

Introduction to theoretical cryptography

Ingrid Biehl
Johannes Buchmann
Fachbereich Informatik
Universität des Saarlandes

March 15, 2005

Contents

1	Probabilistic Turing Machines	2
1.1	Definition and basic properties	2
1.2	Running time of a probabilistic Turing machine	6
1.3	Combining probabilistic Turing machines	7
2	Interactive systems	9
2.1	Interactive Turing machines	9
2.2	Interactive systems	10
2.3	Interactive proof systems	12
2.4	Arthur-Merlin-Games	32
3	Distinguishing probability distributions	33
3.1	Notions of distinguishability	33
3.2	Indistinguishability	34
4	Zero knowledge proofs	42
4.1	The idea	42
4.2	Zero knowledge proof systems	43
4.3	Secure encryption	45
4.4	$NP \subseteq ZK$	46

Chapter 1

Probabilistic Turing Machines

1.1 Definition and basic properties

In many areas of computer science the known probabilistic algorithms are more powerful than the known deterministic algorithms. The partners in a communication are able to use the most powerful tools available. If we want to model such a communication we must, therefore, start by introducing a model for probabilistic computation, the probabilistic Turing machines. We will assume that the reader is familiar with the notation and results of the theory of deterministic and nondeterministic Turing machines. A good reference for those results is the book of [2]. Throughout this book we let Σ be an alphabet containing 0, 1, # and not containing the symbols L and R and $\Sigma_0 = \Sigma - \{\#\}$.

Definition 1.1.1 A *k*-tape nondeterministic Turing machine is a quadruple (K, Σ, Δ, s) , where

- K* is a finite set of states, not containing the halt state *h*;
- s* \in *K* is the initial state;
- Δ is a subset of $(K \times \Sigma^k) \times ((K \cup \{h\}) \times (\Sigma \cup \{L, R\})^k)$.

A *k*-tape deterministic Turing machine is a nondeterministic Turing machine where Δ is a function from $(K \times \Sigma^k)$ to $((K \cup \{h\}) \times (\Sigma \cup \{L, R\})^k)$.

Definition 1.1.2 A probabilistic Turing machine (PTM) is a pair $Z = (M, p)$ where $M = (K, \Sigma, \Delta, s)$ is a *k*-tape nondeterministic Turing machine and $p: \Delta \rightarrow [0, 1]$ is a function which for every $q \in K$ and $\mathbf{a} \in \Sigma^k$ and for $\Delta(q, \mathbf{a}) = \Delta \cap (\{q\} \times \{\mathbf{a}\} \times (K \cup \{h\}) \times (\Sigma \cup \{L, R\})^k)$ satisfies

$$\sum_{d \in \Delta(q, \mathbf{a})} p(d) = 1.$$

The function p is called the *transition probability* of M .

A (single tape) deterministic Turing machine $M' = (K', \Sigma', \delta', s')$ can be considered

as a special probabilistic Turing machine if we set $k = 1$, $K = K'$, $\Sigma = \Sigma'$, $\Delta = \{(q, a, \delta'(q, a)) : q \in K, a \in \Sigma\}$ and $p(q, a, \delta'(q, a)) = 1$ for every $q \in K$ and $a \in \Sigma$.

A first example for a probabilistic Turing machine is the machine CT which simulates unbiased coin tossing. The machine has two states

$$K = \{s, q\}$$

and the transitions

$$\Delta = \{(s, a, q, i), (q, i, h, R), (q, \#, h, R) : i \in \{0, 1\}, a \in \Sigma\}.$$

The transition probability is defined as follows:

$$\begin{aligned} p(s, a, q, i) &= \frac{1}{2} \text{ for } a \in \Sigma, i \in \{0, 1\}, \\ p(q, i, h, R) &= 1 \text{ for } i \in \{0, 1\}, \\ p(q, \#, h, R) &= 1. \end{aligned}$$

More generally, Z is called a *coin tossing machine* (CTM) if $p(\Delta) \subseteq \{0, 1, \frac{1}{2}\}$. Almost all probabilistic Turing machines considered in the literature are, in fact, coin tossing machines.

A *configuration* of Z is a configuration of M , i.e. a member of

$$\Gamma_Z = (K \cup \{h\}) \times B_1^k$$

where

$$B_1 = B_1(Z) = ((\Sigma_0)\Sigma^* \cup \{e\}) \times \Sigma \times (\Sigma^*(\Sigma_0) \cup \{e\}).$$

A configuration $\alpha = (q, u_1, a_1, v_1, \dots, u_k, a_k, v_k)$ describes the current state q of the machine and the contents (u_i, a_i, v_i) ($1 \leq i \leq k$) of its tapes. Here u_i is the string to the left of the scanned square; the single symbol a_i lies in the scanned square; and v_i is the string to the right of the scanned square. If the state q is the halt state h , then we call α a *halted configuration*. The set of all halted configurations of Z is denoted by H_Z . We also use the abbreviation

$$u_i \underline{a_i} v_i = (u_i, a_i, v_i).$$

For the sake of simplicity we assume for the remainder of this section that $k = 1$. For $k > 1$ the analogous definitions and statements are obtained as easy generalizations.

If $\alpha_1 = (q_1, u_1 a_1 v_1)$ and $\alpha_2 = (q_2, u_2 a_2 v_2)$ are configurations of Z , then the probability $\wp_Z(\alpha_1, \alpha_2)$ for α_1 to yield α_2 in one step is

$$\wp_Z(\alpha_1, \alpha_2) = \begin{cases} p(q_1, a_1, q_2, a_2) & \text{if } (q_1, a_1, q_2, a_2) \in \Delta \\ & \text{and } u_1 = u_2, v_1 = v_2; \\ p(q_1, a_1, q_2, R) & \text{if } (q_1, a_1, q_2, R) \in \Delta \\ & \text{and } u_1 a_1 = u_2, v_1 = a_2 v_2 \text{ or} \\ & u_1 a_1 = u_2, a_2 = \#, v_1 = v_2 = e; \\ p(q_1, a_1, q_2, L) & \text{if } (q_1, a_1, q_2, L) \in \Delta \\ & \text{and } u_1 = u_2 a_2, a_1 v_1 = v_2 \text{ or} \\ & a_1 v_1 = v_2, a_2 = \#, u_1 = u_2 = e; \\ 0 & \text{otherwise.} \end{cases}$$

Proposition 1.1.1 *Let α be a configuration of Z which is not a halted configuration. Then*

$$\sum_{\alpha' \in \Gamma_Z} \wp_Z(\alpha, \alpha') = 1.$$

Proof: Let $\alpha = (q, u a v)$. Then it follows from the properties of the probability function p and the definition of \wp_Z that

$$\sum_{\alpha' \in \Gamma_Z} \wp_Z(\alpha, \alpha') = \sum_{d \in \Delta(q, a)} p(d) = 1. \quad \blacksquare$$

For $\alpha, \beta \in \Gamma_Z$ we define

$$\wp_Z^{(0)}(\alpha, \beta) = \begin{cases} 1 & \text{for } \alpha = \beta; \\ 0 & \text{otherwise;} \end{cases}$$

and for $n \geq 1$

$$\wp_Z^{(n)}(\alpha, \beta) = \sum_{\gamma \in \Gamma_Z} \wp_Z^{(n-1)}(\alpha, \gamma) \wp_Z(\gamma, \beta).$$

Then $\wp_Z^{(n)}(\alpha, \beta)$ is the probability for α to yield β after precisely n steps.

Proposition 1.1.2 *For every $n \geq 0$ and $\alpha \in \Gamma_Z$ we have*

$$\sum_{\beta \in \Gamma_Z} \wp_Z^{(n)}(\alpha, \beta) \leq 1.$$

Proof: For $n = 0$ the statement follows immediately from the definition of $\wp_Z^{(0)}$. Suppose that the assertion has been proved for n . Then we have by Proposition 1.1.1

$$\begin{aligned} \sum_{\beta \in \Gamma_Z} \wp_Z^{(n+1)}(\alpha, \beta) &= \sum_{\beta \in \Gamma_Z} \sum_{\gamma \in \Gamma_Z} \wp_Z^{(n)}(\alpha, \gamma) \wp_Z(\gamma, \beta) \\ &= \sum_{\gamma \in \Gamma_Z} \wp_Z^{(n)}(\alpha, \gamma) \sum_{\beta \in \Gamma_Z} \wp_Z(\gamma, \beta) \\ &\leq 1. \end{aligned} \quad (1.1)$$

The equation 1.1 follows from the theory of convergent series. \blacksquare

If we want to determine the probability for a starting configuration $\alpha = (s, x, \#, e)$ to yield a halted configuration β after any number of steps, we need the following result:

Lemma 1.1.1 *Let $\alpha \in \Gamma_Z$. Then the series*

$$\sum_{\beta \in H_Z} \sum_{n=0}^{\infty} \wp_Z^{(n)}(\alpha, \beta)$$

is convergent and bounded by 1.

Proof: For $\beta = (h, uav) \in H_Z$ we set $\bar{\beta} = (\bar{h}, uav)$ where $\bar{h} \notin K \cup \{h\}$. Let $\bar{H}_Z = \{\bar{\beta} : \beta \in H_Z\}$, $\bar{\Gamma}_Z = \Gamma_Z \cup \bar{H}_Z$. For $\alpha \in \bar{\Gamma}_Z, \beta \in \bar{H}_Z$ we define

$$\wp_Z(\alpha, \beta) = \begin{cases} 1 & \text{if } \alpha \in H_Z \text{ and } \beta = \bar{\alpha}; \\ 0 & \text{otherwise.} \end{cases}$$

Using that definition we extend $\wp_Z^{(n)}$ to $\bar{\Gamma}_Z \times \bar{\Gamma}_Z$. It can easily be verified by induction that for every $N \geq 0$ and $\alpha \in \Gamma_Z$

$$\sum_{\beta \in \bar{\Gamma}_Z} \wp_Z^{(N)}(\alpha, \beta) = 1.$$

Moreover, we can prove as follows for $\alpha \in \Gamma_Z, \beta \in H_Z$

$$\wp_Z^{(N+1)}(\alpha, \bar{\beta}) = \sum_{n=0}^N \wp_Z^{(n)}(\alpha, \beta).$$

If $\beta = \alpha$ then we have $\wp_Z^{(0)}(\alpha, \beta) = 1 = \wp_Z^{(1)}(\alpha, \bar{\beta})$. But if $\beta \neq \alpha$ then $\wp_Z^{(0)}(\alpha, \beta) = 0$ and $\wp_Z^{(1)}(\alpha, \bar{\beta}) = \wp_Z^{(0)}(\alpha, \bar{\beta}) = 0$. This proves the formula for $N = 0$. Now suppose that the formula is true for N . By the definition of \wp_Z we have

$$\begin{aligned} \wp_Z^{(N+1)}(\alpha, \bar{\beta}) &= \sum_{\gamma \in \bar{\Gamma}_Z} \wp_Z^{(N)}(\alpha, \gamma) \wp_Z(\gamma, \bar{\beta}) \\ &= \wp_Z^{(N)}(\alpha, \beta) + \wp_Z^{(N)}(\alpha, \bar{\beta}) \\ &= \sum_{n=0}^N \wp_Z^{(n)}(\alpha, \beta). \end{aligned}$$

Consequently, we have for $\alpha \in \Gamma_Z$

$$\sum_{\beta \in H_Z} \sum_{n=0}^{\infty} \wp_Z^{(n)}(\alpha, \beta) = \lim_{N \rightarrow \infty} \sum_{\beta \in H_Z} \wp_Z^{(N)}(\alpha, \bar{\beta}) \leq 1.$$

The last equation follows from the induction hypothesis. ■

We assume in the sequel that all the halted configurations of a probabilistic Turing machine are of the form $(h, y\#)$ with $y \in \Sigma_0^*$. Let $x, y \in \Sigma_0^*$. Let $\alpha = (s, x\#)$, $\beta = (h, y\#)$. Then we define *the probability for Z to yield y upon input of x* to be

$$\Pi_Z(x, y) = \sum_{n=0}^{\infty} \wp_Z^{(n)}(\alpha, \beta)$$

which makes sense by Lemma 1.1.1. Since there might also be non terminating computations we set

$$\Pi_Z(x, \infty) = 1 - \sum_{(h, y\#) \in H_Z} \Pi_Z(x, y).$$

Again, by Lemma 1.1.1 we have

$$\Pi_Z(x, \infty) \in [0, 1].$$

Proposition 1.1.3 *For every $x \in \Sigma_0^*$ the function $\Pi_Z(x, \cdot) : \Sigma_0^* \cup \{\infty\} \rightarrow [0, 1]$ is a probability function, i.e.*

$$\sum_{y \in \Sigma_0^* \cup \{\infty\}} \Pi_Z(x, y) = 1.$$

For example, $Z = CT$. When starting with configuration $(s, x\#)$ the only halted configurations which CT can reach are $(h, xi\#)$, $i \in \{0, 1\}$, and we have

$$\Pi_Z(x, xi) = \frac{1}{2} \text{ for } i \in \{0, 1\}.$$

1.2 Running time of a probabilistic Turing machine

Let $Z = ((K, \Sigma, \Delta, s), p)$ be a probabilistic Turing machine. A *computation* of Z is a sequence $\gamma_0, \gamma_1, \dots, \gamma_k$ of configurations of Z such that $\wp_Z(\gamma_i, \gamma_{i+1}) > 0$ for $0 \leq i \leq k-1$. The *running time* of this computation is k .

We say that *the running time of Z is bounded by a function $T: \mathbb{N} \rightarrow \mathbb{N}$* if every computation starting with an input of length n has a running time which is bounded by $T(n)$. In particular, if T is a polynomial function with integer coefficients, then Z is said to have *polynomial running time*.

Now we want to introduce the expected running time of Z : the probability for Z to halt on input of x after exactly n steps is

$$P_n(x) = \sum_{\beta \in H_Z} \wp_Z^{(n)}((s, x\#), \beta).$$

If

$$\Pi_Z(x, \infty) = 0$$

and the series

$$E_Z(x) = \sum_{n=1}^{\infty} nP_n(x)$$

converges, its limit is called *the expected running time of Z on input x*.

If the expected running time of Z is defined for every input $x \in \Sigma_0^*$ and if for $T: \mathbb{N} \rightarrow \mathbb{N}$ the inequality $E_Z(x) \leq T(|x|)$ is satisfied for every $x \in \Sigma_0^*$, then we say that *the expected running time of Z is bounded by T*.

Moreover, if for $T: \mathbb{N} \rightarrow \mathbb{N}$ and for every $x \in \Sigma_0^*$ the probability for Z to perform more than $T(|x|)$ operations is zero then we say that *the running time of Z is bounded by T*. In particular, if T is a polynomial function with integer coefficients, then Z is said to have *expected polynomial running time*, respectively it has *polynomial running time*.

1.3 Combining probabilistic Turing machines

From a finite set \mathcal{M} of k -tape probabilistic Turing machines with a common alphabet Σ and disjoint state sets we can build more complicated probabilistic Turing machines by using a *machine schema* (\mathcal{M}, η, M_0) where $M_0 \in \mathcal{M}$ is the initial machine and η is a function from a subset of $\mathcal{M} \times \Sigma^k$ to \mathcal{M} . The schema represents a single probabilistic Turing machine \overline{M} made up of all the individual machines $M \in \mathcal{M}$. For our purposes it is sufficient to consider only deterministic transitions between the individual members of \mathcal{M} . Hence, the definition of \overline{M} is completely analogous to the definition of a Turing machine corresponding to schema of deterministic Turing machines (see [2]).

Example 1.3.1 We present a two-tape probabilistic Turing machine ED which on input of the binary representation of $n \in \mathbb{N}_{\geq 1}$ outputs each number in $\{0, \dots, n-1\}$ with probability $\frac{1}{n}$.

The machine works over the alphabet $\Sigma = \{0, 1, \#\}$ and can be constructed as follows:

On input of the k -bit number n (the representation contains no leading zeros), the machine ED performs k coin tosses and writes the resulting random string on its second tape. It then compares the random string with the input string. If the random string represents an integer that is less than n , the machine copies the random string to its first tape and erases its second tape. Otherwise, the machine starts again.

The probability for ED to construct in the first round a random number which is not less than n is

$$1 - \frac{n}{2^k} \leq \frac{1}{2}.$$

Hence, the probability for ED to halt in the l -th round is

$$\epsilon_{n,l} = \left(1 - \frac{n}{2^k}\right)^{l-1} \frac{n}{2^k}.$$

Therefore the probability $\Pi_{ED}(n, \infty)$ for ED to have infinite running time is zero.

