSum has a fptas B, we need to use additional ideas (and technical calculations). Consider an instance $a_1, a_2, \dots, a_n$ and t; we search for a subset $I \subseteq \{1, 2, \dots, n\}$ such that $\sum_{i \in I} a_i$ is maximal but $\leq t$.

Recalling the strategy of the pseudopolynomial algorithm for MaxKnapsack, it is clear that we can perform a cycle for $i = 1, 2, \dots, n$ and maintain an ordered list $L = \langle y_1, y_2, \dots \rangle$ of values $\leq t$ so that after the ith run L contains all possible values $\sum_{j \in I} a_j$ for subsets $I \subseteq \{1, 2, \dots, i\}$. We can do this by performing $L_i := merge(L_{i-1}, L_{i-1} + a_i)$, where $L + a$ arises from L by adding a to all elements of L; we omit the values $> t$ in our case.
Since now we have no polynomiality assumption on the values t and $a_1, a_2, \dots, a_n$, this algorithm can be exponential. The key idea of the algorithm B is to trim L after each run of the cycle, in the following way: if we include y_0 to L_i then we do not include further elements z in the range $y_0 < z \leq (1 + \frac{\varepsilon}{2n})y_0$ (if any); after including the first $y_1 > (1 + \frac{\varepsilon}{2n})y_0$, we do not include elements z in the range $y_1 < z \leq (1 + \frac{\varepsilon}{2n})y_1$, etc.

Exercise. Show that the suggested algorithm B is indeed a fptas for MaxSubsetSum. (It might be useful to note that $(1 + \frac{a}{n})^n < e^a$ [$a \geq 0$, e is Euler's number, i.e., the base of natural logarithm; in fact, $\lim_{n \to \infty}(1 + \frac{a}{n})^n = e^a$]. For $a \leq 1$ we also have $1 + a \leq e^a \leq 1 + a + a^2$.)

Approximation algorithms - some additional words

Let us survey some known results about the approximation ratios obtainable for some problems (by polynomial approximation algorithms); we have touched on some of them.

- $O(\frac{n}{\log^2 n})$ for MaxClique
- $O(\log n)$ for MinSetCover
- 2 for MinVertexCover
- $\frac{3}{2}$ for MinTSP$_\Delta$
- $\frac{8}{7}$ for Max-3-Sat
- PTAS for MinVertexCover$_{planar}$
- FPTAS for MaxKnapsack

These results provide some upper bounds.

Remark. These upper bounds relate to the worst-case aproximation ratio; sometimes the *asymptotic* worst-case approximation ratio is a better measure, but we do not discuss this topic here.

As a lower bound, we have shown

- MinTSP is not in APX(2^n) (if P≠NP).

This result was easy but it is generally more difficult to get some nontrivial lower bounds. For many optimization problems we can get at least weak inapproximability (i.e., lower bound) results in the following way.

Let us say that an optimization problem O has *small solution values* if the values of solutions are positive integers bounded by a polynomial in the length of the input. (Examples are scheduling problems, covering problems, clique problems, and also problems like MinTSP, MaxKnapsack when we restrict ourselves to the instances with number values bounded by a polynomial in the input size.)

Theorem. If P$\neq$NP then no NP-hard problem with small solution values belongs to FPTAS.

Proof. (By contradiction.) Suppose an NP-hard problem O with small solution values, bounded by a polynomial $p(n)$, and with an fptas A. The algorithm A' which on input x sets $\varepsilon := \frac{1}{p(size(x))}$ and then computes $A(x, \varepsilon)$ is a polynomial approximation algorithm for O with the approximation ratio $r_{A'}(n) \leq 1 + \frac{1}{p(n)}$ (why?). Since we assume P$\neq$NP, A' does not compute the optimal solutions for all inputs; but any returned nonoptimal solution (for an input of size n) shows $r_{A'}(n) \geq \frac{p(n)}{p(n)-1} = 1 + \frac{1}{p(n)-1} > 1 + \frac{1}{p(n)}$, which is a contradiction. □

Theorem. If P$\neq$NP, and for a minimization problem O with integer solution values it is NP-hard to decide whether $v_{opt}(x) \leq k$ (for a fixed k) then O does not belong to APX($r(n)$) for any $r(n) < 1 + \frac{1}{k}$; let us write APX($< 1 + \frac{1}{k}$) for short.
Similarly for a maximization problem O and deciding whether $v_{opt}(x) \geq k + 1$.

Proof. Suppose an algorithm A which shows that $O \in$ APX($< 1 + \frac{1}{k}$). Given some x with $v_{opt}(x) \leq k$, the algorithm A cannot output a solution s with $v(x, s) \geq k + 1$ since $\frac{v(x,s)}{v_{opt}(x)} \geq \frac{k+1}{k} = 1 + \frac{1}{k} > r_{A'}(size(x))$; A can thus be used to decide whether $v_{opt}(x) \leq k$ (how?), which contradicts with P$\neq$NP. □

As a corollary, we can derive, e.g.,

- MinGraphColour is not in APX($< \frac{4}{3}$) (if P$\neq$NP).

(It is not the best known lower bound for MinGraphColour, though.)

Recall the polynomial reductions $\leq_p$ among the decision problems. It is possible to suggest finer reductions $\leq_{PTAS}$ among the optimization problems, guaranteeing that if $P \leq_{PTAS} Q$ and $Q \in$ PTAS ($Q \in$ APX) then $P \in$ PTAS ($P \in$ APX).

