Interactive proof systems[*]

The goal of these notes is to define interactive proof systems, and to show that the class of languages that have such proof system coincides with the class PSPACE. We follow the Section 10.4 of Sipser's book, but with a bit different notation.

Imagine we want to know whether a theorem is true or not. A clever person, might convince us by presenting a proof of the theorem. If a proof exists that is not too long and not too complicated, we can be convinced in a short time. If the theorem is false, then we assume that no proof can convince us. Now, assume that no short proof exists. Is it still possible to be convinced that a theorem is true? Imagine a PhD student claims to have proven a difficult theorem and wants to defend his thesis. Unfortunately, the professors in the committee have only a few days time to study the manuscript, and can impossibly learn about all the necessarily background from the references to check all details. Still, the people in the committee must figure out whether the theorem is true or not. The student and the committee will participate in a discussion. Members of the committee might ask for generalizations of some claims in the proof. Then they consider special cases of these generalizations, and then ask again about new generalizations of these special cases and so on. If the student can give logically sound answers, it is likely the committee will grant the student his PhD degree.

In computational complexity, the class NP contains all sets for which there is a way of convincing a verifier of set membership for all its elements. For languages outside NP there is no such procedure. For example, it is not known how to make short proofs that Boolean formulas are unsatisfiable (this would imply NP = coNP). The goal of this section is to define the class IP of sets for which a randomized dialogue can convince a verifier that an element belongs to the set. If the element does not belong to the set, then it should be unlikely that the verifier can be convinced by such a dialogue. We show that this class coincides with PSPACE. Hence, it is believed to be much larger than NP.

1 Example: graph non-isomorphism

To illustrate the power of randomized dialogues, consider the following story about a blind man who has two socks and to whom is told that they are of different colour, say blue and red. The blind man trusts nobody, but still there is a way that someone can convince him that his socks are different? He proceeds as follows: First, he hides both socks. Then he randomly selects one of them, shows it to a person and asks him whether it is blue or red. He repeats this experiment several times and if this person gives consistent answers, i.e., always call the same sock red and the other blue, he concludes that either his socks are different or some rare event has occurred.

A similar strategy exists to determine whether two graphs are not isomorphic. A graph $G = (V, E)$ is represented as a set V of vertices and a set $E \subseteq V \times V$ of edges. Two

[*]Version May 29, 2017

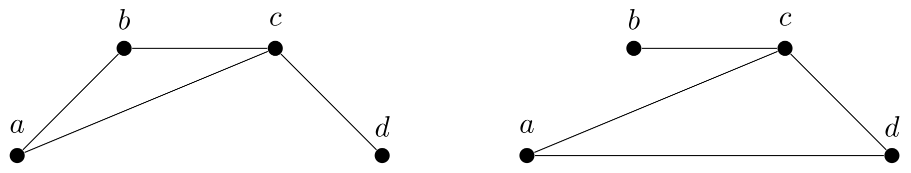

Figure 1: Two isomorphic graphs. The bijection f maps a, b, c, d to a, d, c, b.

graphs are isomorphic if they are equal after renaming the vertices, see figure **??**. More precisely, a *bijection* from V to W is a function $f : V \to W$ for which for each $w \in W$ there is exactly one $v \in V$ such that $f(v) = w$. Two graphs $G = (V, E)$ and $H = (V, F)$ are *isomorphic* if there exists a bijection $f : V \to V$ such that $(a, b) \in E \subseteq V \times V$ if and only if $(f(a), f(b)) \in F \subseteq W \times W$, see figure 1 for an example. Let

$$\mathit{ISO} = \{\langle G, H\rangle : G \text{ and } H \text{ are isomorphic graphs}\}\,.$$

ISO is in NP, because a bijection provides a certificate. Despite extensive research there is no proof that *ISO* is in P or NP-complete. Recently L. Babai proved that *ISO* can be decided in quasi polynomial time, i.e., $\mathit{ISO} \in \mathrm{TIME}(2^{\log^k n})$. Here, we consider the complement of this language, i.e., $\mathit{NONISO} = \{\langle G, H\rangle : G$ and H are *not* ismorphic graphs$\}$. It is unknown whether this language is in NP: nobody knows how to make short certificates that imply graphs are not isomorphic. However, with a dialogue a prover can convince a verifier that two graphs are not isomorphic.