The number of steps the machine ED has to perform per round is bounded by $c \cdot k$ for some constant $c > 0$. Hence the expected running time of ED can be estimated as follows

$$\begin{aligned} E_{ED}(n) &\leq \sum_{l=1}^{\infty} c \cdot k \cdot l \cdot \left(1 - \frac{n}{2^k}\right)^{l-1} \cdot \frac{n}{2^k} \\ &\leq c \cdot k \sum_{l=1}^{\infty} l \cdot \left(\frac{1}{2}\right)^{l-1} \\ &= 4 \cdot c \cdot k. \end{aligned}$$

The last equation is derived from the following consideration: For all x in the convergence region of $\sum_{l=1}^{\infty} l \cdot x^{l-1}$ we are allowed to differentiate the series. Thus we get

$$\sum_{l=1}^{\infty} l \cdot x^{l-1} = \left(\sum_{l=0}^{\infty} x^l\right)' = \left(\frac{1}{1-x}\right)' = \frac{1}{(1-x)^2}$$

Hence, the expected running time of ED is linear.

Finally, the probability for ED to output $y \in \{0, \dots, n-1\}$ in the l -th round is $\epsilon_{n,l}/n$. Therefore, the probability for ED to output y is

$$\frac{1}{n} \cdot \frac{n}{2^k} \sum_{l=1}^{\infty} \left(1 - \frac{n}{2^k}\right)^{l-1} = \frac{1}{n}.$$

Example 1.3.2 Frequently we will have to use the following machine, PED, which has polynomial running time and which outputs any $y \in \{0, \dots, n-1\}$ on input of $(n, w) \in \mathbb{N}_{\geq 1}^2$ with a probability which is approximately $\frac{1}{n}$.

PED operates exactly as ED except that PED never performs more than w rounds. This means that even if PED in the w -th round generates a random number y which is too large, it will output $y = 0$ and then terminate. Consequently the running time of PED is linear in $\log n$ and w .

Moreover, we have for $1 \leq y \leq n-1$

$$\begin{aligned} \Pi_{PED}((n, w), y) &= \frac{1}{n} \cdot \frac{n}{2^k} \cdot \sum_{l=1}^w \left(1 - \frac{n}{2^k}\right)^{l-1} \\ &= \frac{1}{n} \left[1 - \left(1 - \frac{n}{2^k}\right)^w\right] \end{aligned}$$

and so

$$\frac{1}{n} \left(1 - \frac{1}{2^w}\right) \leq \Pi_{PED}((n, w), y) \leq \frac{1}{n}.$$

Finally

$$\Pi_{PED}((n, w), 0) = \frac{1}{n} \left[1 - \left(1 - \frac{n}{2^k}\right)^w\right] + \left(1 - \frac{n}{2^k}\right)^w$$

and so

$$\frac{1}{n} \left(1 - \frac{1}{2^w}\right) \leq \Pi_{PED}((n, w), 0) \leq \frac{1}{n} \left(1 - \frac{1}{2^w}\right) + \frac{1}{2^w}.$$

Chapter 2

Interactive systems

2.1 Interactive Turing machines

An *interactive Turing machine* is a $(k + 2)$ -tape probabilistic Turing machine

$$I = ((K, \Sigma, \Delta, s), p)$$

which can participate in communications with other interactive Turing machines. The $(k + 1)$ st tape of I is its *send tape* and the $(k + 2)$ nd tape is its *receive tape*. Send and receive tapes also are called *communication tapes*.

The interactive machine I terminates its messages by sending the character \square , hence whenever we are talking about interactive Turing machines we require that $\square \in \Sigma$.

The fact that I is an interactive machine is reflected by the conditions imposed on the elements $(q, \mathbf{a}, r, \mathbf{b})$ ($\mathbf{a} = (a_1, \dots, a_{k+2}) \in \Sigma^{k+2}$, $\mathbf{b} = (b_1, \dots, b_{k+2}) \in (\Sigma \cup \{L, R\})^{k+2}$) of the transition relation Δ .

The interactive machine is not supposed to move the head on its communication tapes to the left, i.e.

$$b_{k+i} \in \Sigma \cup \{R\} \text{ for } i = 1, 2.$$

Also, I is neither allowed to change old messages, nor send $\#$, i.e.

$$\begin{aligned} b_{k+1} \in \Sigma &\Rightarrow a_{k+1} = \# \text{ or } a_{k+1} = b_{k+1}, \\ b_{k+1} = R &\Rightarrow a_{k+1} \neq \#. \end{aligned}$$

After completing a message, the machine I must look for the next message on its receive tape, i.e.

$$b_{k+1} = \square \Rightarrow b_{k+2} = R.$$

The machine I must not write on its receive tape and can only move the head on its receive tape after having received something, i.e.

$$b_{k+2} \neq R \Rightarrow a_{k+2} = b_{k+2},$$

$$b_{k+2} = R \Rightarrow a_{k+2} \neq \#.$$

Unless the computation time is used up the machine I completely reads the message on its receive tape, i.e.

$$a_{k+2} \neq \square, \# \Rightarrow b_{k+2} = R.$$

The machine I is not allowed to send a message until it has finished reading, i.e.

$$a_{k+2} \neq \square \Rightarrow b_{k+1} = a_{k+1}.$$

Those definitions are easily generalized to *interactive Turing machines with multiple communication tapes*.

2.2 Interactive systems

Let I_1 and I_2 be interactive Turing machines that satisfy some conditions we specify in the following. Those Turing machines can be combined to an *interactive system* $S = S(I_1, I_2)$. The system S is a probabilistic Turing machine $S = ((K, \Sigma, \Delta, s), p)$. To be precise, let

$$I_j = ((K_j, \Sigma, \Delta_j, s_j), p_j), \quad j \in \{1, 2\}.$$

The set of states of S is

$$K = (K_1 \cup \{h_1\}) \times (K_2 \cup \{h_2\}) - \{(h_1, h_2)\}.$$

and the start state is

$$s = (s_1, s_2).$$

The state (h_1, h_2) is identified with the halt state h , i.e. we replace each occurrence of (h_1, h_2) by h .

Assume that I_j has $k_j + 2$ tapes, $j = 1, 2$. Then S has $k = k_1 + k_2 + 2$ tapes. The first k_1 tapes correspond to the work tapes of I_1 . The next k_2 tapes correspond to the work tapes of I_2 . The $(k_1 + k_2 + 1)$ st tape is both the send tape of I_1 and the receive tape of I_2 and the $(k_1 + k_2 + 2)$ nd tape is the send tape of I_2 and the receive tape of I_1 . Thus on both the $(k_1 + k_2 + 1)$ st and the $(k_1 + k_2 + 2)$ nd tape there are two heads.

S simulates the communication between I_1 and I_2 . The transitions of S are of the form $(q, \mathbf{a}, r, \mathbf{b})$ where $q = (q_1, q_2) \in K$, $r = (r_1, r_2) \in K$,

$$\begin{aligned} \mathbf{a} &= ((a_{1,1}, \dots, a_{k_1,1}), (a_{1,2}, \dots, a_{k_2,2}), (a_{k_1+1,1}, a_{k_2+2,2}), (a_{k_2+1,2}, a_{k_1+2,1})), \\ \mathbf{b} &= ((b_{1,1}, \dots, b_{k_1,1}), (b_{1,2}, \dots, b_{k_2,2}), (b_{k_1+1,1}, b_{k_2+2,2}), (b_{k_2+1,2}, b_{k_1+2,1})), \end{aligned}$$

with $a_{i,j} \in \Sigma$, $b_{i,j} \in \Sigma \cup \{L, R\}$ ($j \in \{1, 2\}$, $1 \leq i \leq k_j + 2$).

The symbols $a_{k_1+1,1}$ and $a_{k_2+2,2}$ represent the characters under the heads of the $(k_1 + k_2 + 1)$ st tape and, the symbols $a_{k_2+1,2}$ and $a_{k_1+2,1}$ represent the characters under the heads of the $(k_1 + k_2 + 2)$ nd tape of S .

The vectors $\mathbf{a}_j = (a_{1,j}, \dots, a_{k_j+2,j})$, $\mathbf{b}_j = (b_{1,j}, \dots, b_{k_j+2,j})$, $j \in \{1, 2\}$, describe the action of the communication partners on their own tapes. Unless one of the machines reads $\#$ on its receive tape, the machine carries out the action specified in its definition.

$$\begin{aligned} a_{k_1+2,1} \neq \# &\Rightarrow (q_1, \mathbf{a}_1, r_1, \mathbf{b}_1) \in \Delta_1; \\ a_{k_2+2,2} \neq \# &\Rightarrow (q_2, \mathbf{a}_2, r_2, \mathbf{b}_2) \in \Delta_2. \end{aligned}$$

The machines I_1 and I_2 have to satisfy the following conditions:

If one of the communication partners reads $\#$ on its receiving tape, then this partner must wait until it receives the next character:

$$\begin{aligned} a_{k_1+2,1} = \# &\Rightarrow r_1 = q_1, \mathbf{b}_1 = \mathbf{a}_1; \\ a_{k_2+2,2} = \# &\Rightarrow r_2 = q_2, \mathbf{b}_2 = \mathbf{a}_2. \end{aligned}$$

As usual, the contents of a tape with two heads is written

$$u \overset{1}{\underline{a}} v \overset{2}{\underline{b}} w$$

with $u, v, w \in \Sigma^*$, $a, b \in \Sigma$.

Using the notation from above, we define the transition probability p as follows

$$\begin{aligned} &p(q, \mathbf{a}, r, \mathbf{b}) \\ &= \begin{cases} p_1(q_1, \mathbf{a}_1, r_1, \mathbf{b}_1) \cdot p_2(q_2, \mathbf{a}_2, r_2, \mathbf{b}_2) & \text{if } a_{k_1+2,1} \neq \#, a_{k_2+2,2} \neq \#; \\ p_2(q_2, \mathbf{a}'_2, r_2, \mathbf{b}'_2) & \text{if } a_{k_1+2,1} = \#, a_{k_2+2,2} \neq \#; \\ p_1(q_1, \mathbf{a}'_1, r_1, \mathbf{b}'_1) & \text{if } a_{k_1+2,1} \neq \#, a_{k_2+2,2} = \#; \end{cases} \end{aligned}$$

and $\Delta = \{(q, \mathbf{a}, r, \mathbf{b}) : p(q, \mathbf{a}, r, \mathbf{b}) > 0\}$.

Proposition 2.2.1 *The interactive system S is a probabilistic Turing machine.*

Proof: We must only verify that for every $q \in K$ and for every $\mathbf{a} \in \Sigma^{k_1+k_2+2}$ the normalization condition

$$\sum_{(q, \mathbf{a}, r, \mathbf{b}) \in \Delta(q, \mathbf{a})} p(q, \mathbf{a}, r, \mathbf{b}) = 1$$

is satisfied. This is easily seen from the definition of p . ■

If $x = (x_1, x_2) \in \Sigma_0^* \times \Sigma_0^*$ is the *input of the interactive system*, then the content of its tapes is

$$(x_1 \overset{1}{\underline{\#}}, \overset{2}{\underline{\#}}, \dots, x_2 \overset{1}{\underline{\#}}, \overset{2}{\underline{\#}}, \dots, \overset{1,2}{\underline{\#}}, \overset{1,2}{\underline{\square}}).$$

Let $\gamma_0, \gamma_1, \dots, \gamma_k$ be a computation of S . Assume that the contents of the send tape of I_j are of the form $a_{1_j} \square a_{2_j} \square \dots a_{m_j}$ where $a_{i_j} \in (\Sigma - \{\square, \#\})^*$, $1 \leq i \leq m$, $1 \leq j \leq 2$. Each a_{i_1} is called a *message* from I_1 to I_2 and each a_{i_2} is called a message from I_2 to I_1 .

If n is the number of messages from I_2 to I_1 then the number of messages from I_1 to I_2 is either n or $n + 1$.

Let γ_u be the configuration in which I_1 starts to send the message a_{i_1} to I_2 and γ_v be the configuration in which I_1 sees the terminating character \square of the message a_{i_2} received from I_2 as an “answer”, then we say the sequence $\gamma_u, \gamma_{u+1}, \dots, \gamma_v$ is the i -th interaction. More exactly we define the following: The *i -th interaction* is the sequence $\gamma_u, \gamma_{u+1}, \dots, \gamma_v$ such that γ_u is the configuration in which the first character of the message a_{i_1} is written and γ_v is either

- the first configuration that occurs while I_2 receives a_{i_1} and that contains a halted configuration of I_2 , or
- the first configuration that occurs while I_1 receives a_{i_2} and that contains a halted configuration of I_1 , or
- the first configuration that contains the terminating character \square for the message a_{i_2} .

The *number of interactions* for this computation is m_1 .

Those definitions are easily generalized to interactive systems where the communicating machines have multiple communication tapes. Those are used to model multiparty computation.

2.3 Interactive proof systems

In this section we model a situation in which a prover P wants to prove to a verifier V a certain claim. For example, proving a mathematical theorem means to show that a formula of the predicate calculus encoded as a string over some alphabet belongs to the language of valid formulas. In our model the verifier has the computing power of a human being, i.e. he is a polynomial time interactive Turing machine. When determining the computing power of the prover we must distinguish between the two possible objectives of interactive proof systems. If the prover wants to show that a string x belongs to a language L then he may have infinite computing power. He can only use polynomially many interactions to convince the verifier. But if the prover wants to convince a verifier that he knows the value of some function then this claim is only non trivial if the verifiers computing power does not allow him to determine that value.

Definition 2.3.1 *An interactive proof system is an interactive system $S = (P, V)$ where P is an interactive Turing machine and V is a polynomial time coin tossing interactive*

Turing machine. We call P the prover and V the verifier of S . The input of the interactive system S has the form (x, x) , $x \in \Sigma_0^*$. We say S accepts the input if the output is $(z, 1)$, $z \in \Sigma_0^*$. S rejects the input if the output is $(z, 0)$, $z \in \Sigma_0^*$.

We write $\Pi_S(x, 1)$ (resp. $\Pi_S(x, 0)$) instead of $\Pi_S((x, x), 1)$ (resp. $\Pi_S((x, x), 0)$).

We now show that it does not make sense to allow the prover more computing power than PSPACE.

Theorem 2.3.1 *Let V be a polynomial time interactive Turing machine. Then there exists an interactive proof system $S' = (P', V)$ with a deterministic PSPACE prover such that for every interactive proof system $S = (P, V)$ with verifier V and for every input x the inequality*

$$\Pi_{S'}(x, 1) \geq \Pi_S(x, 1)$$

holds.

Proof: Since V has polynomial running time, there is a constant $c \in \mathbb{N}$ such that for every interactive system (P, V) , for every input $x \in \Sigma_0^*$, and for every possible computation of (P, V) with input x , the number of interactions, the number of coin tosses during the k -th interaction and the length of sent messages are bound by $N = |x|^c$.

P' operates as follows:

Suppose that the system $S' = (P', V)$ is about to start its $(2k + 1)$ st interaction. Now P' has to come up with the $(2k + 1)$ st message m . This message is certainly an element of

$$M_1 = \bigcup_{i=0}^N \Sigma_0^i.$$

The set M_1 is finite and there is a PSPACE Turing machine T_1 which, on input of I^N , sequentially enumerates all the elements of M_1 .

The machine P' starts T_1 and for each choice of $m \in M_1$, determines the probability for the system to accept x during the remaining computation. This is done as follows: the remaining computation depends on the remaining messages sent by P' and the coin tosses carried out by V .

The sequence $\mu = (m_1, m_2, \dots, m_l)$ of the remaining messages belongs to the set

$$M_2 = \bigcup_{l=0}^N M_1^l.$$

Again, this set can be enumerated by means of a PSPACE Turing machine T_2 . Finally, the random string ν of all coin tosses that V carries out during the remaining computation belongs to the set

$$M_3 = \bigcup_{j=0}^{N^2} \{0, 1\}^j.$$

M_3 also can be sequentially enumerated by means of a PSPACE Turing machine T_3 . Using T_2 and T_3 , the machine P' generates sequentially all the possible combinations of remaining messages and random strings and counts the number n of accepting computations. From that number, the acceptance probability is easily computed. Since

$$n = |\Sigma|^{cN^k}$$

for some $c, k \in \mathbb{N}$ the number n can be written down in PSPACE. So the whole computation of P can be carried out in PSPACE. Once P has determined the optimal choice for m , it sends precisely that message.

This concludes the description of P' .