We will not give a technical definition but only mention that one can show that

- Max-3-Sat $\leq_{PTAS}$ MaxClique
- MaxClique $\equiv_{PTAS}$ MaxIndSet (where $A \equiv_{PTAS} B$ iff $A \leq_{PTAS} B$ and $B \leq_{PTAS} A$)

by using the analogues of the standard polynomial reductions for the decision versions.

Exercise. Recall (or suggest) the straightforward polynomial reduction IndSet$_{dec}$ $\leq_p$ VertexCover$_{dec}$ (noting that $C \subseteq V$ is a vertex cover in $G = (V, E)$ iff $V - C$ is an independent set). Though we have not defined $\leq_{PTAS}$ precisely, argue that this polynomial reduction cannot serve to show MaxIndSet $\leq_{PTAS}$ MinVertexCover. (Though MinVertexCover $\in$ APX(2), the reduction can not be used to deduce MaxIndSet $\in$ APX (why?).)

We just note that by using the notion of $\leq_{PTAS}$-reductions we can define *NPO-complete problems*, *APX-complete problems*, *PTAS-complete problems*, ... Here NPO is the class of

"nondeterministic polynomial-time optimization problems"; a problem A belongs to NPO if we can decide in polynomial time for a given (x, s) if s is a solution of x and compute $v(x, s)$ in the positive case. It is reasonable to restrict APX, PTAS, … to problems in NPO; then we have FPTAS ⊆ PTAS ⊆ APX ⊆ NPO.
An example NPO-complete problem is MaxWeightSat: its instance is a cnf boolean formula $\phi(x_1, x_2, \ldots, x_n)$ with nonnegative integer weights of variables, $w : \{x_1, x_2, \ldots, x_n\} \to \mathbb{N}$; the value $v(\phi, a)$ for a non-satisfying assignment a is 1, and $v(\phi, a) = max\{\, 1\, ,\, \sum_{i,a(x_i)=1} w(x_i)\, \}$ for a satisfying assignment a.
Also MinTSP is NPO-complete.

We have already mentioned that the PCP-theorem enables to show that ε-robust-3-Sat is NP-complete, for some $\varepsilon > 0$. In more detail, there is a polynomial algorithm which allows for the use of the *gap technique*: it transform any 3-SAT formula ϕ into a 3-SAT formula ϕ' so that the following holds: if ϕ is satisfiable then also ϕ' is satisfiable, and if ϕ is not satisfiable then each assignment satisfies less than $(1-\varepsilon)m$ clauses in ϕ' where m is the number of clauses in ϕ'.

This result has enabled to show other lower bounds for approximability. In particular we can deduce that

- Max-3-Sat does not belong to PTAS, when P≠NP.

Moreover, it can be even shown that Max-3-Sat is APX-complete.

It can be also shown that

- MaxClique does not belong to APX, when P≠NP.

Let us at least sketch the use of the gap technique for showing the last claim MaxClique $\notin$ APX. (Recall that we used the gap technique when showing TSP $\notin$ APX; to find a direct construction for MaxClique, which would do the job, seems much more difficult, but with the PCP theorem it is easy, as we shall see.)

Let us consider the problem 3-SAT and the respective polynomial program (protocol) P guaranteed by the PCP theorem. Thus our program P takes a 3-cnf boolean formula ϕ as an input, let $n = size(\phi)$, generates (and uses) $c \log n$ random bits and asks for k proof-bits (where c, k are fixed constants). We know that if ϕ is satisfiable then there is a proof(string) such that P accepts ϕ with certainty; if ϕ is not satisfiable then any proof can make P to accept with probability 1/2 at most.

We suggest the following construction. Given ϕ, we construct a graph G where each node corresponds to a particular bit string r of length $c \log n$ and a particular bit string q of length k for which P accepts. (There are at most $2^{c \log n} \cdot 2^k$, i.e. $O(n^c)$, such nodes, for each of which we can check in polynomial time whether P accepts.) Now we put an edge between two nodes v, v' of G iff they correspond to different vectors r, r' with compatible proof-strings (if the address of the asked bit $q(i)$ is the same in both cases v, v' then this bit must have the same value in both cases).

It is now easy to check that if ϕ is satisfiable then there is a clique in G of size n^c, and if ϕ is not satisfiable then any clique has the size at most $\frac{n^c}{2}$.

Finally, we recall that we can achieve that accepting a nonsatisfiable ϕ can happen with the probability at most $1/d$ for any fixed constant d (by repeating P g-times where $2^g \geq d$), and thus the result MaxClique $\notin$ APX easily follows (when assuming P$\neq$NP, of course).

Část 5

7 Automata and logic on infinite words

As a motivating example, we describe briefly one method in the area of *model checking*, which is a part of the broader area of (automated) verification.