Assume G_1 and G_2 have the same sets of vertices V. The verifier randomly selects either G_1 or G_2. Then he randomly reorders the vertices of the graph. The verifier sends this graph H to the prover, and the prover must identify whether H came from G_1 or G_2.

If G_1 and G_2 are not isomorphic, then the prover can figure out from which graph H was generated. Otherwise, he can not do better than guessing with 1/2 chance of success. Let us explain why this probability is 1/2 in detail. Consider the set $B_{V \to V}$ of all bijections from V into V. If h is uniformly randomly chosen in $B_{V \to V}$, then the composition $h \circ f$ is also a bijection, and moreover it is uniformly randomly distributed in $B_{V \to V}$; this is because the mapping $h \to f \circ h$ is itself a bijection on $B_{V \to V}$. Thus, if f is the bijection on V that maps G_1 to G_2, then $h(G_1)$ has the same distribution as $(h \circ f)(G_1) = h(G_2)$.

2 Definition

An *interactive proof system* is a pair (P, V) where P is called the prover and V is called the verifier. The verifier is a Turing machine with a private work tape and a tape that contains randomly generated bits. He interacts with the prover through a separate communication tape, see figure 2. On input $x \in \Sigma^*$, this interaction happens as follows: Initially, the communication tape contains x. The verifier reads the contents of the communication tape, makes a computation and writes a message $m_1 \in \Sigma^*$. When he halts, the prover reads the communication tape and writes a message m_2 on the tape. He then returns control to the verifier, who reads the tape, computes, and writes a message m_3, and so on. The prover is any deterministic device that reads and writes on the communication tape. Besides being deterministic, there are no restrictions on the messages the prover, or on the computability of these messages. For example, one can consider a prover whose messages contain the solution

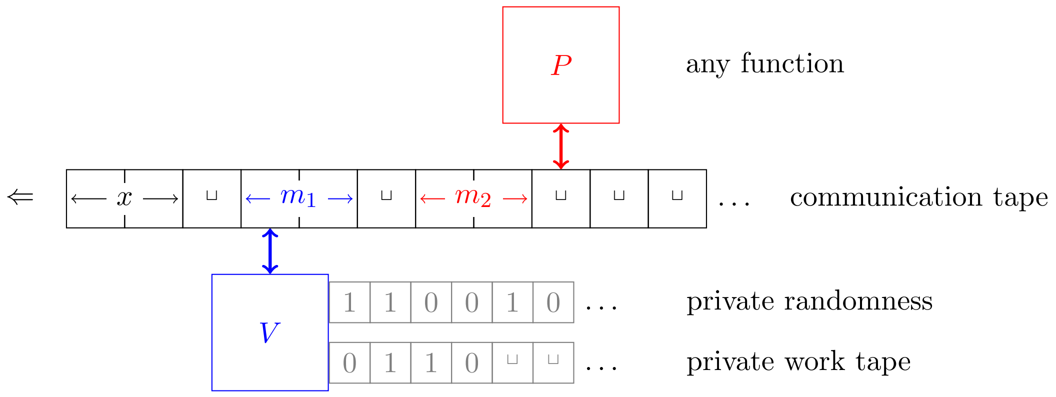

Figure 2: Definition of an interactive proof system

of the halting problem. The process terminates if the verifier arrives either in an accept or a reject state. The statement

$$(V \leftrightarrow P)(x, r) = \texttt{accept}$$

means that if the process is started with x on the communication tape, and a sequence r on the verifier's private randomness tape, then the verifier terminates in an accept state. When r is generated uniformly, the probability that a proof system accepts x is written as

$$\mathsf{Prob}\Big((V \leftrightarrow P) \text{ accepts } x\Big) = \mathsf{Prob}_r\Big((V \leftrightarrow P)(x, r) = \texttt{accept}\Big).$$

A verifier is said to run in *polynomial time* if there exists a polynomial p such that for all x, for all choices of random bits, for all provers P, in the interaction the verifier has terminated after executing at most $p(|x|)$ many computation steps.