We must now show that for every interactive proof system $S = (P, V)$

$$\Pi_{S'}(x, 1) \geq \Pi_S(x, 1). \quad (2.1)$$

For $k \geq 0$, we let P_k be the interactive Turing machine which works exactly as P during the first k interactions, and as P' during the remaining interactions. Let S_k be the interactive system (P_k, V) .

We have

$$\Pi_{S'}(x, 1) = \Pi_{S_0}(x, 1) \text{ and } \Pi_{S_N}(x, 1) = \Pi_S(x, 1).$$

It is therefore sufficient to show that for every $k < N$

$$\Pi_{S_k}(x, 1) \geq \Pi_{S_{k+1}}(x, 1).$$

Let $k \in \mathbb{N}$. Let R_k be the set of all pairs (\mathbf{m}, ν') where $\mathbf{m} = (\mathbf{m}_1, \dots, \mathbf{m}_k) \in \mathbf{M}_1^k$ is a sequence of messages, $\nu' = (\nu'_1, \dots, \nu'_k) \in M_3^k$ is a sequence of random strings such that there is a computation of S with at least k interactions, the first k messages of P being $\mathbf{m}_1, \dots, \mathbf{m}_k$, the random strings used by V in the first k interactions being ν'_1, \dots, ν'_k .

For $r = (\mathbf{m}, \nu') \in R_k$, let $\pi_k(r)$ be the probability for S_k to accept x if the sequence of messages from P to V after k interactions is exactly \mathbf{m} and if the sequence of random strings used by the verifier during the first k interactions is exactly ν' . Then we have

$$\pi_k(r) = \max \left\{ \frac{1}{2^N} \sum_{\nu \in \{0,1\}^N} \pi_{k+1}((\mathbf{m}, m), (\nu', \nu)): m \in M_1 \right\}. \quad (2.2)$$

For $r = (\mathbf{m}, \nu') \in R_k$ we let $\lambda_k(r)$ be the probability for S to produce the message sequence \mathbf{m} and the random sequence ν' in the first k rounds. Finally, for $r \in R_k$, $m \in M_1$, we let $\kappa_k(r, m)$ be the probability for P to send the message m after k rounds of interaction if the message string generated so far is \mathbf{m} and the random string is ν' .

Using those notations we get

$$\begin{aligned}
& \Pi_{S_{k+1}}(x, 1) \\
= & \sum_{r \in R_{k+1}} \lambda_{k+1}(r) \cdot \pi_{k+1}(r) \\
= & \sum_{r=(\mathbf{m}, \nu') \in R_k} \sum_{m \in M_1} \sum_{\nu \in \{0,1\}^N} \lambda_{k+1}((\mathbf{m}, m), (\nu', \nu)) \cdot \pi_{k+1}((\mathbf{m}, m), (\nu', \nu)) \\
= & \sum_{r=(\mathbf{m}, \nu') \in R_k} \sum_{m \in M_1} \sum_{\nu \in \{0,1\}^N} \lambda_k(r) \cdot \kappa_k(r, m) \cdot \frac{1}{2^N} \cdot \pi_{k+1}((\mathbf{m}, m), (\nu', \nu)) \\
\leq & \sum_{r=(\mathbf{m}, \nu') \in R_k} \lambda_k(r) \sum_{m \in M_1} \kappa_k(r, m) \cdot \max \left\{ \frac{1}{2^N} \sum_{\nu \in \{0,1\}^N} \pi_{k+1}((\mathbf{m}, m), (\nu', \nu)) : m \in M_1 \right\} \\
= & \sum_{r \in R_k} \lambda_k(r) \cdot \pi_k(r) \\
= & \Pi_{S_k}(x, 1).
\end{aligned}$$

■

An interactive system works as follows:

The system receives an input $x \in \Sigma_0^*$. In case x belongs to a specified language $L \subseteq \Sigma_0^*$ the all powerful prover P tries to convince the verifier V , which is only polynomial time but probabilistic, about the fact that $x \in L$. Moreover the construction of the system must guarantee that it is not probable for a cheating prover P' to convince V in case $x \notin L$.

Definition 2.3.2

Let $L \subseteq \Sigma_0^*$ and $S = (P, V)$ be an interactive system with V being a CTM with polynomial running time and P a CTM. We call S an interactive proof system for L , if the following conditions are satisfied:

a) (Completeness)

The system S accepts every $x \in L$ with probability at least $\frac{2}{3}$.

b) (Correctness)

For all interactive systems $S^* = (P^*, V)$ and all $x \notin L$ the probability for S^* to accept x is at most $\frac{1}{3}$.

The class of all languages $L \subseteq \Sigma_0^*$ for which there exists an interactive proof system is denoted by IP.

Example 2.3.1 We describe an interactive proof system for graph nonisomorphism: The input to the interactive system is an encoding of two graphs G_1 and G_2 . Prover P and verifier V do the following steps:

1. V chooses randomly $i \in \{1, 2\}$.
2. Then V permutes the nodes of G_i with a randomly chosen permutation. Let \tilde{G}_i be the resulting graph. V sends an encoding of \tilde{G}_i to P .
3. P checks if \tilde{G}_i is isomorphic to G_j , $j \in \{1, 2\}$ and returns the number j to V .

The protocol is repeated twice. V accepts iff in both rounds i is equal to j .

Completeness: Because there exists no isomorphism between two nonisomorphic graphs the prover can always determine which Graph permutet, so that the acceptance probability is 1. We call this *perfect completeness*.

Correctness: If G_1 and G_2 are isomorphic the probability that P gives the right answer is always $\leq \frac{1}{2}$. The best P can do is choosing randomly the number j . The probability that P guesses twice the right j is less than $\frac{1}{3}$.

Proposition 2.3.1 $NP \subseteq IP$

Proof: Let $L \in NP$. Thus there is a function $\tau : \Sigma_0^* \times \Sigma_0^* \rightarrow \{true, false\}$ which can be evaluated in polynomial time by a deterministic Turing machine and a polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$, so that for every $x \in \Sigma_0^*$ there exists an $y \in \Sigma_0^*$, $|y| \leq p(|x|)$, satisfying $\tau(x, y) = true$, iff $x \in L$. Consider the following interactive system. The common input is $x \in \Sigma_0^*$. At first P calculates a “certificate” y for x in case $x \in L$. (This can be done by exhaustive search since $\bigcup_{i \leq p(|x|)} \Sigma_0^i$ is finite).

If $x \notin L$, P sends $m = 0$ to V .

If $x \in L$, P sends $m = y$ to V .

Then V calculates $\tau(x, m)$ and accepts and stops if $\tau(x, m) = true$. Otherwise V rejects and stops.

The system (P, V) accepts $x \in \Sigma_0^*$ with probability 1, if $x \in L$, and with probability 0, if $x \notin L$. ■

An interactive proof system stops with the wrong decision with probability $\frac{1}{3}$ at most. But this seems to be quite large. Therefore it is natural to consider systems with an asymptotically vanishing error probability.

Definition 2.3.3 Let $L \subseteq \Sigma_0^*$ and $S = (P, V)$ be an interactive system with V being a polynomial time CTM and P a CTM. We call S an asymptotic proof system for L , if the following conditions are satisfied:

a) (asymptotic completeness)

For every $k \in \mathbb{N}$

$$\lim_{\substack{|x| \rightarrow \infty \\ x \in L}} |x|^k (\Pi_S(x, 1) - 1) = 0$$

is satisfied.

b) (asymptotic correctness)

For all interactive systems $S^* = (P^*, V)$ and all $k \in \mathbb{N}$

$$\lim_{\substack{|x| \rightarrow \infty \\ x \notin L}} |x|^k \Pi_{S^*}(x, 1) = 0.$$

Theorem 2.3.2 Let $L \subseteq \Sigma_0^*$. Then there exists an asymptotic proof system for L iff there exists an interactive proof system for L .

Proof: Suppose that $S = (P, V)$ is an interactive proof system for L . The asymptotic interactive proof system for $L, S' = (P', V')$ works as follows : The prover P' uses the original prover P as a subroutine. The verifier V' uses V as a subroutine. On input of x the machines P' and V' carry out the interaction of P and V exactly $n = |x|$ times. The new verifier accepts its input, if the old verifier accepts the input at least $m = \lfloor \frac{n}{2} \rfloor$ times. Otherwise, he rejects x . Let p be the probability for the original system S to accept x . Then the probability for S' to accept x exactly i times is

$$\binom{n}{i} p^i (1-p)^{n-i}.$$

Consequently the probability for S' to accept x is

$$\begin{aligned} \Pi_{S'}(x, 1) &= \sum_{i=m}^n \binom{n}{i} p^i (1-p)^{n-i} \\ &= \sum_{i=0}^n \binom{n}{i} p^i (1-p)^{n-i} - \sum_{i=0}^{m-1} \binom{n}{i} p^i (1-p)^{n-i} \\ &= 1 - \sum_{i=0}^{m-1} \binom{n}{i} p^i (1-p)^{n-i}. \end{aligned}$$

Now put

$$f(i) = \binom{n}{i} p^i (1-p)^{n-i}.$$

Since $p \geq \frac{2}{3}$ we have for $i \leq m-1 \leq \frac{n}{2} - 1$

$$\frac{f(i)}{f(i+1)} = \frac{i+1}{n-i} \frac{(1-p)}{p} \leq \frac{1}{2}.$$

This implies that

$$\Pi_{S'}(x, 1) \geq 1 - f(m-1) \sum_{i=0}^{m-1} \left(\frac{1}{2}\right)^{m-1-i} \geq 1 - 2f(m-1).$$

Now we have to give an upper bound for $f(m-1)$. We have

$$\binom{n}{m-1} \leq 2^{n-1}.$$

Also, for $p \geq \frac{2}{3}$ we have

$$p^{m-1}(1-p)^{n-m+1} \leq 2^{m-1} \left(\frac{1}{3}\right)^n.$$

To see this consider the function $g(x) = x^{m-1}(1-x)^{n-m+1}$. The function is continuous and is monotonously decreasing for $x \geq \frac{2}{3}$.

Hence we have

$$\Pi_{S'}(x, 1) \geq 1 - \left(\frac{2^{1.5}}{3}\right)^n$$

which implies the asymptotic completeness condition. The asymptotic correctness condition can be proved analogously.

Conversely, if S is an asymptotic interactive proof system for L then it is easy to see that both the correctness and the completeness condition are satisfied for all but finitely many inputs x . Those finitely many exceptions can be encoded in P and V without changing the complexity of those machines. ■

Example 2.3.2 Look at steps one to three of the interactive proof system of graph nonisomorphism. Let n be the length of the input. The interactive proof system of graph nonisomorphism do the steps one to three n times. The verifier accepts if the answers of the prover are correct, otherwise he rejects. It follows that the interactive proof system has perfect completeness and so

$$\Pi_S(x, 1) = 1 \Rightarrow \lim_{\substack{|x| \rightarrow \infty \\ x \in L}} |x|^k (\Pi_S(x, 1) - 1) = 0.$$

The probability to convince a verifier with a wrong input is $\frac{1}{2^n}$. Hence we have

$$\lim_{\substack{|x| \rightarrow \infty \\ x \notin L}} |x|^k \Pi_{S^*}(x, 1) = \lim_{n \rightarrow \infty} \underbrace{n^k \frac{1}{2^n}}_{\text{L'Hospital}} = 0.$$

Theorem 2.3.3 *Let $L \in \Sigma_0^*$ and assume that there is an interactive proof system for L . Then there exists an interactive proof system for L with a PSPACE prover.*

Proof: Let $S = (P, V)$ be an interactive proof system for L . Let $S' = (P', V)$ be the interactive system from Theorem 2.3.1. Then it follows that S' satisfies the completeness condition. The correctness condition for S' is satisfied since it is satisfied for S . ■

As a consequence of Theorem 2.3.1 we obtain

Theorem 2.3.4 *If a language $L \in \Sigma_0^*$ has an interactive proof system then it has an interactive proof system with a deterministic PSPACE prover.*

Corollary 2.3.1 $IP \subseteq PSPACE$.

Proof: Let $S = (P, V)$ be an interactive proof system for the language $L \in \Sigma_0^*$ with deterministic PSPACE prover P . Then S is a probabilistic PSPACE device for deciding L which can be simulated by a deterministic PSPACE machine deciding L . ■

We now prove

Theorem 2.3.5 $IP = PSPACE$

In order to do so we introduce the language of all quantified boolean formulas (QBF). This language is PSPACE-complete. We then present an interactive proof system for that language.

The language QBF is constructed over $\Sigma_B = \{\perp, \top, v, I, \$, \wedge, \vee, \neg, \forall, \exists, (,)\}$. For $n \in \mathbb{N}$ the n -th boolean variable is $x_n = vI^n\$$. The set of all boolean variables is denoted by \mathcal{X} . QBF is inductively defined as follows:

- (a) The constants \perp, \top are in QBF.
- (b) For $n \in \mathbb{N}$ each boolean variable x_n is in QBF and so is its *negation* $\neg x_n$.
- (c) If F and G are in QBF then so are $(F \vee G), (F \wedge G), \neg F$.
- (d) If F is in QBF and $x \in \mathcal{X}$ then $\forall xF$ and $\exists xF$ is in QBF.

A *subformula* of $F \in \text{QBF}$ is a substring of F which is in QBF. The *free variables* are inductively defined as follows:

- (a) For $x \in \mathcal{X}$ the only free variable in x and $\neg x$ is x .
- (b) For $F, G \in \text{QBF}$ the free variables of $(F \wedge G)$ or $(F \vee G)$ are the free variables of F and the free variables of G .
- (c) For $F \in \text{QBF}$ the free variables of $\neg F$ are the free variables of F .
- (d) For $F \in \text{QBF}$ the free variables of $\forall xF$ and $\exists xF$ are the free variables of F except x .

An occurrence of a variable x in $F \in \text{QBF}$ is a *bound occurrence* if it is an occurrence within a subformula of F of the form $\forall xG$ or $\exists xG$. Any other occurrence of x in F is a *free occurrence*. We say that the indicated occurrence of \forall and \exists binds each free occurrence of x in F . A formula $F \in \text{QBF}$ is called *closed* if there are no free occurrences of variables in F . If F contains a subformula of the form QxG , where G is a QBF and $Q \in \{\forall, \exists\}$, then G is called the *scope* of the quantifier Q .

Example 2.3.3 The formulas \perp , y , $\forall x\top$, $\neg((x \wedge \top) \vee y)$, $\forall x \exists y(x \vee y)$, $(\top \wedge \perp)$ are in QBF.

\perp , $\forall x\top$, $\forall x \exists y(x \vee y)$, $(\top \wedge \perp)$ are closed formulas.

The subformulas of $\forall x \exists y(x \vee y)$ are x , y , $(x \vee y)$, $\exists y(x \vee y)$, $\forall x \exists y(x \vee y)$.

The free variables of $(x \vee y)$ are x and y . In $\exists y(x \vee y)$ has x a free and y a bound occurrence.

The scope of \forall in $\forall x \exists y(x \vee y)$ is $\exists y(x \vee y)$. The scope of \exists is $(x \vee y)$.

Let $F, G \in \text{QBF}$, $x \in \mathcal{X}$. Then the formula $F[x/G]$ is obtained from F by replacing each free occurrence of x with G .

We define Λ to be the set of all QBF.

A *boolean assignment* is a mapping

$$V : \mathcal{X} \rightarrow \{0, 1\}.$$

This assignment can be extended to Λ as follows:

We define

$$\begin{aligned} \neg 0 &= 1, \\ \neg 1 &= 0, \\ 0 \wedge 0 &= 0 \wedge 1 = 1 \wedge 0 = 0, \\ 1 \wedge 1 &= 1, \\ 0 \vee 1 &= 1 \vee 0 = 1 \vee 1 = 1, \\ 0 \vee 0 &= 0. \end{aligned}$$

We set $V(\perp) = 0$, $V(\top) = 1$. For $F \in \Lambda$ we define $V(\neg F) = \neg V(F)$. For $F, G \in \Lambda$ we set $V(F \wedge G) = (V(F) \wedge V(G))$, $V(F \vee G) = (V(F) \vee V(G))$. For $x \in \mathcal{X}$ and $F \in \Lambda$ we define:

$$\begin{aligned} V(\forall xF) &= V(F[x/\perp] \wedge F[x/\top]), \\ V(\exists xF) &= V(F[x/\perp] \vee F[x/\top]). \end{aligned}$$

Example 2.3.4 Let V be a boolean assignment, $V(x) = 1$ and $V(y) = 0$.