Peterson's mutual exclusion protocol

For concreteness, recall Peterson's protocol solving the mutual exclusion problem. We imagine two processes A, B running in parallel. (An abstraction of) process A performs the infinite loop

```
while true do
$(noncrit_A;$
$flag_A := true;\ turn := B;$ waitfor $(flag_B = false) \vee (turn = A);\ crit_A;\ flag_A := false)$
```

and process B performs

```
while true do
$(noncrit_B;$
$flag_B := true;\ turn := A;$ waitfor $(flag_A = false) \vee (turn = B);\ crit_B;\ flag_B := false)\,.$
```

A *global state* of our system $\mathcal{S} = A \| B$ (processes A, B running in parallel; they communicate by shared variables) can be described by the values of a few (boolean) variables, i.e., by a (column) vector from $\{0,1\}^k$ for some fixed k. Our system $\mathcal{S}$ thus determines (can be equated with) a (finite) transition system $(G, \rightarrow)$ where G is the set of all global states and $\rightarrow \subseteq G \times G$ is the generated (unlabelled) transition relation, describing how a global state can change in one step; a concrete $g_0 \in G$ is the initial state. (E.g., the vector from $\{0,1\}^k$ which corresponds to the global state with $noncrit_A = true$, $noncrit_B = true$, $flag_A = false$, $flag_B = false$, $turn = A$, $\ldots$ can be the initial state.)

Exercise. You can draw at least a part of the graph of $\mathcal{S} = (G, \rightarrow)$.

Each run of our system can be seen as a (potentially infinite) sequence $g_0, g_1, g_2, \ldots$ such that $g_i \rightarrow g_{i+1}$ for all i. The system thus determines the set of its infinite runs, which is a *language of infinite words*, i.e. a subset of Σ^ω

> (so $w \in \Sigma^\omega$ is a function $w : \{0, 1, 2, \ldots\} \rightarrow \Sigma$, i.e., an infinite sequence $a_0 a_1 a_2 \ldots$ where $a_i = w(i)$)

where $\Sigma = \{0,1\}^k$ in our case. (Thus a *letter* is a [column] vector of k 0's and/or 1's.)

Remark. If there is a global state g with no successor (there is no g' such that $g \rightarrow g'$) then we can assume that there is a loop $g \rightarrow g$, by which we make all maximal runs infinite. This is usually the case at so called Kripke structures, which are, in fact, nothing else than the transition systems as described above. (We have a set of atomic propositions, like "$turn = B$", "$crit_A$", etc., which are either true or false in each given [global] state.)

Model checking (safety and liveness properties)

In our example system $\mathcal{S} = A \| B$ we can be naturally interested in checking if the following property, an example of a so-called *safety property*, is satisfied: (starting from the initial state,) the processes can never be in the critical section simultaneously, i.e., the global state g satisfying $(crit_A \wedge crit_B)$ is not reachable; in other words, the states on each run satisfy $\neg crit_A \vee \neg crit_B$. Another desirable property, an example of a so-called *liveness property*, is that whenever process A wants to enter the critical section, i.e., sets $flag_A := true$, it will eventually enter that section, i.e., a global state in future will satisfy $crit_A$.

One general method for checking if a given property ϕ is satisfied in (the initial state of) system $\mathcal{S}$ is to construct an automaton accepting all runs that violate ϕ, let us denote this automaton $\mathcal{B}_{\neg\phi}$; this automaton will be combined with the (transition) system $\mathcal{S}$, yielding an automaton $\mathcal{B}_{(\mathcal{S},\phi)}$, and we will ask if $\mathcal{B}_{(\mathcal{S},\phi)}$ accepts an (infinite) word. If yes then $\mathcal{S}$ allows a run which violates ϕ, if not, i.e., if $\mathcal{L}_\omega(\mathcal{B}_{(\mathcal{S},\phi)}) = \emptyset$ [the language of all infinite words accepted by $\mathcal{B}_{(\mathcal{S},\phi)}$ is empty], then $\mathcal{S}$ satisfies ϕ (i.e., all runs of $\mathcal{S}$ satisfy ϕ).

Büchi automata

What kind of automata do we have in mind when speaking about $\mathcal{B}_{\neg\phi}$, $\mathcal{B}_{(\mathcal{S},\phi)}$? We mean *Büchi automata*; a Büchi automaton is, in fact, the usual nondeterministic finite automaton, only the acceptance condition is different (dealing with infinite words, not the finite ones). Given a (nondeterministic finite) automaton $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ (where $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation and $F \subseteq Q$ is the set of accepting states) we denote

$$\mathcal{L}_\omega(\mathcal{B}) = \{w \in \Sigma^\omega \mid \text{there is a run of } B \text{ on } w \text{ which goes through } F \text{ infinitely often}\}.$$

As expected, a run of B on $w = a_0a_1a_2\ldots$ is a sequence $\sigma = q_0, q_1, q_2, \ldots$ such that $q_i \xrightarrow{a_i} q_{i+1}$ (i.e., $(q_i, a_i, q_{i+1}) \in \delta$) for all $i = 0, 1, 2, \ldots$. By $Infin(\sigma)$ we denote the set of those states $q \in Q$ which appear infinitely often (i.e., infinitely many times) in σ. Hence $\mathcal{L}_\omega(\mathcal{B}) = \{w \in \Sigma^\omega \mid$ there is a run σ of $\mathcal{B}$ on w such that $Infin(\sigma) \cap F \neq \emptyset\}$.

It is easy to construct $\mathcal{B}_{\neg\phi_1}$ where ϕ_1 is the above safety property. We can take $(\{q_0, q_1\}, \Sigma, \delta, q_0, \{q_1\})$ where $\Sigma = \{0,1\}^k$ as discussed above, and δ contains the triples (the transitions) $q_0 \xrightarrow{x} q_0$ for all $x \in \Sigma$ which represent the global states satisfying $\neg crit_A \vee \neg crit_B$, $q_0 \xrightarrow{y} q_1$ for all $y \in \Sigma$ which represent the global states satisfying $crit_A \wedge crit_B$, and $q_1 \xrightarrow{z} q_1$ for all $z \in \Sigma$.