Definition 1. *A set A is in* IP *if there exists a polynomial time verifier V and a prover P such that:*

- *If $x \in A$, then* $\mathsf{Prob}\big((V \leftrightarrow P)$ *accepts* $x\big) \geq 2/3$,
- *If $x \notin A$, then for every prover P',* $\mathsf{Prob}\Big((V \leftrightarrow P')$ *accepts* $x\Big) \leq 1/3$.

Observe that BPP $\subseteq$ IP and NP $\subseteq$ IP; for the first inclusion we can choose a proof system in which the prover does not communicate at all, and for the second inclusion we can choose a deterministic verifier that reads only one message from the prover. For the same reason as in the definition of BPP, the class IP does not change if we replace the constants 2/3 and 1/3 by any $\varepsilon_\in$ and $\varepsilon_\notin < \varepsilon_\in$. The class also does not change, if we replace these constants by $1 - 2^{-q(|x|)}$ and $2^{-q(|x|)}$ for any polynomial q.

Exercise 1. Suppose that in the definition of IP the verifier is deterministic (i.e., has no access to random bits). Show that this class equals NP.

Exercise 2. Let A be a set such that both A and its complement are in IP. Show that $P^A \subseteq$ IP. What about NP^A?

Exercise 3. Suppose we want to show that the complement of a set in IP is also in IP. Why is it not enough to convert all accepting states in the verifier to rejecting ones and vice versa? (Note that the result IP = PSPACE implies that IP is closed under complement, but no short direct proof seems to be known.)

Using our knowledge of oracles machines and probabilistic machines, a much shorter definition of the class IP can be given. We say that that a probabilistic oracle machine runs in polynomial time if for all oracles and for all random choices the machine halts in polynomial time.

Lemma 2. *A set A is in* IP *if and only if there exists a probabilistic polynomial time oracle machine U and an oracle $\tilde{P}$ such that:*

- *if $x \in A$, then* $\mathsf{Prob}\big(U^{\tilde{P}}(x) \text{ accepts}\big) \geq 2/3$,
- *if $x \notin A$, then for every oracle $\tilde{P}'$,* $\mathsf{Prob}\Big(U^{\tilde{P}'}(x) \text{ accepts}\Big) \leq 1/3$.

Proof. A prover in the definition of IP can always write the answers to queries of an oracle on the tape, so if a probabilistic oracle machine exists that satisfies the conditions above, then $A \in$ IP. For the other direction, we need to simulate an interaction with a prover by queries to an oracle. We fix a method to interpret each state of the dialogue tape as an index of an oracle such that all this addresses are at polynomial distance from each other. The first bit of a prover's reply for a given state of the dialogue tape is encoded in $\tilde{P}$ at the corresponding position. The remaining bits are coded directly after it. Because the verifier has only a polynomial time to scan the answer, we can assume that the next index that encodes a reply is far enough. It is now easy to verify that U also works in polynomial time. □

In the remainder of these notes we prove our main result.

Theorem 3. IP = PSPACE

3 IP $\subseteq$ PSPACE

In this section we transform an interactive proof system to a deterministic decider that runs in polynomial space, i.e., we prove IP $\subseteq$ PSPACE.