Then we get for the formulas of the previous example:

$$\begin{aligned}
V(\perp) &= 0, \\
V(y) &= 0, \\
V(\forall x \top) &= V(\top \wedge \top) = V(\top) \wedge V(\top) = 1 \wedge 1 = 1, \\
V(\neg((x \wedge \top) \vee y)) &= \neg V((x \wedge \top) \vee y) = \neg(V(x \wedge \top) \vee V(y)) \\
&= \neg((V(x) \wedge V(\top)) \vee 0) = \neg((1 \wedge 1) \vee 0) = 0, \\
V(\forall x \exists y(x \vee y)) &= V(\exists y(\perp \vee y) \wedge \exists y(\top \vee y)) \\
&= (V(\exists y(\perp \vee y)) \wedge V(\exists y(\top \vee y))) \\
&= (V((\perp \vee \perp) \vee (\perp \vee \top)) \wedge V((\top \vee \perp) \vee (\top \vee \top))) \\
&= \dots = 1
\end{aligned}$$

Two QBF F and F' are called *equivalent* if $V(F) = V(F')$ for every boolean assignment V . A QBF F is called to be *prenex* if F is of the form

$$F = Q_1 x_1 \dots Q_n x_n G$$

for $Q_i \in \{\forall, \exists\}$, $1 \leq i \leq n$, and G has no quantifiers.

Proposition 2.3.2 *Let $F \in \Lambda$. There is a QBF F' in prenex form which is equivalent to F . Moreover F' can be constructed in time $O(|F|^k)$ for some $k \in \mathbb{N}$.*

Proof: See [2]. ■

Proposition 2.3.3 *Let V_1, V_2 be boolean assignments and let F be a closed QBF then we have $V_1(F) = V_2(F)$.*

Proof: We use induction on the number n of quantifiers in F . If F is a closed formula without any quantifier then there are no occurrences of variables in F and we have for all boolean assignments $V_1(F) = V_2(F)$. Let $n \geq 1$: according to Proposition 2.3.2 there is a QBF $F' = QxG$, $Q \in \{\forall, \exists\}$, $G \in \Lambda$ in prenex form which is equivalent to F . We have two cases:

- $Q = \forall$ then

$$\begin{aligned}
V_1(F) &= V_1(F') = V_1(QxG) \\
&= V_1(G[x/\perp] \wedge G[x/\top]) \\
&= (V_1(G[x/\perp]) \wedge V_1(G[x/\top]))
\end{aligned}$$

$G[x/\perp]$ and $G[x/\top]$ are both closed formulas in QBF with less than n quantifiers. Therefore we have

$$V_1(G[x/\perp]) = V_2(G[x/\perp]), \quad \text{and} \quad V_1(G[x/\top]) = V_2(G[x/\top])$$

and

$$\begin{aligned} V_1(F) &= (V_2(G[x/\perp]) \wedge V_2(G[x/\top])) \\ &= V_2(G[x/\perp] \wedge G[x/\top]) \\ &= V_2(F) \end{aligned}$$

- $Q = \exists$ analogous. ■

For a closed QBF F we call the value $V(F)$ which is assigned to F for every boolean assignment V the *truth value* of F . The language of all closed QBF which are true is denoted by Λ_{\top} .

Proposition 2.3.4 *The language Λ_{\top} is PSPACE-complete.*

Proof: See literature: [1], 171–172. ■

A QBF F is called *simple* if it satisfies the following conditions:

- (a) Each variable in F is bound by at most one quantifier.
- (b) If $\neg G$ is a subformula of F then G is a boolean variable.
- (c) A variable that has a bound occurrence never has a free occurrence.
- (d) If F contains a quantifier then F is of the form $F = QxG$ for $Q \in \{\forall, \exists\}$ and $x \in \mathcal{X}$.
- (e) If x is a variable in F and if F is of the form $F = S_1QxS_2xS_3$ for $Q \in \{\forall, \exists\}$ and for $S_i \in \Sigma_B^*$, $1 \leq i \leq 3$, then S_2 contains at most one universal quantifier \forall .
- (f) If x is a variable in F and F is of the form $F = S_1QxGS_2$ for $S_1, S_2 \in \Sigma_B^*$, $Q \in \{\forall, \exists\}$ and for a subformula G , then S_2 consists only of right parentheses or $S_2 = \varepsilon$.

Proposition 2.3.5 *Each QBF of length n can be transformed in polynomial time into an equivalent simple QBF whose length is polynomial in n .*

Proof: Let F be a QBF. It is easy to fulfill condition (b) by application of de Morgan's law. This formula can be transformed into prenex form in polynomial time. So we may assume without loss of generality that F is prenex. This implies that all the conditions with the possible exception of (e) of the definition of a simple QBF are satisfied. For a bound variable x of F we denote by $m(F, x)$ the maximum number of universal quantifiers separating the binding quantifier Q and any occurrence of x . Choose a bound variable x of F with $m(F, x) > 1$. Among all the subformulas of the form $\forall u \exists x' G$ in which x has a free occurrence choose one. Replace that subformula with $\forall u \exists x' (((x \wedge x') \vee (\neg x \wedge \neg x')) \wedge G')$ where x' is a variable not occurring in F and $G' = G[x/x']$. The new formula F' is obviously equivalent to F . Also, F' satisfies the conditions (a), (b), (c) of the definition of a simple QBF and we have

$$m(F', x) = 1$$

and

$$m(F', x') = m(F, x) - 1.$$

If we apply this procedure to all variables x in F with maximum $m(F, x)$ then we have reduced $m(F, x)$ by one. Applying this procedure repeatedly we obtain a simple QBF. The number of applications necessary to transform F into a simple formula is

$$\sum_{x \in \mathcal{X}} (m(F, x) - 1) \leq |F|^2.$$

The length of the variables we are using is $O(|F|^2)$. Hence, each time we apply the procedure we add $O(|F|^2)$ characters to the formula. Therefore, the length of the resulting formula is $O(|F|^4)$. ■

The language of all closed simple QBF which are true is denoted by Λ_{\top}^* .

Example 2.3.5

Given the formula $F = \exists x \forall y (\neg \exists z (y \vee (y \wedge z)) \wedge \forall z (\neg z \wedge x))$ we can do the following steps to construct an equivalent simple formula F' :

- Changing the variables: $F_1 = \exists x \forall y (\neg \exists z_1 (y \vee (y \wedge z_1)) \wedge \forall z_2 (\neg z_2 \wedge x))$. F_1 satisfies condition (a).

•

$$\begin{aligned} F_1 &= \exists x \forall y (\neg \exists z_1 (y \vee (y \wedge z_1)) \wedge \forall z_2 (\neg z_2 \wedge x)) \\ &\equiv \exists x \forall y (\forall z_1 \neg (y \vee (y \wedge z_1)) \wedge \forall z_2 (\neg z_2 \wedge x)) \\ &\equiv \exists x \forall y (\forall z_1 (\neg y \wedge \neg (y \wedge z_1)) \wedge \forall z_2 (\neg z_2 \wedge x)) \\ &\equiv \exists x \forall y (\forall z_1 (\neg y \wedge (\neg y \vee \neg z_1)) \wedge \forall z_2 (\neg z_2 \wedge x)) = F_2 \end{aligned}$$

F_2 satisfies the conditions (a), (c) and (d).

•

$$\begin{aligned} F_2 &= \exists x \forall y (\forall z_1 (\neg y \wedge (\neg y \vee \neg z_1)) \wedge \forall z_2 (\neg z_2 \wedge x)) \\ &\equiv \exists x \forall y \forall z_1 \forall z_2 ((\neg y \wedge (\neg y \vee \neg z_1)) \wedge (\neg z_2 \wedge x)) = F_3 \end{aligned}$$

F_3 is a prenex formula and satisfies the conditions (a), (c), (d) and (f).

• According to Proposition 2.3.5 we get:

$$\begin{aligned} F_3 &= \exists x \forall y \forall z_1 \forall z_2 ((\neg y \wedge (\neg y \vee \neg z_1)) \wedge (\neg z_2 \wedge x)) \\ &\equiv \exists x \forall y \exists x' (((x \wedge x') \vee (\neg x \wedge \neg x')) \wedge \\ &\quad \forall z_1 \exists y' (((y \wedge y') \vee (\neg y \wedge \neg y')) \wedge \forall z_2 ((\neg y' \wedge (\neg y' \vee \neg z_1)) \wedge (\neg z_2 \wedge x')))) \\ &\equiv \exists x \forall y \exists x' (((x \wedge x') \vee (\neg x \wedge \neg x')) \wedge \forall z_1 \exists x'' (((x' \wedge x'') \vee (\neg x' \wedge \neg x'')) \\ &\quad \wedge \exists y' (((y \wedge y') \vee (\neg y \wedge \neg y')) \wedge \forall z_2 ((\neg y' \wedge (\neg y' \vee \neg z_1)) \\ &\quad \wedge (\neg z_2 \wedge x'')))) = F_4 \end{aligned}$$

F_4 is a simple formula but F_4 is no longer prenex.

Corollary 2.3.2 *The language Λ_{\top}^* is PSPACE-complete.*

The goal is now to describe an interactive proof system for Λ_{\top}^* . For this purpose we need an arithmetic condition for the validity of a QBF. To each simple QBF F with free variables x_1, \dots, x_n we assign the *characteristic polynomial* $p_F \in \mathbb{Z}[X_1, \dots, X_n]$. This is done inductively as follows: for $i \in \mathbb{N}$ the i -th variable is $X_i = VI^n\$$.

(a) $p_{\top} = 1, p_{\perp} = 0$.

(b) $p_{x_i} = X_i, 1 \leq i \leq n$.

(c) For $F, G \in \Lambda$ we define $p_{F \wedge G} = p_F \cdot p_G, p_{F \vee G} = (p_F + p_G), p_{\neg F} = (1 - p_F)$.

(d) For $F \in \Lambda, x \in \mathcal{X}$ we set $p_{\forall x F} = (p_{F[x/\perp]} \cdot p_{F[x/\top]}), p_{\exists x F} = (p_{F[x/\perp]} + p_{F[x/\top]})$.

In particular we have $p_F \in \mathbb{Z}$ for any closed formula F .

Example 2.3.6 Let $F = \exists x \forall y ((\neg x \wedge z) \vee \neg y)$ and $p_z = z$.

$$\begin{aligned} p_F &= (p_{\forall y ((\neg \perp \wedge z) \vee \neg y)} + p_{\forall y ((\neg \top \wedge z) \vee \neg y)}) \\ &= ((p_{((\neg \perp \wedge z) \vee \neg \perp)} \cdot p_{((\neg \perp \wedge z) \vee \neg \top)}) + (p_{((\neg \top \wedge z) \vee \neg \perp)} \cdot p_{((\neg \top \wedge z) \vee \neg \top)})) \\ &= (((p_{(\neg \perp \wedge z)} + p_{\neg \perp}) \cdot (p_{(\neg \perp \wedge z)} + p_{\neg \top})) + ((p_{(\neg \top \wedge z)} + p_{\neg \perp}) \cdot (p_{(\neg \top \wedge z)} + p_{\neg \top}))) \\ &= (((((1 - 0) \cdot z) + (1 - 0)) \cdot (((1 - 0) \cdot z) + (1 - 1))) + \\ &\quad (((1 - 1) \cdot z) + (1 - 0)) \cdot (((1 - 1) \cdot z) + (1 - 1)))) \end{aligned}$$

Proposition 2.3.6 *Let F be a closed simple QBF. Then F is true if and only if $p_F \neq 0$.*

Proof: We show by structural induction that for every simple QBF G with free variables x_1, \dots, x_n and for every boolean assignment V we have

$$V(G) = 1 \iff p_G(V(x_1), \dots, V(x_n)) \neq 0.$$

This equivalence implies the assertion of the proposition.

The equivalence is obviously true if G is just a boolean constant or a boolean variable.

If $G = (G_1 \wedge G_2)$, $G = (G_1 \vee G_2)$, $G = \forall x G_1$ or $G = \exists x G_1$ then the assertion can be easily verified.

Finally, if $G = \neg G_1$ then by the simplicity of G it follows that $G_1 = x$ which again yields the assertion. \blacksquare

Unfortunately, the criterion for the truth of a simple QBF given in the previous proposition cannot be checked in polynomial time since the size of the coefficients of the characteristic polynomial of a QBF may be exponential in the size of the formula.

Proposition 2.3.7 *Let F be a closed simple QBF of size n . Then the integer value $p_F = O(2^{2^n})$.*

Proof: We show by structural induction that for any QBF G with free variables x_1, \dots, x_n and for every choice $(r_1, \dots, r_n) \in \{\top, \perp\}^n$ we have

$$p_{G[x_1/r_1, \dots, x_n/r_n]} \leq 2^{2^{|G|}}.$$

This is clear if $G \in \{\top, \perp\} \cup \{x_i : i \in \mathbb{N}\} \cup \{\neg x_i : i \in \mathbb{N}\}$.

If $G = (G_1 \wedge G_2)$ then

$$\begin{aligned} p_{G[x_1/r_1, \dots, x_n/r_n]} &= (p_{G_1[x_1/r_1, \dots, x_n/r_n]} \cdot p_{G_2[x_1/r_1, \dots, x_n/r_n]}) \\ &\leq 2^{2^{|G_1|}} \cdot 2^{2^{|G_2|}} \\ &= 2^{2^{|G_1|} + 2^{|G_2|}} \\ &\leq 2^{2^{|G|}}. \end{aligned}$$

If $G = (G_1 \vee G_2)$ we can argue analogously.

If $G = \forall y G_1$:

$$\begin{aligned} p_{G[x_1/r_1, \dots, x_n/r_n]} &= (p_{G_1[x_1/r_1, \dots, x_n/r_n, y/\top]} \cdot p_{G_1[x_1/r_1, \dots, x_n/r_n, y/\perp]}) \\ &\leq 2^{2^{|G_1|-2}} \cdot 2^{2^{|G_1|-2}} \\ &= 2^{2^{|G_1|-1}} \\ &< 2^{2^{|G|}}. \end{aligned}$$

If $G = \exists y G_1$ we can use an analogous argument. \blacksquare

Proposition 2.3.8 *Let F be a closed simple QBF of size n . Then there exists a prime number p of length polynomial in n , $p \geq 2n^2$, such that $p_F \not\equiv 0 \pmod p$ if and only if F is true.*

Proof: If F is false then $p_F = 0$ by Proposition 2.3.6 and hence we have $p_F \equiv 0 \pmod p$ for every prime number p .

Conversely, let F be true and $c \in \mathbb{N}$ and assume that for every prime number p with $\log p \leq (\log e)n^c$ we have $p_F \equiv 0 \pmod p$. Since by a theorem of Rosser and Schönfeld

$$e^{x \log x} = x^x \geq x! \geq \prod_{p \leq x} p \geq e^{\frac{x}{2}}$$

we have

$$\begin{aligned} \prod_{2n^2 < p \leq \exp(n^c)} p &\geq \frac{\exp(\frac{1}{2}e^{n^c})}{\prod_{2n^2 \geq p} p} \\ &\geq \exp(\frac{1}{2}e^{n^c} - 2n^2 \log(2n^2)) \\ &= \Omega(2^{2^{n^c}}). \end{aligned}$$

Since $p_F \equiv 0 \pmod p$ for all p with $\log p \leq (\log e) \cdot n^c$ we have

$$\prod_{2n^2 < p \leq \exp(n^c)} p | p_F.$$

For sufficiently large $c \geq 2$ we therefore have the contradiction

$$\Omega(2^{2^{n^c}}) = \prod_{2n^2 < p \leq \exp(n^c)} p \leq p_F = O(2^{2^n}).$$

■

We now prove

Theorem 2.3.6 $\Lambda_{\top}^* \subseteq IP$.

Proof: Let F be a closed simple QBF and suppose that F is true. Let x_1, \dots, x_n be the variables in F in the order of their quantification from left to right and for $0 \leq i \leq n$ let F_i be the formula obtained from F by removing the first i quantifiers. Also, put $G_0 = F$ and for $1 \leq i \leq n$ let G_i be the scope of the i -th quantifier of F .

We describe now an interactive proof system $S = (P, V)$ for Λ_{\top}^* :

At first the verifier checks whether F is simple. If not he rejects. Otherwise the prover determines $g_0 = p_F$ and the smallest prime number $p > 2|F|^2$ such that

$$g_0 \not\equiv 0 \pmod p.$$

In the initialization step of the protocol the prover sends $g_0 \bmod p$ and a primality proof for p to the verifier. The language $\mathbb{P} = \{p : p \text{ is prime}\}$ is an element of NP . Therefore the prover can find in polynomial space a witness for the primality of p and moreover the verifier can check it in polynomial time (see [3]). The verifier checks whether $g_0 \not\equiv 0 \bmod p$. During the remaining part of the protocol, the prover must demonstrate that in fact

$$g_0 = p_F$$

holds.