Remark. Our automaton is, in fact, deterministic. A certain disadvantage of Büchi automata, discussed also later, is that the nondeterministic version is more powerful than the deterministic one (unlike the case for finite automata accepting languages of finite words). This is demonstrated in the next exercise.

Exercise. Construct a (nondeterministic) Büchi automaton $\mathcal{B}$ such that $\mathcal{L}_\omega(\mathcal{B}) = \{w \in \{0,1\}^\omega \mid w$ contains only finitely many 1's$\}$. There is no deterministic Büchi automaton accepting this language; can you prove this?

Exercise. Construct $\mathcal{B}_{\neg\phi_2}$ where ϕ_2 is the above liveness property.

We now look how to create the above mentioned automaton $\mathcal{B}_{(\mathcal{S},\phi)}$, which is a combination of the transition system $\mathcal{S}$ (like the one generated by our "Peterson's system") and the Büchi automaton $\mathcal{B} = \mathcal{B}_{\neg\phi}$. In principle, we will do the usual product construction (which synchronizes the runs of $\mathcal{S}$ and $\mathcal{B}$); we just have to overcome the technical problem that the states of $\mathcal{S}$ are, in fact, the letters for $\mathcal{B}$. But this problem is easy: we can introduce a special "starting state" q_{start} and add a special "exit state" e_g to each $g \in \mathcal{S}$; now the original states $g \in \mathcal{S}$ are handled as letters (edge-labels), and we put $q_{start} \xrightarrow{g_0} e_{g_0}$, and $e_g \xrightarrow{g'} e_{g'}$ for all $g \to g'$. Then we can define $\mathcal{B}_{(\mathcal{S},\phi)} = \mathcal{A}_{\mathcal{S}} \times \mathcal{B}_{\neg\phi}$, where $\mathcal{A}_{\mathcal{S}}$ is the (nondeterministic finite) automaton arising from $\mathcal{S}$ as discussed above.

Exercise. Give a precise definition of $\mathcal{A} \times \mathcal{B}$.

There are more variants how to define the accepting states of the product automaton. In our discussed case, $\mathcal{A}_{\mathcal{S}}$ has no accepting states, and the accepting states of $\mathcal{A}_{\mathcal{S}} \times \mathcal{B}_{\neg\phi}$ will be the pairs (q_1, q_2) with $q_2 \in F$, where F is the set of accepting states of $\mathcal{B}_{\neg\phi}$. (Why?)

Recall that our general (model checking) method finally checks whether $\mathcal{L}_\omega(\mathcal{B}_{(\mathcal{S},\phi)}) = \emptyset$.

Exercise. Suggest a method how to decide whether $\mathcal{L}_\omega(\mathcal{B}) = \emptyset$ for a given (description of a) Büchi automaton $\mathcal{B}$.

Note that if $\mathcal{L}_\omega(\mathcal{B}_{(\mathcal{S},\phi)}) \neq \emptyset$ then you can provide a *counterexample*, i.e., (a description of) a run of the (original) system $\mathcal{S}$ (like the "Peterson's system") which violates the property ϕ.

Monadic second order logic S1S

Now we are interested in characterizing what kind of properties ϕ allow to construct the appropriate Büchi automata $\mathcal{B}_\phi$ (or $\mathcal{B}_{\neg\phi}$). It turns out that they are precisely those expressible in the monadic second-order logic of one successor, briefly denoted as S1S.

Recall what is the first order logic: the language contains (symbols for) variables $x, y, z, \ldots$, logical connectives like $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$, quantifiers $\exists, \forall$, usually also the special predicate $=$ (obligatorily interpreted as the equality), and "nonlogical" symbols, i.e., some function symbols $f, g, \ldots$ (with their arities) and/or some predicate symbols $P, Q, \ldots$ (with their arities).

Exercise. Recall the syntactic rules for creating *terms*, *atomic formulas*, *formulas* (including the "syntactic sugar" like parantheses), *free* and *bound occurences* of variables, etc., and the *interpretation* of terms and formulas in concrete (mathematical) structures.

Remark. Also recall that we can "narrow" the logical symbols to $\neg$, $\vee$, $\exists$, and handle the other connectives and $\forall$ as derived. For expressing statements it is useful to have the "broad" logic, for proving things about the logic it is useful to keep it "narrow".

Logic S1S has all ingredients of the first order logics, with only one nonlogical symbol: the unary function symbol s ("successor"). Moreover, it is a second-order logic, so it also has variables $X, Y, Z, \ldots$ for predicates but it is monadic, i.e., these variables only range over unary predicates, i.e., sets. Finally, we have a special predicate $\in$; as expected, its type only allows to use it in atomic formulas of the form $t \in X$ where t is a (1st order) term (so $t = ss \ldots sx$ for some (maybe zero) number of occurences of s).

We have a concrete structure in mind, in which we interpret logic S1S, namely the structure $(\mathbb{N}, s, \in)$, with the usual interpretation ($s(n) = n + 1$, $n \in C$ iff number n is an element of set C).

Remark. Note that we follow the usual practice and use the same typographical symbols $s, \in$ for both the symbols in the logic and for denoting one concrete function and one concrete predicate in the structure $\mathbb{N}$. It is also useful to note that here $\mathbb{N} = \{0, 1, 2, \dots\}$ serves primarily for modelling (discrete) time (rather than for arithmetics).

We will naturally need to express that a time point x is initial, i.e., $x = 0$. The constant 0 is not in the logic S1S (for keeping the logic as "narrow" as possible) but $x = 0$ can be expressed by the formula $\forall y \neg(x = sy)$. In fact, even "=" is not included in S1S since $t_1 = t_2$ can be expressed by $\forall X(t_1 \in X \leftrightarrow t_2 \in X)$, and $X = Y$ can be expressed by $\forall x(x \in X \leftrightarrow x \in Y)$. It is certainly also useful to express that a time point x is earlier than y, i.e. $x < y$.

Exercise. Find an S1S-formula expressing $x < y$ (in our intended structure $(\mathbb{N}, s, \in)$).
Also find S1S-formulas expressing "X is finite" and "X is infinite".

Remark. As mentioned above, it is usual (and convenient) to use the symbols 0, 1 ($= s0$), 2, $\dots$, $=$, $<$, $\leq$, $\subseteq$ (for $X \subseteq Y$) as if they belonged to S1S, while we are aware that they are, in fact, just "abbreviations".

Expressing properties of runs in S1S

Let us look how we can express properties of runs, like our safety property ϕ_1 and our liveness property ϕ_2, in S1S. Imagine a run ρ of (e.g., "Peterson's") system $\mathcal{S}$, and recall that ρ can be viewed as a sequence of column vectors $a_0 a_1 a_2 \dots$ ($a_i \in \{0,1\}^k$); we can say that in time $i \in \mathbb{N}$ the run goes through the (global) state a_i. Run ρ thus determines a k-track tape with cells $0, 1, 2, \dots$. Each track j is (filled with) an infinite sequence of 0's and 1's (an element of $\{0,1\}^\omega$), and thus can be viewed as the *characteristic sequence* of a set $T_j^\rho \subseteq \mathbb{N}$: $i \in T_j^\rho$ iff the value in Track j in the cell (time point) i is 1. In our example, one of the tracks, say Track 1, determines the set T_1^ρ of time points where $crit_A$ holds; another track, say Track 2, determines the set T_2^ρ, the set of time points where $crit_B$ holds.

So we have seen how a sequence $\rho \in (\{0,1\}^k)^\omega$ determines the k-tuple of sets $T_1^\rho, T_2^\rho, \dots, T_k^\rho$. On the other hand, each k-tuple $T_1, T_2, \dots, T_k$ of subsets of $\mathbb{N}$ determines one $\rho \in (\{0,1\}^k)^\omega$ such that $T_j^\rho = T_j$ (for $j = 1, 2, \dots, k$).

It is thus clear that the S1S formula $\Phi_1(X_1, X_2, \dots, X_k)$ defined as $\forall i : i \in X_1 \to i \notin X_2$ can be naturally viewed as an expression of the safety property ϕ_1, since $\Phi_1(T_1, T_2, \dots, T_k)$ is true precisely for those k-tuples of sets $T_1, T_2, \dots, T_k \subseteq \mathbb{N}$ which correspond to sequences of (potential) global states of $\mathcal{S}$ which do not contain any "forbidden" state, i.e., the state with $crit_A \wedge crit_B$.

Remark. $\Phi_1(X_1, X_2, \dots, X_k)$ thus determines all "safe sequences" from $(\{0,1\}^k)^\omega$, where not all of them necessarily correspond to real runs of system $\mathcal{S}$. But we have that system $\mathcal{S}$ has (i.e., all runs of system $\mathcal{S}$ have) the property ϕ_1 iff $\Phi_1(T_1^\rho, T_2^\rho, \dots, T_k^\rho)$ holds for all runs ρ of $\mathcal{S}$; in other words, iff there is no run ρ of $\mathcal{S}$ for which $\neg\Phi_1(T_1^\rho, T_2^\rho, \dots, T_k^\rho)$.

Exercise. Suggest a formula $\Phi_2(X_1, X_2, \dots, X_k)$ so that $\mathcal{S}$ has the above defined liveness property ϕ_2 iff for all runs ρ of $\mathcal{S}$ we have $\Phi_2(T_1^\rho, T_2^\rho, \dots, T_k^\rho)$.

Equivalence of S1S and Büchi automata

Now we want to show the equivalence (wrt the expressive power) between S1S and Büchi automata. The following exercise deals with the easier direction.

Exercise. Assume a Büchi automaton $\mathcal{B} = (Q, \{0,1\}, \delta, q_0, F)$ and suggest an S1S formula

$\Phi_{\mathcal{B}}(X)$ such that $\Phi_{\mathcal{B}}(T)$ for $T \subseteq \mathbb{N}$ iff $charseq(T) \in \mathcal{L}_\omega(\mathcal{B})$. (Here $charseq(T) \in \{0,1\}^\omega$ is the characteristic sequence of T.)
Sketch how you would generalize the result for alphabet $\{0,1\}^k$.