Since there is one unbound variable in G_1 , the variable x_1 , the characteristic polynomial p_{G_1} has one argument. For simplicity we use the variable z here and in the analogue case in the following as the argument variable.

In the first step of the protocol the prover sends $g_1(z) \equiv p_{G_1}(z) \bmod p$ to the verifier. The verifier uses $g_1(z)$ to construct $g_0 \bmod p$. This is possible since F is of the form

$$F = Qx_1G_1, Q \in \{\forall, \exists\}$$

so

$$g_0 \equiv \begin{cases} g_1(0) + g_1(1) & \text{if } Q = \exists; \\ g_1(0) \cdot g_1(1) & \text{if } Q = \forall. \end{cases}$$

Thus at this point, the proof is complete if the verifier is convinced that

$$g_1(z) \equiv p_{G_1}(z) \bmod p.$$

This proof is given recursively.

In the $(i + 1)$ st step the verifier sends a random number $r_i \in \{0, \dots, p - 1\}$ to the prover. The prover determines $g_{i+1}(z) = p_{G_{i+1}}(r_1, \dots, r_i, z) \bmod p$ and sends that polynomial to the verifier. The verifier constructs in polynomial time the value $p_{G_i}(r_1, \dots, r_i) \bmod p$ from that polynomial and if that value coincides with $g_i(r_i)$ then he is convinced that g_i is correct under the condition that g_{i+1} is.

In the $(n + 1)$ st step finally, the verifier chooses a random number $r_n \in \{0, \dots, p - 1\}$ and computes the value of $p_{G_n}(r_1, \dots, r_n) \bmod p$ which he can calculate directly from the formula F without any help of the prover. This can be done in polynomial time as Lemma 2.3.3 states. The verifier accepts if the calculated value $p_{G_n}(r_1, \dots, r_n) \bmod p$ and $g_n(r_n)$ are equivalent.

In order to show that the interactive system that was just described is in fact an interactive proof system we must show that the verifier is a polynomial time coin tossing machine, that each true simple QBF is accepted with high probability, and that each false simple QBF is rejected with high probability.

As to the polynomiality of the verifier we prove

Lemma 2.3.1 *Let F be a simple QBF. For a free variable x in F let $d(F, x)$ be the maximum number of universal quantifiers appearing on the left of an occurrence of x in F . Then*

$$\deg_x p_F \leq 2^{d(F,x)} |F|.$$

Proof: By structural induction:

If $F \in \{\top, \perp\} \cup \{x_i : i \in \mathbb{N}\} \cup \{\neg x_i : i \in \mathbb{N}\}$ then the assertion is trivial.

If $F = (F_1 \wedge F_2)$ then $d(F_i, x) \leq d(F, x), i = 1, 2$. Hence we have

$$\begin{aligned} \deg_x p_F &= \deg_x (p_{F_1} \cdot p_{F_2}) \\ &\leq \deg_x p_{F_1} + \deg_x p_{F_2} \\ &\leq 2^{d(F_1,x)} |F_1| + 2^{d(F_2,x)} |F_2| \\ &\leq 2^{d(F,x)} (|F_1| + |F_2|) \\ &\leq 2^{d(F,x)} |F|. \end{aligned}$$

If $F = (F_1 \vee F_2), F = \exists y F_1$ then the proof is analogous.

If $F = \forall y F_1$ then $d(F_1, x) = d(F, x) - 1$. So we have

$$\begin{aligned} \deg_x p_F &\leq \deg_x p_{F_1[y/\top]} + \deg_x p_{F_1[y/\perp]} \\ &\leq 2 \cdot 2^{d(F_1,x)} |F_1| \\ &\leq 2^{d(F,x)} |F|. \end{aligned}$$

■

Lemma 2.3.2 *Let F be a simple QBF without quantifiers, let x_1, \dots, x_n be the free variables of F , let $m \in \mathbb{N}$, $r_1, \dots, r_n \in \{0, \dots, m-1\}$. Then the value $p_F(r_1, \dots, r_n) \bmod m$ can be computed by using at most $|F|$ arithmetic operations in $\mathbb{Z}/m\mathbb{Z}$.*

Proof: By structural induction.

If $F \in \{\top, \perp\} \cup \{x_i : i \in \mathbb{N}\} \cup \{\neg x_i : i \in \mathbb{N}\}$ then the assertion is trivial.

If $F = (F_1 \wedge F_2)$, then the number of arithmetic operations necessary to evaluate $p_F(r_1, \dots, r_n)$ is at most $|F_1| + |F_2| + 1 \leq |F|$. The same is true if $F = (F_1 \vee F_2)$.

Lemma 2.3.3 *Let F be a QBF of the form $F = S_1 QxGS_2$ where Q is a quantifier, where G is a subformula of F such that QxG is simple, where $S_1, S_2 \in \Sigma_B^*$ such that S_1 is free of quantifiers and S_2 consists only of right parentheses. Let $m \in \mathbb{N}$. Let x_1, \dots, x_i be the free variables in F and let $r_1, \dots, r_i \in \{0, \dots, m\}$. Then the degree of the polynomial $g(z) = p_G(r_1, \dots, r_i, z)$ is at most $2|F|$. Moreover, if the values $g(0) \bmod m$ and $g(1) \bmod m$ are known then the value $p_F(r_1, \dots, r_i)$ can be determined by using $|S_1| + |S_2| + 1$ arithmetic operations in $\mathbb{Z}/m\mathbb{Z}$.*

Proof: Since the formula QxG is simple it follows that the number of universal quantifiers in G appearing on the left of an occurrence of x is at most one. Hence, by Lemma 2.3.1 it follows that $\deg g(z) \leq 2|G| \leq 2|F|$.

The rest is proved by induction on $|S_2|$.

If $|S_2| = 0$ then $F = QxG$ and so

$$p_F(r_1, \dots, r_i) = \begin{cases} g(0) + g(1) & \text{if } Q = \exists; \\ g(0) \cdot g(1) & \text{if } Q = \forall \end{cases}$$

where r_1, \dots, r_i are the values of the free variables which implies the assertion.

Suppose that the assertion has been proved for $|S_2| \leq n$ and let $|S_2| = n + 1$.

Then we can write

$$F = (F_1 \wedge F_2)$$

or

$$F = (F_1 \vee F_2).$$

According to the special form of F the formula QxG must be a subformula of F_2 , more precisely we can write

$$F_2 = S'_1 QxGS'_2$$

where S'_1 and S'_2 have the same properties as S_1 and S_2 but $|S'_2| = |S_2| - 1$. Hence we can evaluate $p_{F_2}(r_1, \dots, r_i)$ by using $|S'_1| + |S'_2| + 1$ operations in $\mathbb{Z}/m\mathbb{Z}$. But since F_1 is free of quantifiers we see from Lemma 2.3.2 that $p_{F_1}(r_1, \dots, r_i)$ can be evaluated using at most $|F_1|$ arithmetic operations in $\mathbb{Z}/m\mathbb{Z}$. Since

$$p_F(r_1, \dots, r_i) = p_{F_1}(r_1, \dots, r_i) + p_{F_2}(r_1, \dots, r_i)$$

or

$$p_F(r_1, \dots, r_i) = p_{F_1}(r_1, \dots, r_i) \cdot p_{F_2}(r_1, \dots, r_i)$$

it follows that the total number of arithmetic operations necessary to evaluate the term $p_F(r_1, \dots, r_i)$ is at most

$$|F_1| + |S'_1| + |S'_2| + 2 \leq |S_1| + |S_2| + 1.$$

■

It follows from those statements that the verifier in the interactive system is, in fact, polynomial time.

Now suppose that $F \in \Lambda_\top^*$. In that case the prover will find a prime number p such that $g_0 \not\equiv 0 \pmod{p}$ and he will be able to construct the sequence of polynomials g_i such that all the tests of the verifier are passed. So every element in the language Λ_\top is accepted with probability 1 which means that the interactive system has perfect completeness.

On the other hand, if $F \notin \Lambda_{\top}^*$ then F is not simple or $g_0 \equiv 0 \pmod p$ for every prime number p . In the first case V will reject, in the other case in order to convince the verifier of the truth of F any prover must present a wrong value for g_0 and in order to support his proof he must present a wrong polynomial g_1 . Suppose that g_i is the last polynomial given by the prover to the verifier which is incorrect. If $i < n$ then the polynomial g_{i+1} is correct. In any case, in the $(i+1)$ st step the verifier is able to compute the correct value of $p_{G_i}(r_1, \dots, r_i)$. Since $H = Q_i x_i G_i$, where $G_i \in \{\forall, \exists\}$, is simple the maximal number of universal quantifiers on the left of an occurrence of x_i in H is $d(H, x) = m(H, x) \leq 1$. By Lemma 2.3.1 the degree of $p_{G_i}(r_1, \dots, r_{i-1}, z)$ is at most $2|F|$, hence the polynomials $p_{G_i}(r_1, \dots, r_{i-1}, z)$ and $g_i(z)$ can agree at no more than $2|F|$ places. If p is chosen such that $p > 2|F|^2$ then the probability of acceptance at that point is at most $\frac{1}{|F|}$ which shows that the interactive system satisfies the correctness condition. \blacksquare

Example 2.3.7

Example to Theorem 2.3.6:

Let $F = \exists z \exists x \forall y ((\neg x \wedge z) \vee \neg y)$.

$$\begin{aligned}
p_F &= (p_{\exists x \forall y ((\neg x \wedge \perp) \vee \neg y)} + p_{\exists x \forall y ((\neg x \wedge \top) \vee \neg y)}) \\
&= (p_{\forall y ((\neg \perp \wedge \perp) \vee \neg y)} + p_{\forall y ((\neg \top \wedge \perp) \vee \neg y)}) + p_{\forall y ((\neg \perp \wedge \top) \vee \neg y)} + p_{\forall y ((\neg \top \wedge \top) \vee \neg y)} \\
&= p_{((\neg \perp \wedge \perp) \vee \neg \perp)} \cdot p_{((\neg \perp \wedge \perp) \vee \neg \top)} + p_{((\neg \top \wedge \perp) \vee \neg \perp)} \cdot p_{((\neg \top \wedge \perp) \vee \neg \top)} \\
&\quad + (p_{((\neg \perp \wedge \top) \vee \neg \perp)} \cdot p_{((\neg \perp \wedge \top) \vee \neg \top)}) + p_{((\neg \top \wedge \top) \vee \neg \perp)} \cdot p_{((\neg \top \wedge \top) \vee \neg \top)} \\
&= 2.
\end{aligned}$$

$$|F| = 11 + 2 \underbrace{|x|}_{=3} + 2 \underbrace{|y|}_{=4} + 2 \underbrace{|z|}_{=5} = 35.$$

Since p has to be bigger than $2|F|^2 = 2450$, let $p = 2459$. We remark that $2 \not\equiv 0(2459)$. P sends $g_0 \pmod p = 2$ and a certificate that p is prime. V checks that $2 \not\equiv 0(p)$.

P computes

$$\begin{aligned}
p_{G_1} &= p_{\exists x \forall y ((\neg x \wedge z) \vee \neg y)} \\
&= (p_{\forall y ((\neg \perp \wedge z) \vee \neg y)} + p_{\forall y ((\neg \top \wedge z) \vee \neg y)}) \\
&= (p_{((\neg \perp \wedge z) \vee \neg \perp)} \cdot p_{((\neg \perp \wedge z) \vee \neg \top)}) + (p_{((\neg \top \wedge z) \vee \neg \perp)} \cdot p_{((\neg \top \wedge z) \vee \neg \top)}) \\
&= (z + 1) \cdot z \\
&= z^2 + z
\end{aligned}$$

and sends $g_1(z) = z^2 + z \equiv p_{G_1}(z) \pmod p$ to V .

Since $g_1(0) = 0$ and $g_1(1) = 2$, V checks that $g_1(0) + g_1(1) \equiv g_0 \pmod p$. V chooses $r_1 \in_R \{0, \dots, p-1\}$, for example $r_1 = 923$ and sends r_1 to the verifier.

P gives $p_{G_2}(r_1, 1)$ where

$$\begin{aligned}
p_{G_2} &= P_{\forall y} ((\neg x \wedge z) \vee \neg y) \\
&= (P_{((\neg x \wedge z) \vee \neg \perp)} \cdot P_{((\neg x \wedge z) \vee \neg \top)}) \\
&= (z - zx + 1) \cdot (z - zx) \\
&= z^2 - z^2x + x - z^2x + z^2x^2 - zx \\
&= z^2x^2 - (zz^2 + z)x + z^2 + z.
\end{aligned}$$

Therefore $p_{G_2}(r_1, x) = 851929x^2 - 1704781x + 852852$.

P sends $g_2(x) = 1115x^2 + 1765x + 2038 \equiv p_{G_2}(r_1, x) \pmod p$ to V .

V checks if $g_2(0) + g_2(1) \equiv g_1(r_1) \pmod p$. This is true, because we have $g_2(0) = 2038$, $g_2(1) = 0$ and $g_1(r_1) = 852852 \equiv 2038 \pmod p$. V chooses a number $r_2 \in \{0, \dots, p-1\}$ at random, for example $r_2 = 14$, and sends r_2 to the V .

P gives $p_{G_3}(r_1, r_2, y)$ where

$$\begin{aligned}
p_{G_3} &= P_{((\neg x \wedge z) \vee \neg y)} \\
&= (1 - x)z + (1 - y) \\
&= -y + (1 - x)z + 1
\end{aligned}$$

and therefore $p_{G_3}(r_1, r_2, y) = -y - 11998$.

P sends $g_3(y) = 2458y + 297 \equiv p_{G_3}(r_1, r_2, y) \pmod p$ to V .

V checks if $g_3(0) \cdot g_3(1) \equiv g_2(r_2) \pmod p$. This congruence can be verified, because it is $g_3(0) = 297$, $g_3(1) = 296$, $297 \cdot 296 = 87912 \equiv 1847$ and $g_2(14) = 245288 \equiv 1847$. V chooses $r_3 \in_R \{0, \dots, p-1\}$, for example $r_3 = 1001$, and computes $p_{G_3}(r_1, r_2, r_3)$ and $g_3(r_3)$. It is $p_{G_3}(r_1, r_2, r_3) = -12999 \equiv 1755$ and $g_3(r_3) = 2460755 \equiv 1755$, which means that V accepts.

Corollary 2.3.3 $PSPACE \subseteq IP$.

Proof: Let $L \in PSPACE$. The following system $S = (P, V)$ is an interactive proof system for L . Since Λ_{\top}^* is PSPACE - complete there is a polynomial time reduction τ from L to Λ_{\top}^* . On input x both P and V calculate $\tau(x) \in \Sigma_B^*$ and then start the interactive proof system for Λ_{\top}^* with input $\tau(x)$. ■

Corollary 2.3.4 *Every language in IP is accepted by an interactive proof system having perfect completeness.*

Theorem 2.3.7 *There is a language in IP for which there exists no accepting interactive proof system having perfect correctness.*

2.4 Arthur-Merlin-Games

For a function $T: \mathbb{N} \rightarrow \mathbb{N}$, $IP[T]$ is the class of all languages which are accepted by interactive proof systems which, on input of $x \in \Sigma^*$ perform at most $T(|x|)$ interactions.

An *Arthur-Merlin game* is a particular interactive proof system. The additional condition is that the verifier A (Arthur) has to transmit the result of all its coin tosses to the prover M (Merlin).

For a function $f: \mathbb{N} \rightarrow \mathbb{N}$, we denote by $AM[f]$ the class of all languages that are accepted by an Arthur-Merlin game which, on input of $x \in \Sigma^*$, performs at most $f(|x|)$ interactions. In particular, if $f(n) = k$ for every $n \in \mathbb{N}$ and some constant $k \in \mathbb{N}$ we write $AM[f] = AM[k]$.

The proofs of the following theorems will be given in the full version.

Theorem 2.4.1 *If $k \geq 2$ then $AM[k] = AM[2]$.*

Theorem 2.4.2 *For every polynomial $\mathcal{U} \in \mathbb{N}[x]$, we have $IP[\mathcal{U}] \subseteq AM[\mathcal{U} + 2]$.*

Chapter 3

Distinguishing probability distributions

In order to be able to talk about zero knowledge proofs we have to define precisely what we mean by “a verifier V learns nothing from an interaction”. This is done as follows: Everything that is learned from an interaction must be learned from the sequence of messages of the prover. Since the prover is a probabilistic Turing machine, this sequence is in fact a family of random variables which are parameterized by the input string. If a probabilistic Turing machine that has the same computational power as the verifier can produce an output distribution which is indistinguishable for V from the distribution of the message sequences then we consider the verifier to have learned nothing. This is why we talk about distinguishing probability functions in this section.

3.1 Notions of distinguishability

Definition 3.1.1 *A probability function on a countable set M is a function $p : 2^M \rightarrow [0, 1]$ with $p(M) = 1$ and which for any two disjoint subsets A_1, A_2 of M satisfies $p(A_1 \cup A_2) = p(A_1) + p(A_2)$. For $a \in M$, we write $p(a)$ for $p(\{a\})$.*

Let $L \subseteq \Sigma^*$ be a language. The distinguishability of two families $\{U_x\}_{x \in L}, \{V_x\}_{x \in L}$ of probability functions on Σ^* depends on the power of the distinguisher. If he is able to apply statistical tests then he will be able to differentiate between more such families than if he were only able to use a polynomial time algorithm. Between those two extremes there is the possibility of using different circuits for each input x . This is still more powerful than using the same algorithm for every input but less powerful than using statistical tests. Here are the precise notions:

3.2 Indistinguishability

Definition 3.2.1 Let U and V be two probability distributions on Σ^* .

The series

$$\delta_S(U, V) = \sum_{y \in \Sigma^*} |U(y) - V(y)|$$

is called the statistical difference between U and V .

In general it is impossible to determine in polynomial time that two probability distributions have a non zero statistical difference. Therefore one uses tools like probabilistic circuits and probabilistic algorithms (i.e. probabilistic Turing machines) to distinguish between probability distributions.

Definition 3.2.2 Let U and V be two probability distributions on Σ^* . For a probabilistic circuit with m input nodes and one output node we call

$$\delta_C(U, V) = \left| \sum_{y \in \Sigma^m} \Pi_C(y, 1)(U(y) - V(y)) \right|$$

the circuit difference between U and V with respect to C .

Finally, for a probabilistic Turing machine Z we call the series

$$\delta_Z(U, V) = \left| \sum_{y \in \Sigma^*} \Pi_Z(y, 1)(U(y) - V(y)) \right|$$

the algorithmical difference between U and V with respect to Z .

Definition 3.2.3 Let $L \subseteq \Sigma^*$, let $U = \{U_x\}_{x \in L}$ and $V = \{V_x\}_{x \in L}$ be two families of probability distributions on Σ^* .

- (1) The families U and V are called perfectly indistinguishable (p -indistinguishable) if $U = V$.
- (2) The families U and V are called statistically indistinguishable (s -indistinguishable) if for every $k \in \mathbb{N}$

$$\lim_{\substack{|x| \rightarrow \infty \\ x \in L}} |x|^k \delta_S(U_x, V_x) = 0.$$

- (3) The families U and V are called circuit indistinguishable (c -indistinguishable) if for every polynomial family $\{C_x\}_{x \in L}$ of probabilistic circuits C_x and for every $k \in \mathbb{N}$ we have

$$\lim_{\substack{|x| \rightarrow \infty \\ x \in L}} |x|^k \delta_{C_x}(U_x, V_x) = 0.$$

(4) The families U and V are called algorithmically indistinguishable (*a-indistinguishable*) if for every polynomial time probabilistic Turing machine Z and for every $k \in \mathbb{N}$ we have

$$\lim_{\substack{|x| \rightarrow \infty \\ x \in L}} |x|^k \delta_Z(U_x, V_x) = 0.$$

Lemma 3.2.1 *Let $L \subseteq \Sigma_0^*$. Let Z and Z' be homogeneous probabilistic Turing machines. Assume that Z has polynomial running time. If $\{\Pi_Z(x, \cdot)\}_{x \in L}$ and $\{\Pi_{Z'}(x, \cdot)\}_{x \in L}$ are circuit indistinguishable then for almost all $x \in L$ the elements of $Z(x)$ and $Z'(x)$ are of the same length.*

Proof: Assume that there is an infinite subset $L' \subseteq L$ such that for every $x \in L'$ the elements of $Z(x)$ and $Z'(x)$ are of different length. For $x \in L$ let C_x be the circuit with m_x input nodes, m_x being the length of the elements in $Z(x)$, which always outputs 1. Then we have

$$\delta_{C_x}(\Pi_Z(x, \cdot), \Pi_{Z'}(x, \cdot)) = 1$$

for all $x \in L'$, hence

$$\lim_{\substack{|x| \rightarrow \infty \\ x \in L}} |x| \delta_{C_x}(U_x, V_x) \neq 0.$$

For $x \in \Sigma_0^*$ we denote by $Z(x)$ the set of all elements in Σ_0^* that can with a positive probability occur as an output of Z on input of x and $Z(L) = \bigcup_{x \in L} Z(x)$.

Let Z_1 and Z_2 be probabilistic Turing machines. Then the *concatenation* $Z_1 Z_2$ of Z_1 and Z_2 is defined as the probabilistic Turing machine that first operates as Z_1 . Whenever Z_1 terminates, Z_2 is called where the input of Z_2 is the output of Z_1 . We also use the notation Z_1^n for $\underbrace{Z_1 Z_1 \dots Z_1}_{n \text{ times}}$.

Theorem 3.2.1 *Let $L \subseteq \Sigma_0^*$, $|L| = \infty$. Let $\{M_x\}_{x \in L}$, $\{O_x\}_{x \in L}$ be families of probabilistic Turing machines. Let N and N' be homogeneous probabilistic Turing machines. We define $\Gamma = \bigcup_{x \in L} M_x(x)$. Assume that the following conditions hold:*

- $|z| \geq |x|$ for all $x \in L$ and $z \in M_x(x)$.
- For $x \in L$ all elements of $M_x(x)$ are of the same length.
- $\{\Pi_N(x, \cdot)\}_{x \in \Gamma}$ and $\{\Pi_{N'}(x, \cdot)\}_{x \in \Gamma}$ are *c-indistinguishable*.
- Every O_x in $\{O_x\}_{x \in L}$ is a homogeneous polynomial time coin tossing machine.

Then $\{\Pi_{M_x N O_x}(x, \cdot)\}_{x \in L}$ and $\{\Pi_{M_x N' O_x}(x, \cdot)\}_{x \in L}$ are *c-indistinguishable*.

Proof: For $x \in L$ we set $A_x = M_x N O_x$ and $B_x = M_x N' O_x$. Let $C = \{C_x\}_{x \in L}$ be a polynomial family of circuits. Let $x \in L$ and let $\{O_{x,n}\}_{n \in \mathbb{N}}$ be a polynomial family of circuits simulating the probabilistic Turing machine O_x (see Lemma ??). Let $\bar{N}(u) = N(u) \cup N'(u)$.

Let $I \in \mathbb{N}$ such that for every $u \in \Gamma$, $|u| > I$ the elements of $N(u)$ and $N'(u)$ are of the same length. According to Lemma 3.2.1 such an I exists. Let $x \in L$, $|x| > I$, the elements of $A_x(x)$ and $B_x(x)$ are of the same length, say m_x . To measure a non zero circuit difference between $\Pi_{A_x}(x, \cdot)$ and $\Pi_{B_x}(x, \cdot)$ a circuit C must have m_x input nodes. Let $\{C_x\}_{x \in L}$ be a polynomial family of circuits. We assume that C_x has exactly m_x input nodes.

For $u \in M_x(x)$ all elements in $\bar{N}(u)$ have the same length $\ell(u)$. Let $O'_{x,u} = O_{x,\ell(u)} C_x$ be the composition of $O_{x,\ell(u)}$ and C_x . Now we have:

$$\begin{aligned}
& \delta_{C_x}(\Pi_{A_x}(x, \cdot), \Pi_{B_x}(x, \cdot)) \\
&= \left| \sum_{y \in \Sigma^{m_x}} \Pi_{C_x}(y, 1) (\Pi_{M_x N O_x}(x, y) - \Pi_{M_x N' O_x}(x, y)) \right| \\
&= \left| \sum_{y \in \Sigma^{m_x}} \Pi_{C_x}(y, 1) \sum_{u \in M_x(x)} \sum_{v \in \bar{N}(u)} \Pi_{M_x}(x, u) (\Pi_N(u, v) - \Pi_{N'}(u, v)) \Pi_{O_x}(v, y) \right| \\
&= \left| \sum_{u \in M_x(x)} \Pi_{M_x}(x, u) \sum_{v \in \bar{N}(u)} \sum_{y \in \Sigma^{m_x}} \Pi_{O_x}(v, y) \Pi_{C_x}(y, 1) (\Pi_N(u, v) - \Pi_{N'}(u, v)) \right| \\
&= \left| \sum_{u \in M_x(x)} \Pi_{M_x}(x, u) \sum_{v \in \bar{N}(u)} \Pi_{O'_{x,u}}(v, 1) (\Pi_N(u, v) - \Pi_{N'}(u, v)) \right| \\
&\leq \sum_{u \in M_x(x)} \Pi_{M_x}(x, u) \delta_{O'_{x,u}}(\Pi_N(u, \cdot), \Pi_{N'}(u, \cdot)).
\end{aligned}$$

Let $u \in \Sigma_0^*$ and let $x' \in L$ such that $u \in M_{x'}(x')$ then $|x'| \leq |u|$. Hence there are only finitely many $x' \in L$ with $u \in M_{x'}(x')$. Among the finitely many circuits $O'_{x',u}$, $u \in M_{x'}(x')$, we denote by O'_u the circuit which maximizes $\delta_{O'_{x',u}}(\Pi_N(u, \cdot), \Pi_{N'}(u, \cdot))$. Then we have

$$\delta_{C_x}(\Pi_{A_x}(x, \cdot), \Pi_{B_x}(x, \cdot)) \leq \max_{u \in M_x(x)} \delta_{O'_u}(\Pi_N(u, \cdot), \Pi_{N'}(u, \cdot)).$$

This implies that

$$\begin{aligned}
& \lim_{\substack{|x| \rightarrow \infty \\ x \in L}} |x|^k \delta_{C_x}(\Pi_{A_x}(x, \cdot), \Pi_{B_x}(x, \cdot)) \\
&\leq \lim_{\substack{|x| \rightarrow \infty \\ x \in L}} \max_{u \in M_x(x)} |u|^k \delta_{O'_u}(\Pi_N(u, \cdot), \Pi_{N'}(u, \cdot)) \\
&= 0.
\end{aligned}$$

Let $L \subseteq \Sigma_0^*$ be a language and let $\{n_x\}_{x \in L}$ be a sequence in \mathbb{N} . That sequence is called *polynomially bounded* if there is $d > 0$ such that for every $x \in L$ we have $n_x \leq |x|^d$.

Theorem 3.2.2 *Let $L \subseteq \Sigma_0^*$, $|L| = \infty$. Let $\{n_x\}_{x \in L}$ be a polynomially bounded sequence in \mathbb{N} . Let S and T be homogeneous probabilistic Turing machines. Assume that the following conditions hold:*

- $|y| \geq |x|$ for all $x \in L$ and $y \in S(x)$.
- $S(L) \subseteq L$.
- T is a coin tossing machine such that T^i , $1 \leq i \leq n_x$, has polynomial running time.
- $\{\Pi_S(x, \cdot)\}_{x \in L}$ and $\{\Pi_T(x, \cdot)\}_{x \in L}$ are c -indistinguishable.

Then $\{\Pi_{S^{n_x}}(x, \cdot)\}_{x \in L}$ and $\{\Pi_{T^{n_x}}(x, \cdot)\}_{x \in L}$ are c -indistinguishable.

Proof: We have

$$\delta_{C_x}(\Pi_{S^{n_x}}(x, \cdot), \Pi_{T^{n_x}}(x, \cdot)) = \left| \sum_{y \in \Sigma^{m_x}} \Pi_{C_x}(y, 1) (\Pi_{S^{n_x}}(x, y) - \Pi_{T^{n_x}}(x, y)) \right|.$$

We can write

$$\Pi_{S^{n_x}}(x, y) - \Pi_{T^{n_x}}(x, y) = \sum_{i=0}^{n_x-1} \Pi_{S^{n_x-i}T^i}(x, y) - \Pi_{S^{n_x-i-1}T^{i+1}}(x, y)$$

and therefore

$$\begin{aligned} & \delta_{C_x}(\Pi_{S^{n_x}}(x, \cdot), \Pi_{T^{n_x}}(x, \cdot)) \\ &= \left| \sum_{y \in \Sigma^{m_x}} \Pi_{C_x}(y, 1) \left(\sum_{i=0}^{n_x-1} \Pi_{S^{n_x-i}T^i}(x, y) - \Pi_{S^{n_x-i-1}T^{i+1}}(x, y) \right) \right| \\ &\leq \left| \sum_{i=0}^{n_x-1} \sum_{y \in \Sigma^{m_x}} \Pi_{C_x}(y, 1) (\Pi_{S^{n_x-i}T^i}(x, y) - \Pi_{S^{n_x-i-1}T^{i+1}}(x, y)) \right| \\ &\leq \sum_{i=0}^{n_x-1} \left| \sum_{y \in \Sigma^{m_x}} \Pi_{C_x}(y, 1) (\Pi_{S^{n_x-i}T^i}(x, y) - \Pi_{S^{n_x-i-1}T^{i+1}}(x, y)) \right| \\ &\leq n_x \max_{0 \leq i \leq n_x-1} \delta_{C_x}(\Pi_{S^{n_x-i}T^i}(x, \cdot), \Pi_{S^{n_x-i-1}T^{i+1}}(x, \cdot)), \end{aligned}$$

i.e. for $x \in L$ there is $0 \leq i_x \leq n_x - 1$ with

$$\delta_{C_x}(\Pi_{S^{n_x}}(x, \cdot), \Pi_{T^{n_x}}(x, \cdot)) \leq n_x \delta_{C_x}(\Pi_{S^{n_x-i_x}T^{i_x}}(x, \cdot), \Pi_{S^{n_x-i_x-1}T^{i_x+1}}(x, \cdot)). \quad (3.1)$$

Let $M_x = S^{n_x-i_x-1}T^{i_x}$, $N = S$, $N' = T$ and $O_x = T^{i_x}$. We know:

- M_x being a concatenation of homogeneous polynomial time Turing machines is also homogeneous. Since we have $|y| \geq |x|$ for $y \in S(x)$, we find $|y| \geq |x|$ for all $y \in M_x(x)$.

- The families $\{\Pi_N(x, \cdot)\}_{x \in L}$ and $\{\Pi_{N'}(x, \cdot)\}_{x \in L}$ are c -indistinguishable. The set $\Gamma = \bigcup_{x \in L} M_x(x)$ is a subset of L and therefore the families $\{\Pi_N(x, \cdot)\}_{x \in \Gamma}$ and $\{\Pi_{N'}(x, \cdot)\}_{x \in \Gamma}$ are c -indistinguishable, too.
- O_x is a polynomial time probabilistic Turing machine and being a concatenation of homogeneous Turing machines, O_x itself is a homogeneous Turing machine.

Therefore we can apply Theorem 3.2.1.

Using (3.1) for every $k \in \mathbb{N}$ we have

$$\begin{aligned}
& \lim_{\substack{|x| \rightarrow \infty \\ x \in L}} |x|^k \delta_{C_x} (\Pi_{S^{n_x}}(x, \cdot), \Pi_{T^{n_x}}(x, \cdot)) \\
& \leq \lim_{\substack{|x| \rightarrow \infty \\ x \in L}} n_x |x|^k \delta_{C_x} (\Pi_{M_x N O_x}(x, \cdot), \Pi_{M_x N' O_x}(x, \cdot)) \\
& \quad (\{n_x\}_{x \in L} \text{ is polynomially bounded. Therefore there is } d \in \mathbb{N} \text{ with } n_x < |x|^d) \\
& \leq \lim_{\substack{|x| \rightarrow \infty \\ x \in L}} |x|^{d+k} \delta_{C_x} (\Pi_{M_x N O_x}(x, \cdot), \Pi_{M_x N' O_x}(x, \cdot)) \\
& = 0.
\end{aligned}$$

Thus $\{\Pi_{S^{n_x}}(x, \cdot)\}_{x \in L}$ and $\{\Pi_{T^{n_x}}(x, \cdot)\}_{x \in L}$ are c -indistinguishable. ■

Theorem 3.2.3 *Let $L \subseteq \Sigma_0^*$, $|L| = \infty$. Let $\{n_x\}_{x \in L}$ be a sequence in \mathbb{N} . Let S and T be probabilistic Turing machines. Assume that $S(L) \subseteq L$. If $\{\Pi_S(x, \cdot)\}_{x \in L}$ and $\{\Pi_T(x, \cdot)\}_{x \in L}$ are p -indistinguishable, then also $\{\Pi_{S^{n_x}}(x, \cdot)\}_{x \in L}$ and $\{\Pi_{T^{n_x}}(x, \cdot)\}_{x \in L}$.*

Proof:

$$\begin{aligned}
& \Pi_{S^{n_x}}(x, y) - \Pi_{T^{n_x}}(x, y) \\
& = \sum_{i=0}^{n_x-1} \Pi_{S^{n_x-i} T^i}(x, y) - \Pi_{S^{n_x-i-1} T^{i+1}}(x, y) \\
& = \sum_{i=0}^{n_x-1} \sum_{u \in \Sigma_0^*} \sum_{v \in \Sigma_0^*} \Pi_{S^{n_x-i-1}}(x, u) \underbrace{(\Pi_S(u, v) - \Pi_T(u, v))}_{=0} \Pi_{T^i}(v, y) \\
& = 0
\end{aligned}$$

Theorem 3.2.4 *Let $L \subseteq \Sigma_0^*$, $|L| = \infty$. Let $\{n_x\}_{x \in L}$ be a polynomially bounded sequence in \mathbb{N} . Let S and T be probabilistic Turing machines. Assume that the following conditions hold:*

- $|y| \geq |x|$ for all $x \in L$ and all $y \in S(x)$.
- $S(L) \subseteq L$.