Now assume a S1S formula $\Phi(x_1, x_2, \ldots, x_m, X_1, X_2, \ldots, X_k)$. We want to construct a Büchi automaton $\mathcal{B}_\Phi$, over the alphabet $\{0,1\}^{m+k}$ such that $\Phi(t_1, t_2, \ldots, t_m, T_1, T_2, \ldots, T_k)$ (for $t_i \in \mathbb{N}$, $T_i \subseteq \mathbb{N}$) is true iff $\mathcal{B}_\Phi$ accepts the infinite word $\rho \in (\{0,1\}^{m+k})^\omega$ where $T_1^\rho = \{t_1\}, \ldots, T_m^\rho = \{t_m\}, T_{m+1}^\rho = T_1, \ldots, T_{m+k}^\rho = T_k$.

We proceed by induction on the structure of Φ.

Exercise. Show how to construct an equivalent automaton for atomic formula $ss\ldots sx \in X$.

Exercise. Show how to construct an equivalent automaton for the formula $\Phi_1(\overline{x}, \overline{X}) \vee \Phi_2(\overline{x}, \overline{X})$ when we already have the respective automata $\mathcal{B}_1, \mathcal{B}_2$ for $\Phi_1(\overline{x}, \overline{X})$, $\Phi_2(\overline{x}, \overline{X})$. ($\overline{x}$ is a shorthand for $x_1, \ldots, x_m$ and $\overline{X}$ is a shorthand for $X_1, \ldots, X_k$.)

Exercise. Show how to construct an equivalent automaton for the formula $\exists x_1 \Phi(x_1, \ldots, x_m, \overline{X})$ [with free variables $x_2, \ldots, x_m, \overline{X}$] when we already have an automaton $\mathcal{B}$ equivalent with $\Phi(x_1, \ldots, x_m, \overline{X})$. (Hint. Use nondeterministic guessing of a [nonpresent] track content.) Generalize for the case $\exists X_1 \Phi(\overline{x}, X_1, \ldots, X_k)$.

Complementation; Muller automata

We can see that it only remains to handle the case $\neg\Phi(\overline{x}, \overline{X})$, when having $\mathcal{B}$ for $\Phi(\overline{x}, \overline{X})$. (Other logical connectives and the universal quantifier are expressible by those handled.)

We note that even in the case when $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ is deterministic, the construction (of $\mathcal{B}'$ accepting the complement of $\mathcal{L}_\omega(\mathcal{B})$) is not immediate. (We could not just replace F with $Q - F$; why?) This problem could be solved by using (deterministic) *Muller automata.* Such an automaton $\mathcal{M} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ differs in that $\mathcal{F} \subseteq 2^Q$, i.e., there is a set of accepting sets of states (instead of a set of accepting states); the language is then defined

$$\mathcal{L}_\omega(\mathcal{M}) = \{w \in \Sigma^\omega \mid \text{there is a run } \sigma \text{ of } \mathcal{M} \text{ on } w \text{ such that } \mathit{Infin}(\sigma) \in \mathcal{F}\}.$$

Exercise. Show how, given a Büchi automaton $\mathcal{B}$, we can construct a Muller automaton $\mathcal{M}$ so that $\mathcal{L}_\omega(\mathcal{M}) = \mathcal{L}_\omega(\mathcal{B})$; moreover, if $\mathcal{B}$ is deterministic then $\mathcal{M}$ is deterministic.

Exercise. Show how to construct an equivalent Muller automaton for the formula $\neg\Phi(\overline{x}, \overline{X})$ when we have a deterministic Muller automaton $\mathcal{M}$ equivalent with $\Phi(\overline{x}, \overline{X})$.

Unfortunately, Büchi automata cannot be generally determinized, as we already noted. Muller automata can be determinized, as we shall show. The next exercise thus shows that nondeterministic Muller automata, deterministic Muller automata, and nondeterministic Büchi automata are equally expressive.

Exercise. Show how, given a (nondeterministic) Muller automaton $\mathcal{M}$, we can construct a (nondeterministic) Büchi automaton $\mathcal{B}$ so that $\mathcal{L}_\omega(\mathcal{B}) = \mathcal{L}_\omega(\mathcal{M})$.

Rabin automata

Hence we will be done when we show how a nondeterministic Büchi automaton can be transformed to an equivalent deterministic Muller automaton. As a convenient intermediate step, we introduce *Rabin automata*:

A Rabin automaton $\mathcal{R}$ is defined by (Q, Σ, δ, q_0) and some pairs $(G_1, R_1), \ldots, (G_m, R_m)$; where $G_i, R_i \subseteq Q$ (think of G as "green light" and R as "red light"); the accepted language is now defined as follows:

$$\begin{aligned}\mathcal{L}_\omega(\mathcal{R}) = \{w \in \Sigma^\omega \mid {} & \text{there is a run } \sigma \text{ of } \mathcal{R} \text{ on } w \text{ such that for some } i \text{ we have} \\ & Infin(\sigma) \cap G_i \neq \emptyset \text{ and } Infin(\sigma) \cap R_i = \emptyset\}.\end{aligned}$$

(A green light is on infinitely often while the respective red light is on only finitely often.)

Exercise. Show that a Rabin automaton $\mathcal{R}$ can be transformed to an equivalent Muller automaton $\mathcal{M}$; moreover, if $\mathcal{R}$ is deterministic then $\mathcal{M}$ is deterministic.

Safra's construction (nondet-Büchi $\rightarrow$ det-Rabin)

The crucial point is a construction due to Safra; we now explain its basic ideas.

Assume a given nondeterministic Büchi automaton $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ over alphabet Σ, with n states; we will construct an equivalent deterministic Rabin automaton $\mathcal{R}$ (thus $\mathcal{L}_\omega(\mathcal{B}) = \mathcal{L}_\omega(\mathcal{R})$), with $2^{O(n \log n)}$ states.

Consider a word $w = a_1 a_2 a_3 \cdots \in \Sigma^\omega$, and imagine the respective "determinized token-run" on the graph of $\mathcal{B}$: we start with a token on the initial state q_0, and move (and duplicate and remove) the tokens so that after processing $a_1 a_2 \ldots a_t$ the tokens are precisely on the states which are reachable from q_0 by $a_1 a_2 \ldots a_t$.

Exercise. Recall the construction of a deterministic finite automaton equivalent to a given nondeterministic finite automaton; this is, in fact, the above idea.

It is clear that if there is an infinite run of $\mathcal{B}$ on w then the tokens can never completely disappear from the "board" in our "token-game" when processing w. (Why?) The other direction needs a moment of thought:

Exercise. Show that if the tokens never completely disappear from the board in our token-game when processing w then there is an infinite run of $\mathcal{B}$ on w.
(Hint. Note that each state q which has a token after processing the prefix $a_1 a_2 \ldots a_t$ $(t \geq 1)$ can be reached by a_t from a state q' (a "predecessor") which had a token after processing $a_1 a_2 \ldots a_{t-1}$; we can imagine an appropriate edge from "vertex" $(q', t-1)$ to vertex (q, t). This idea leads to an infinite tree which is finitely branching (i.e., each vertex has finitely many outgoing edges). We can thus apply the well-known *König's Lemma* which tells us that every infinite tree which is finitely branching has an infinite branch. [Can you prove this lemma?])

But the above determinization is not sufficient for us.

Exercise. Show that when we just declare the sets (the token distributions) $Q' \subseteq Q$ such that $Q' \cap F \neq \emptyset$ as the accepting states of the constructed deterministic Büchi automaton $\mathcal{B}'$, it can not be guaranteed that $\mathcal{L}_\omega(\mathcal{B}') = \mathcal{L}_\omega(\mathcal{B})$.

If we want, e.g., that the configuration C_t, i.e. the distribution of tokens after processing $a_1 a_2 \ldots a_t$, provides for each state q not only information if q is reachable (from q_0 by $a_1 a_2 \ldots a_t$) but also if it is reachable via F, we can enhance our token-game as follows:

We imagine that the basic tokens mentioned sofar are white, and we also have a source of blue tokens. If a white token is placed on some (accepting state) $q \in F$, we put a blue token on top of it; thus a "stack" $(white, blue)$ with height 2 arises (its bottom being on the left in our presentation), and we handle this stack as a unit in the following moving (and duplicating and removing); moreover, when there is a conflict, i.e., when in performing step $C_t \overset{a_{t+1}}{\rightarrow} C_{t+1}$ some state q can get both the stack $(white)$ and the stack $(white, blue)$, the conflict is resolved in favour of $(white, blue)$.

Exercise. Verify that this modified token game serves to the announced aim (C_t provides information about the reachable states and also about those reachable via F). But show that even if the $(white, blue)$-stacks are present in all $C_j, C_{j+1}, C_{j+2}, \ldots$ (for some j) when processing w, this does not necessarily mean that there is an accepting run of $\mathcal{B}$ on w.

We have to go more deeply. Note that to each q which has a token in C_t we can attach (maybe several) vectors of the type $(i_1, i_2, \ldots, i_r)$ with $0 \leq i_1 < i_2 < \cdots < i_r \leq t$ presenting information that q is reachable from q_0 by $a_1 a_2 \ldots a_t$ by a path visiting F at "time points" $i_1, i_2, \ldots, i_r$. Given two such vectors $(i_1, i_2, \ldots, i_r)$, $(j_1, j_2, \ldots, j_s)$ we can compare them wrt the following (modified lexicographic) order: $(i_1, i_2, \ldots, i_r) \prec (j_1, j_2, \ldots, j_s)$ (where $x \prec y$ can be read as "x is better than y") iff $i_1 = j_1$, $i_2 = j_2$, $\ldots$, $i_m = j_m$, and ($i_{m+1} < j_{m+1}$ or ($m = s$ and $r > s$)), for some $m \leq \min\{r, s\}$. The quality of q in C_t can be defined as the best appropriate vector $(i_1, i_2, \ldots, i_r)$ (the least wrt $\prec$); if q is reachable by $a_1 a_2 \ldots a_t$ but not via F then its quality is the empty vector $()$ (which is worse than any nonempty vector); if q is not reachable by $a_1 a_2 \ldots a_t$ then we can define its quality (in C_t) as $\perp$, which is viewed as worse than any vector.

You can now contemplate how to implement a program which should process (reading from left to right) an input word $a_1 a_2 a_3 \ldots$ while being able to compare all the states of $\mathcal{B}$ wrt their qualities – after each prefix $a_1 a_2 \ldots a_t$. For this task it is not necessary that your program always stores the current values of the qualities; finite memory should be sufficient. Hopefully you would finally come up with something like the construction described below; it uses more colours and bigger stacks of coloured tokens. We also add some "(green and red) light effects", to make explicitly clear that the resulting program is, in fact, a deterministic Rabin automaton. (So it would be now really useful if you think a while before reading further.)