If $\{\Pi_S(x, \cdot)\}_{x \in L}$ and $\{\Pi_T(x, \cdot)\}_{x \in L}$ are s -indistinguishable, then also $\{\Pi_{S^{n_x}}(x, \cdot)\}_{x \in L}$ and $\{\Pi_{T^{n_x}}(x, \cdot)\}_{x \in L}$.

Proof: As in the proof of Theorem 3.2.2 we obtain

$$\delta_S(\Pi_{S^{n_x}}(x, \cdot), \Pi_{T^{n_x}}(x, \cdot)) \leq n_x \delta_S(\Pi_{M_x N O_x}(x, \cdot), \Pi_{M_x N' O_x}(x, \cdot)),$$

where $M_x = S^{n_x - i_x - 1}$, $N = S$, $N' = T$ and $O_x = T^{i_x}$. The families $\{\Pi_N(x, \cdot)\}_{x \in L}$ and $\{\Pi_{N'}(x, \cdot)\}_{x \in L}$ are s -indistinguishable.

$$\begin{aligned} & \delta_S(\Pi_{M_x N O_x}(x, \cdot), \Pi_{M_x N' O_x}(x, \cdot)) \\ &= \sum_{y \in \Sigma_0^*} |\Pi_{M_x N O_x}(x, y) - \Pi_{M_x N' O_x}(x, y)| \\ &\leq \sum_{y \in \Sigma_0^*} \sum_{u \in \Sigma_0^*} \sum_{v \in \Sigma_0^*} |\Pi_{M_x}(x, u) (\Pi_N(u, v) - \Pi_{N'}(u, v)) \Pi_{O_x}(v, y)| \\ &\leq \sum_{u \in \Sigma_0^*} \Pi_{M_x}(x, u) \sum_{v \in \Sigma_0^*} |(\Pi_N(u, v) - \Pi_{N'}(u, v))| \underbrace{\sum_{y \in \Sigma_0^*} \Pi_{O_x}(v, y)}_{\leq 1} \\ &\leq \underbrace{\sum_{u \in \Sigma_0^*} \Pi_{M_x}(x, u)}_{\leq 1} \sup_{u \in M_x(x)} \sum_{v \in \Sigma_0^*} |\Pi_N(u, v) - \Pi_{N'}(u, v)|. \end{aligned}$$

Then we have for all $k \in \mathbb{N}$

$$\begin{aligned} & \lim_{\substack{|x| \rightarrow \infty \\ x \in L}} |x|^k \delta_S(\Pi_{S^{n_x}}(x, \cdot), \Pi_{T^{n_x}}(x, \cdot)) \\ &\leq \lim_{\substack{|x| \rightarrow \infty \\ x \in L}} n_x |x|^k \sup_{u \in M_x(x)} \delta_S(\Pi_N(u, \cdot), \Pi_{N'}(u, \cdot)) \\ &\leq \lim_{\substack{|x| \rightarrow \infty \\ x \in L}} |x|^{k+d} \sup_{u \in M_x(x)} \delta_S(\Pi_N(w_x, \cdot), \Pi_{N'}(u, \cdot)) \\ &\quad (\{n_x\}_{x \in L} \text{ is polynomially bounded. Therefore there is } d \in \mathbb{N} \text{ with } n_x < |x|^d) \\ &= 0 \end{aligned}$$

Lemma 3.2.2 *Let $L \subseteq \Sigma^*$ be a language and let $L' \subseteq \Sigma^* \times \Sigma^*$ be decidable by a polynomial deterministic Turing machine. Assume that M_1 is a probabilistic Turing machine that on input of x outputs an element of $\{x\} \times \Sigma^*$ and let M_2 be the probabilistic Turing machine that is obtained from M_1 as follows: Whenever M_1 on input of x outputs a $z \in L'$ then M_2 starts M_1 again with input x .*

Then

$$\{\Pi_{M_1}(x, \cdot)\}_{x \in L} \text{ and } \{\Pi_{M_2}(x, \cdot)\}_{x \in L}$$

are statistically indistinguishable, if

$$\lim_{\substack{|x| \rightarrow \infty \\ x \in L}} \left(\sum_{z \in L'} \Pi_{M_1}(x, z) \right) |x|^k = 0$$

for every $k \in \mathbb{Z}_{>0}$.

Proof: Let $x \in L$ and let $Z_x = L' \cap (\{x\} \times \Sigma^*)$, $z \in \Sigma^* - Z_x$. The probability for M_2 to output z is

$$\sum_{k=0}^{\infty} p(x)^k \Pi_{M_1}(x, z) = \Pi_{M_1}(x, z) \frac{1}{1 - p(x)}$$

where

$$p(x) = \sum_{z' \in Z_x} \Pi_{M_1}(x, z').$$

Hence we have

$$\begin{aligned} \delta_S(\Pi_{M_1}(x, \cdot), \Pi_{M_2}(x, \cdot)) &= \sum_{z \in \Sigma^*} |\Pi_{M_1}(x, z) - \Pi_{M_2}(x, z)| \\ &= \frac{p(x)}{1 - p(x)} \sum_{z \in \Sigma^* - Z_x} |\Pi_{M_1}(x, z)| \\ &= p(x) \end{aligned}$$

which implies the assertion. ■

Lemma 3.2.3 *Let $L \subseteq \Sigma^*$ be a language. Let M be a probabilistic Turing machine. Let $\{n_x\}_{x \in L}$ be a sequence in \mathbb{N} . Assume that the machine M on input of $x \in L$ computes, among other things, a number $n \in \{1, \dots, n_x\}$. Modify M as follows. The computation of M consists of drawing (uniformly and independently) a random number $n' \in \{1, \dots, n_x\}$ and then executing M . Only if $n' = n$ the computation terminates with the output of M . Otherwise the computation starts again. The new machine is M' . Then*

$$\{\Pi_M(x, \cdot)\}_{x \in L} \text{ and } \{\Pi_{M'}(x, \cdot)\}_{x \in L}$$

are identical.

Proof: Let $L \subseteq \Sigma^*$. Then the probability for M' to output $z = n'z_1$ after exactly λ iterations is

$$\left(1 - \frac{1}{n_x}\right)^{\lambda-1} \frac{1}{n_x} \Pi_M(x, z)$$

hence the probability for M' to output z is

$$\frac{1}{n_x} \Pi_M(x, z) \sum_{\lambda=1}^{\infty} \left(1 - \frac{1}{n_x}\right)^{\lambda-1} = \Pi_M(x, z).$$

Lemma 3.2.4 *Suppose we are in the situation as above. If we define M'' as M' with the only exception that M'' terminates after at most n_x^2 steps with $n_x \geq |x|$. If M'' terminates without having guessed the correct n' then the output is ERROR. Then*

$$\{\Pi_M(x, \cdot)\}_{x \in L} \text{ and } \{\Pi_{M''}(x, \cdot)\}_{x \in L}$$

are statistically indistinguishable.

Proof: We have

$$\begin{aligned}
& \delta_S(\Pi_M(x, \cdot), \Pi_{M''}(x, \cdot)) \\
&= \sum_{z \neq \text{ERROR}} \left| \Pi_M(x, z) \left(1 - \frac{1}{n_x} \sum_{\lambda=1}^{n_x^2} \left(1 - \frac{1}{n_x} \right)^{\lambda-1} \right) \right| + \left(1 - \frac{1}{n_x} \right)^{n_x^2} \\
&= \sum_{z \neq \text{ERROR}} \left| \Pi_M(x, z) \left(1 - \frac{1}{n_x} \frac{(1 - \frac{1}{n_x})^{n_x^2} - 1}{(1 - \frac{1}{n_x}) - 1} \right) \right| + \left(1 - \frac{1}{n_x} \right)^{n_x^2} \\
&\leq 2 \left(1 - \frac{1}{n_x} \right)^{n_x^2} \\
&\leq \left(\frac{1}{2} \right)^{n_x - 1}.
\end{aligned}$$

This clearly implies that for $k > 0$

$$\lim_{\substack{x \rightarrow \infty \\ x \in L}} |x|^k \delta(\Pi_M(x, \cdot), \Pi_{M''}(x, \cdot)) = 0.$$

■

Lemma 3.2.5 *In the situation of Lemma 3.2.3 the expected number of iterations carried out by the machine M' is n_x .*

Proof: The probability for M' to terminate after exactly λ iterations is $\frac{1}{n_x} \left(1 - \frac{1}{n_x} \right)^{\lambda-1}$. Hence, with $y = \left(1 - \frac{1}{n_x} \right)$, the expected number of iterations is

$$\begin{aligned}
\frac{1}{n_x} \sum_{\lambda=1}^{\infty} \lambda y^{\lambda-1} &= \frac{1}{n_x} \frac{d}{dy} \sum_{\lambda=0}^{\infty} y^{\lambda} \\
&= \frac{1}{n_x} \frac{1}{(1-y)^2} \\
&= n_x.
\end{aligned}$$

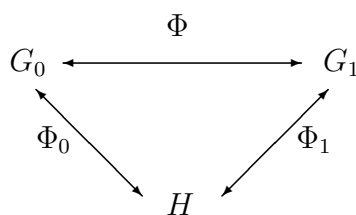
■

Chapter 4

Zero knowledge proofs

4.1 The idea

In standard identification schemes the prover P who wants to prove his identity uses a secret password which is also known to the verifier V . The problem is that anybody who listens to the identification protocol can use the password to pretend that he is P . Zero knowledge techniques enable the prover to prove his knowledge of an identifying secret in a way such that nobody listening to the interactive proof can learn anything about the secret that he could not find out for himself without knowing the proof. Let us look at an example. The secret known to the prover is the isomorphism Φ between two graphs G_0 and G_1 . The graphs G_0 and G_1 are also known to the verifier but the isomorphism is not. The prover wants to demonstrate his knowledge of Φ without revealing any information about Φ . He chooses a random isomorphism Φ_0 of G_0 and computes the isomorphic graph $H = \Phi_0(G_0)$. Since P knows the isomorphism Φ he can compute an isomorphism Φ_1 between H and G_1 . So P knows the following:



The prover sends the graphs G_0, G_1, H to the verifier but he keeps the isomorphism Φ , Φ_0, Φ_1 secret. The verifier tosses a coin and depending on whether he obtains 0 or 1 he asks either for Φ_0 or for Φ_1 . The prover then reveals the requested information and the verifier checks whether the map he obtained is in fact the isomorphism he asked for. If he discovers a mistake he will reject the proof of identity. Clearly, an honest prover will not be rejected. On the other hand, a cheating prover P' who does not know the isomorphism Φ between G_0 and G_1 will either be unable to determine Φ_0 or to find Φ_1 because the composition of those isomorphisms yields the isomorphism Φ . So P' must guess in advance which of the isomorphisms Φ_0 or Φ_1 the verifier will be asking for. He will fail to guess correctly with probability $\frac{1}{2}$. Therefore, if this protocol is repeated n times, then a cheating prover will be detected by the verifier with probability $1 - (\frac{1}{2})^n$. Finally, anybody who is listening to the conversation only obtains a sequence of randomly chosen graphs which are isomorphic either to G_0 or G_1 and such a sequence he could produce easily himself. So he learns nothing. The protocol we have described here informally is a zero knowledge proof system for the language of pairs of isomorphic graphs. We will in this chapter define formally such proof systems and we will characterize the class of languages which allow a zero knowledge proof.

4.2 Zero knowledge proof systems

To formalize the idea described in the previous section, we need the following notation: For an interactive system $S = (P, V)$ we let \bar{S} be the probabilistic Turing machine that exactly operates as S except that it outputs the final content of the receive tape of the verifier and the random string used by V rather than an accepting 1 or a rejecting 0.

Definition 4.2.1 *Let $L \in \Sigma_0^*$ be an interactive proof system for L . We say that S has the statistical or circuit or algorithmical zero knowledge property if for any polynomial time interactive Turing machine V^* there is a coin tossing machine M_{V^*} (simulator) having expected polynomial running time such that for $S^* = (P, V^*)$ the families $\{\Pi_{\bar{S}^*}(x, \cdot)\}_{x \in L}$ and $\{\Pi_{M_{V^*}}(x, \cdot)\}_{x \in L}$ are statistically or circuit or algorithmically indistinguishable. If those families are identical then S is said to have the perfect zero knowledge property. If S has one of the zero knowledge properties the system is called a perfect or statistically secure or circuit secure or algorithmically secure zero knowledge proof system for L .*

Substituting V by V^* means that the verifier V tries to be clever by changing his program. But no matter how clever he is all the information he obtains can also be found by applying the expected polynomial time simulator M_{V^*} without ever using any output of the powerful prover P . So V learns nothing from the interaction with P .

Example 4.2.1 We formalize the example given in the introductory section. Two graphs $G_0 = (V, E_0)$ and $G_1 = (V, E_1)$ are called isomorphic if there is a bijection $\Phi : V \rightarrow V$ such that $E_1 = \{(\Phi(u), \Phi(v)) : (u, v) \in E_0\}$. We write $G_0 \cong G_1$ and $G_1 = \Phi[G_0]$. Consider the language L of all pairs (G_0, G_1) of isomorphic graphs G_0 and G_1 . We present a zero knowledge proof system $S = (P, V)$ for that language. The input is a pair $x = (G_0, G_1)$ of graphs $G_0 = (M, E_0)$ and $G_1 = (M, E_1)$ with identical sets of vertices M . The system carries out the following steps:

0. V sets its counter i to 1.
1. P computes the isomorphism Φ between G_0 and G_1 .
2. P randomly picks a permutation $\Phi_0 : M \rightarrow M$ and computes $H = \Phi_0[G_0]$ as well as the isomorphism Φ_1 between G_1 and H .
3. P sends H to the verifier V .
4. V chooses at random $r \in \{0, 1\}$ and sends r to P .
5. P sends Φ_r to V .
6. V checks whether $H = \Phi_r[G_r]$. If this is not true then V rejects the input. Otherwise V increases the counter i by 1. If $i > |x|$ then V accepts the input. Otherwise the system starts again with step 2.

We must show that the interactive system satisfies the completeness condition and the correctness condition and has the ZK-property. The completeness condition is trivial. As to the correctness condition, if G_1 and G_0 are not isomorphic then the probability for the system to make it to the $|x|$ -th round is $1 - (\frac{1}{2})^{|x|}$ which is less than $\frac{1}{3}$. Now we discuss the ZK-property. For this purpose we must extend the definition of the prover P a little bit since we allow any polynomial time verifier V^* to participate in the communication. If in step 4 the prover receives something different from 0 or 1 he starts sending a string of 0's until the computing time of the verifier is used up. Let V^* be an interactive turing machine whose running time is bounded by $|x|^c$ with some $c \in \mathbb{N}$. Then the final content of the receive tape of V^* after a computation of the system $S^* = (P, V^*)$ on input of $x = (G_0, G_1) \in L, G_i = (V, E_i), i = 1, 2$ is of the form

$$y = H_0 \square \Phi_0 \square H_1 \square \Phi_1 \square \dots \square H_k \square \Phi_k \square$$

or

$$y = H_0 \square \Phi_0 \square H_1 \square \Phi_1 \square \dots \square H_k \square s$$

where $k \leq |x|$, where H_i, Φ_i are encodings of graphs and isomorphisms and where $s \in \{0\}^*$, $|s| \leq |x|^c$. In those strings, each graph H_i and each permutation Φ_i occurs with the same probability, independent of whether P receives 0 or 1. The occurrence of a string of zeros depends only on the behavior of V^* . Therefore we can easily construct a

polynomial time simulator M_{V^*} which produces exactly the same output distribution as $\overline{S^*}$. M_{V^*} works as follows:

At first M_{V^*} simulates the behavior of the prover P by choosing at random $\alpha \in \{0, 1\}$ and a permutation $\Phi_0 : M \rightarrow M$ and calculates $H = \Phi_0[G_\alpha]$. Then M_{V^*} simulates the verifier by choosing at random $r \in \{0, 1\}$. If $\alpha = r$ then M_{V^*} outputs $H \square \Phi_0$. Otherwise M_{V^*} starts again.

So we have shown that the system is a perfect zero knowledge proof system for the language of pairs of isomorphic graphs.

4.3 Secure encryption

The goal in this chapter is to prove that every language in NP has an algorithmically secure zero knowledge proof system. This statement is only known to be valid under the assumption that there exist secure encryption functions. This is why we need to discuss secure encryption in this section. Whenever we use \mathbb{N} as the index set of a family we identify \mathbb{N} with the language $\{I^n : n \in \mathbb{N}\}$.

Definition 4.3.1 *Let $L \subseteq \Sigma^*$ be a language. A polynomial time deterministic Turing machine M is called an encryption machine or an encryption algorithm for L if it satisfies the following properties*

- (1) *On input of $x \in L$ (the string to be encrypted) and $\alpha \in \{0, 1\}^*$ (the key) it terminates and outputs a string $M(x, \alpha) \in \Sigma^*$.*
- (2) *If for $x, y \in L$ and $\alpha, \beta \in \{0, 1\}^*$ the output strings are identical, i.e.*

$$M(x, \alpha) = M(y, \beta)$$

then the encrypted strings x and y must be equal.

The security of an encryption algorithm M depends on the probability with which each string $z \in \Sigma^*$ occurs as an encryption of $x \in \Sigma^*$. If we only consider keys of a fixed length n then this probability is

$$\Pi_{M,n}(x, z) = \frac{1}{2^n} |\{\alpha \in \{0, 1\}^n : M(x, \alpha) = z\}|.$$

Definition 4.3.2 *Let $L \subseteq \Sigma^*$. An encryption algorithm M for L is called circuit or algorithmically secure if for any two strings $x, y \in L$ the families*

$$\{\Pi_{M,n}(x, \cdot)\}_{n \in \mathbb{N}} \text{ and } \{\Pi_{M,n}(y, \cdot)\}_{n \in \mathbb{N}}$$

are circuit or algorithmically indistinguishable.

It can be shown that there cannot exist encryption algorithms which are “identical” or “statistical” secure.

We now show that one can obtain secure encryption functions for strings from secure encryption functions for characters. If M is an encryption function for Σ , then we can encrypt a string $x = x_1x_2 \dots x_k$ by encrypting the single characters separately. The probability for a string $z = z_1z_2 \dots z_k \in \Sigma^*$ to occur as encryption is

$$\Pi_{M,n}(x, z) = \prod_{i=1}^k \Pi_{M,n}(x_i, z_i).$$

4.4 $NP \subseteq ZK$

In this section we show that under the assumption of the existence of circuit secure encryption functions every language in NP has an circuit secure zero knowledge proof. This is done by presenting a zero knowledge proof system for the language $3C$ of three-colourable graphs. Then the fact is used that this language is NP -complete.

Definition 4.4.1 *A directed graph $G = (V, E)$ is called three colourable if there is a map $\Phi : V \rightarrow \{1, 2, 3\}$ such that for every $(v, w) \in E$ we have $\Phi(v) \neq \Phi(w)$.*

We will now present an interactive system S and we will show that under the assumption of the existence of circuit secure encryption functions the system S is an circuit secure zero knowledge proof system.

Without loss of generality we can restrict our attention to connected graphs, since a graph is three colourable if and only if its connected components are three colourable and because the connected components of a graph can be computed in polynomial time. Graphs are encoded over the alphabet $\Sigma_0 = \{0, 1, c\}$. If $G = (V, E)$ is a graph, $V = \{1, 2, \dots, k\}$ then its encoding is

$$\rho(G) = R_1R_2 \dots R_k$$

with

$$R_i = cr_{i1}cr_{i2}c \dots cr_{ik}c$$

where

$$r_{ij} = \begin{cases} 0 & \text{if } (i, j) \notin E \\ 1 & \text{if } (i, j) \in E. \end{cases}$$

Then we have

$$|\rho(G)| = 2k(k+1)$$

and since G is connected and simple we have for $k \geq 2$

$$\frac{k}{2} \leq |E| \leq \frac{k(k-1)}{2}.$$

This implies that

$$4|E| \leq |\rho(G)| \leq 10|E|^2. \quad (4.1)$$

for $k \geq 2$. The idea of the protocol is the following:

The prover commits himself to a three colouring by sending such a map in encrypted form to the verifier. The verifier checks that three colouring by choosing at random an edge and by asking the prover to present the colours of the corresponding vertices. He checks whether those colours are distinct and since he knows the encryption function he can check whether the colours he has obtained are the same that were used in the provers original three colouring that he knows in encrypted form. In order for the verifier not to be able to obtain partial information about a three colouring of the graph, the prover changes his three colouring in each interaction by applying a random permutation to the colours.

Here is the formal description of the interactive system:

Let M be a circuit secure encryption function for Σ^* .

Protocol 4.4.1

Input: $x \in \Sigma^*$.

Output: 0 or 1.

0. *The verifier checks whether x is a correct encoding of a graph. If not he rejects, i.e. the system outputs 0. The prover checks whether x is a three colourable graph. If this is wrong, he sends 0 to the verifier and the verifier rejects, i.e. the system outputs 0. If $x = \rho(G) = \rho((V, E))$ is a three colourable graph then the prover finds a three colouring Φ . Both prover and verifier set $i = 1$.*
 - * *In the sequel we put $k = |V|, g = |\rho(G)|, E = (e_1, \dots, e_\ell)$ and assume that the edges are ordered lexicographically. That means both prover and verifier have the same numbering on the edges. We also denote by i the number of the interaction which P and V are currently carrying out.**
1. *If $i > \ell^2$ then the system outputs 1.*
2. *The prover uses the machine PED with input $(6, g)$ to draw a random permutation π from S_3 . Then $\Phi' = \pi \circ \Phi$ is a new three colouring of G .*
3. *The prover generates k random strings $r_j \in \{0, 1\}^g$ ($1 \leq j \leq k$). Then he computes $\varphi_j = M(\Phi'(j), r_j)$ ($1 \leq j \leq k$). He sends the sequence $(\varphi_1, \dots, \varphi_k)$ to the verifier.*
4. *Using PED with input (ℓ, ℓ) the verifier draws a random edge $e = (u, v)$ and sends it to P .*
5. *The prover sends the pairs $p_s = (\Phi'(s), r_s), s \in \{u, v\}$ to the verifier.*

6. Now the verifier has two pairs $p_s = (x_s, r_s), s \in \{u, v\}$ and checks whether

$$\begin{aligned}\varphi_s &= M(p_s), \\ s &\in \{u, v\}, \\ x_u &\neq x_v \text{ and} \\ x_u, x_v &\in \{1, 2, 3\}.\end{aligned}$$

If one of the conditions is violated then the verifier rejects the input. Otherwise the system sets $i = i+1$, goes back to step 1 and starts the next interaction.

Theorem 4.4.1 *Protocol 4.4.1 is an interactive proof system for the language $3C$.*

Proof:

Completeness:

If the input x is the encoding of a three colourable graph then the prover is able to calculate a three colouring and therefore he will pass all the tests of the verifier. The system accepts.

Correctness:

If the input x is not the encoding of a graph then the verifier rejects that input in the very first step.

Now suppose that the input x is a graph $G = (V, E)$ which is not three colourable, i.e. for every map $\Phi : V \rightarrow \{1, 2, 3\}$ there is an edge $e = (u, v)$ with $\Phi(u) = \Phi(v)$.

If the verifier does not reject the input x in step 6 then, because of the partial injectivity of the encryption function it follows that the colours chosen for the vertices u and v by the prover must have been different. But there is at least one edge whose $e = (u, v)$ where u and v have the same colour. According to Example 1.3.2 that edge is chosen by the verifier at least with probability

$$\frac{1}{\ell} \left(1 - \frac{1}{2^\ell}\right) \geq \frac{1}{2\ell}.$$

The probability for a cheating prover to pass this test ℓ^2 times is, therefore, at most $(1 - \frac{1}{2\ell})^{\ell^2}$. Now using the inequality

$$\left(1 - \frac{1}{n}\right)^n \leq \frac{1}{2}.$$

we find that for $\ell \geq 4$ the probability for a cheating prover to pass the test in step 6 ℓ^2 times is at most $\frac{1}{3}$ since

$$\left(1 - \frac{1}{2\ell}\right)^{\ell^2} = \left(\left(1 - \frac{1}{2\ell}\right)^{2\ell}\right)^{\ell/2} \leq \left(\frac{1}{2}\right)^{\ell/2} \leq 1/3$$

for $\ell \leq 4$. Because of (5.1) and $\ell = |E|$ it follows that the numbers of iterations are polynomial in $|x| = |\rho(G)|$. Therefore the correctness condition is satisfied. ■

Theorem 4.4.2 *If there exists a circuit secure encryption algorithm then Protocol 4.4.1 describes a circuit secure zero knowledge proof system for $3C$.*

Proof: Suppose that the encryption algorithm used in Protocol 4.4.1 is circuit secure. We let $G = (V, E)$ be the input graph and we use the notation from Protocol 4.4.1. Now let V^* be a cheating verifier, i.e. V^* is an arbitrary polynomial time interactive Turing machine. We have to describe a probabilistic Turing machine M with expected polynomial running time that simulates the interaction of $S^* = (P^*, V^*)$, i.e. the families

$$\{\Pi_{S^*}(x, \cdot)\}_{x \in 3C} \text{ and } \{\Pi_M(x, \cdot)\}_{x \in 3C}$$

are circuit indistinguishable.

We describe how the i th interaction is simulated. Let m_j be the message of the prover to the verifier in the j th interaction and let R_j be the random string used by the verifier in the j th interaction. We note that the computation of the verifier in the first $i-1$ interactions is completely determined by $\underline{m}_{i-1} = (m_1, \dots, m_{i-1})$ and by $\underline{R}_{i-1} = (R_1, \dots, R_{i-1})$. We denote by T_1 the machine that on input of \underline{R}_{i-1} and \underline{m}_{i-1} carries out the i th interaction of S^* .

The machine T_2 that simulates T_1 operates as follows:

Input: $i, G, \underline{m}_{i-1}, \underline{R}_{i-1}$.

Output: $i, G, \underline{m}_i, \underline{R}_i$ or 0.

0. If $i > \ell^2$, T_2 outputs 0.
1. The computation of V^* in the first $i-1$ interactions is reconstructed. In particular, the content of the tapes of V^* after $i-1$ interactions is generated. Then a counter λ is set to one.
2. First, k random strings $r_j \in \{0, 1\}^g$ ($1 \leq j \leq k$) are generated. Then by using PED with input (ℓ, ℓ) a random edge $e' = (u', v')$ is chosen and by using PED again with input $(6, g)$ a random pair of "colours" $(a, b) \in \{1, 2, 3\}^2$ is chosen. Then the encryptions $\varphi_{u'} = M(a, r_{u'})$, $\varphi_{v'} = M(b, r_{v'})$ and $\varphi_j = M(0, r_j)$, $1 \leq j \leq k$, $j \neq u', v'$ are computed.
3. Using the program of V^* the simulator comes up with an edge $e = (u, v)$. If $e = e'$ then the new message m_i is constructed from φ_j ($1 \leq j \leq k$), a, b, r_u, r_v ; the simulator outputs m_i and the random string R_i that was used in the simulation of V^* , and terminates its computation. Otherwise the counter λ is incremented by 1.
4. If $\lambda > 2\ell^2$ then φ_j is set to $M(0, 0^g)$, $1 \leq j \leq k$ and the new message m_i is constructed from φ_j ($1 \leq j \leq k$), $a = 1, b = 2, 0^g, 0^g$. The simulator outputs m_i and the random string R_i that was used in the simulation of V^* and terminates its computation.
5. If $\lambda \leq 2\ell^2$ then the computation starts again at step 2.

We denote by \mathcal{T} the set of all possible inputs for T_1 or T_2 that correspond to at most $\ell^2 - 1$ interactions. Clearly, the machine $\overline{S^*}$ can be obtained from applying T_1 ℓ^2 times and then erasing R_{ℓ^2} . The simulator M calls the machine T_2 ℓ^2 times and erases R_{ℓ^2} . We must show that the families

$$\{\Pi_{\overline{S^*}}(x, \cdot)\}_{x \in 3C} \text{ and } \{\Pi_M(x, \cdot)\}_{x \in 3C}$$

are circuit indistinguishable.

For technical reasons we modify T_1 as follows:

* The input and output are the same as for T_1 .

1. In a precomputation, T_1 computes the three-colouring and the content of the tapes of V^* given the history was input to T_1 . It also sets a counter λ to 1.
2. In addition to generating r_j, φ_j and π the prover also chooses a random $e' = (u', v')$.

It is sufficient to show that

$$\{\Pi_{T_1}(x, \cdot)\}_{x \in 3C} \text{ and } \{\Pi_{T_2}(x, \cdot)\}_{x \in 3C}$$

are circuit indistinguishable. By Lemma 3.2.2 it suffices to show that

$$\{\Pi_{T_1}(x, \cdot)\}_{x \in \mathcal{T}} \text{ and } \{\Pi_{T_2}(x, \cdot)\}_{x \in \mathcal{T}}$$

are circuit indistinguishable. The machine T_2 can be obtained from the machine T_1 by means of the following changes.

Rather than computing the output immediately, the machine T_1 checks whether $e = e'$. If that condition is violated the machine starts again with the same input. After a maximum of ℓ^2 attempts it terminates and outputs *ERROR* if it was not successful. The modified machine is called T'_1 . By Lemma 3.2.3 it follows that

$$\{\Pi_{T_1}(x, \cdot)\}_{x \in \mathcal{T}} \text{ and } \{\Pi_{T'_1}(x, \cdot)\}_{x \in \mathcal{T}}$$

are circuit indistinguishable. So it is sufficient to regard only the differences between T'_1 and T_2 : The way in which the encryption $\varphi_j (1 \leq j \leq k)$ of the three-colouring is found. T'_1 encrypts the real three-colouring Φ' whereas T_2 does not compute such a three-colouring. The second change is the way in which the colours a, b of the edge e' are found. The machine T'_1 uses the colours prescribed by the map Φ' and T_2 just chooses two different colours at random.

Those changes amount to replacing a subroutine S'_1 in T'_1 by a subroutine S_2 . The subroutine S'_1 takes the input of T'_1 . It computes the three-colouring, finds $\pi, e', r_j, \varphi_j = M(\Phi(j), r_j)$ ($1 \leq j \leq k$), the colours $a = \Phi'(u'), b = \Phi'(v')$ and outputs those values. The subroutine S_2 finds $\pi, e', r_j (1 \leq j \leq k), \varphi_j = M(\Phi'(j), r_j) (1 \leq j \leq k, j \neq u', v'), a =$

$\pi(1), b = \pi(2), \varphi_{u'} = M(a, r_{u'}), \varphi_{v'} = M(b, r_{v'})$ and outputs those values. The circuit indistinguishability of the two output distributions follows from the circuit security of the encryption machine M .

Since those subroutines are called at most $2\ell^2$ times the circuit indistinguishability of $\{\Pi_{T_1^{\ell^2}}(x, \cdot)\}_{x \in 3C}$ and $\{\Pi_{T_2^{\ell^2}}(x, \cdot)\}_{x \in 3C}$ follows from Theorem 3.2.1. This completes the proof. ■

Bibliography

- [1] M. GAREY AND D. S. JOHNSON, *Computers and Intractability, a Guide to the Theory of NP-Completeness*, Freeman, 1979.
- [2] HARRY R. LEWIS AND CHRISTOS H. PAPADIMITRIOU, *Elements of the theory of Computation*, Prentice-Hall, 1981.
- [3] V. PRATT, *Every prime has a succinct certificate*, SIAM J. Comput. (1975), no. 4.
- [4] K. R. REISCHUK, *Einführung in die Komplexitätstheorie*, Teubner, 1990.