Our program starts just with a white token, i.e. a $(white)$-stack, on q_0; this is configuration C_0.

Generally, configuration C_t will consist of stacks of coloured tokens on states of $\mathcal{B}$, and also contain information about which colour has its green and/or red light on, and information about the current "age-order" of colours (some tokens of) which are present on the board; $c < c'$ is read as "c is older than c'" (i.e., last introducing of c on the board happened earlier than last introducing of c').

Having C_t, the program (automaton) reads a_{t+1} and constructs C_{t+1} as follows:

- It moves (and duplicates and/or removes) all token-stacks along the arrows labelled with a_{t+1}; the conflicts are resolved in favour of stacks which are the least wrt the modified lexicographic order $\prec$; the stack $(c_1, c_2, \ldots, c_r)$ on q in C_t satisfies $c_1 < c_2 < \cdots < c_r$ and can be viewed as a substitute of the quality of q in C_t.
  Each colour which thus disappears (i.e., all tokens of this colour disappear) from the board, has its red light on in C_{t+1}.

- Each stack which now lies on (a state from) F gets a new token on top; for all such stacks of the same type we use the tokens of the same colour which is currently not present on the board, and is thus (newly) introduced on the board. The order of introducing these (youngest) colours, for all types of stacks lying on F, is arbitrary.

- Any colour c (there can be more of them) which is present on the board but invisible from top (since all the tokens of that colour are covered by other tokens) will have its green light on in C_{t+1}; moreover, all tokens above the c-tokens are removed – the colours which disappear in this way will also have their red light on in C_{t+1}.

Remark. Note that some colour can have both its green light and its red light on in C_{t+1}.

Exercise. Check that it is sufficient to use n colours, and thus stacks of height $\leq n$ (where n is the number of states of $\mathcal{B}$).
Also verify the following properties: 1/ any two tokens of the same colour (in C_t) have the same (sub)stacks below; 2/ the tokens removed because of invisibility of some colour(s) all belong to the colours which completely disappear in this way.
Derive an upper bound on the number of states of the Rabin automaton, i.e. on the size of the program finite memory, showing that it is in $2^{O(n \log n)}$ (as announced).
Try to define the pairs (G_i, R_i) to finish the definition of the Rabin automaton $\mathcal{R}$. (Compare with the following correctness proof.)

We still need to show the correctness, i.e. that $\mathcal{L}_\omega(\mathcal{R}) = \mathcal{L}_\omega(\mathcal{B})$.

Suppose first that $w = a_1a_2a_3\dots$ is accepted by $\mathcal{B}$, i.e., there is a run $\rho = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots$ of $\mathcal{B}$ on w which goes through F infinitely often; let $\sigma(t)$ denotes the stack which lies on q_t when $\mathcal{R}$ has processed $a_1a_2\dots a_t$. Let now $m \in \mathbb{N}$ be such that for some t_0 we have $\forall t \geq t_0 : \mathit{height}(\sigma(t)) \geq m$ and $\mathit{height}(\sigma(t)) = m$ infinitely often. (Why must such m exist?) The bottom part $(c_1, c_2, \dots, c_m)$ of $\sigma(t)$ can change for $t = t_0, t_0+1, t_0+2, \dots$ because of possible replacing with a stack which is lesser wrt $\prec$; but this cannot happen infinitely often. (Why?) So for some $t_0' \geq t_0$ we have that the bottom parts of height m of all $\sigma(t)$, $t \geq t_0'$, are the same $(c_1, c_2, \dots, c_m)$. This must mean that the green light of c_m is on infinitely often but its red light is on only finitely often. (Why?)

Now suppose that when $\mathcal{R}$ processes $w = a_1a_2a_3\dots$, some colour c has its green light on infinitely often but its red light is on only finitely often. This also means that from some t_0 on colour c is present on the board and never removed. Let $t_1 < t_2 < t_3 < \cdots$ be the time points (bigger than t_0) when c has its green light on; this also means that in each C_{t_j} there are stacks with a c-token on top.

Exercise. Show that each state q which has a stack with a c-token on top in $C_{t_{j+1}}$ can be reached by $a_{t_j+1}a_{t_j+2}\dots a_{t_{j+1}}$ via F from some q' which has a stack with a c-token on top in C_{t_j}. Then use König's Lemma to deduce that there is an accepting run of $\mathcal{B}$ on w.

Reference

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein. *Introduction to Algorithms, Second Edition*. MIT Press, 2001.

[2] Ivan Damgård. Commitment schemes and zero-knowledge protocols. In *Lectures on Data Security*, volume 1561 of *Lecture Notes in Computer Science*, pages 63–86. Springer, 1999.

[3] Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

[4] Oded Goldreich. Zero-knowledge twenty years after its invention, (www.wisdom.weizmann.ac.il/˜oded/ps/zk-tut02v4.ps, accessed oct 14, 2008). Technical report, 2004.

[5] Juraj Hromkovič. *Theoretical Computer Science*. Springer, 2004.

[6] Dexter C. Kozen. *Theory of Computation*. Springer, 2006.

[7] Uwe Schöning and Randall Pruim. *Gems of Theoretical Computer Science*. Springer, 1998.

[8] Michael Sipser. *Introduction to the theory of computation, Second Edition*. Thomson, 2006.

[9] Ingo Wegener. *Complexity Theory*. Springer, 2005.