We start by arguing that if a language A is in IP, then A is decidable. This is not directly obvious from the definition, because there are no restrictions on the computability of the prover. Fix a verifier for A and some string x. The idea is that since the prover has unbounded computational power, he can compute all probabilities of possible behaviours of the verifier on input x. Then he can figure out an optimal strategy to send messages that makes the probability of acceptance maximal. This means that for every verifier, there exists an optimal strategy for the prover that is computable. Thus also the optimal probability of acceptance is computable. From this probability it can be decided whether x belongs to A, because by definition of IP, this probability is at least $2/3$ if and only if $x \in A$. Hence, A is decidable.

The proof proceeds by formulating this algorithm more carefully and observing it can be executed with polynomial space. What is the algorithm to compute the optimal strategy for the prover? For a fixed verifier V and input x, we represent all reactions of the verifier and all provers on a tree. Note that the reactions of the verifier depend on random bits. A node of the tree at some depth d is specified by the first d messages in the dialogue. The tree has polynomial depth, because the verifier always terminates in polynomial time. This is illustrated in figure 3 for a bitwise dialogue. Every prover defines a subtree in this tree. The subtree of one such prover is represented in red. We now represent the probability that a prover accepts x on this tree. Let t be a polynomial bound on the number of random bits used by the verifier. Consider for some leaf of the tree the set of all $t(|x|)$-bit strings r such that the messages of the verifier are consistent with the dialogue of this leaf. The probability of acceptance is proportional to the ratio of these strings that make the verifier accept.

For each node and for each $t(|x|)$-bit string r that makes the verifier accept in a state where the dialogue tape has the value determined by the node, we add to this node a leaf that we call an *accept leaf.* The probability that a verifier accepts for some prover P is $2^{-t(|x|)}$ times the number of accept leaves in the subtree that corresponds to this prover. In the example of the figure, there are 6 accepting leaves for the strategy in red. In fact, the strategy in red is the optimal strategy.

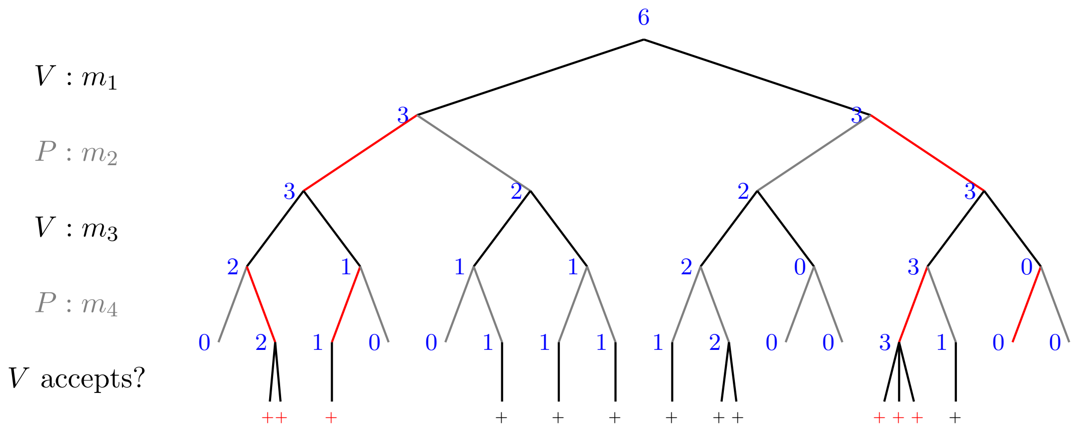

Figure 3: Tree of possible bitwise dialogues for a fixed verifier and a fixed input x. Every prover defines a subtree in this tree. If 3 bits of randomness are used, the optimal strategy has a probability of acceptance of 6/8.

Now it is easy to construct an optimal prover by induction: we need to select the subtree that has the maximal number of accept leaves in it. We do this from bottom to top. We assign a value 1 to every accept leaf. At each level in the tree that corresponds to a message from the verifier, the value of the nodes are equal the sum of the values of the children. At each level corresponding to a message from the prover, the values of the nodes is the maximum of the children's values. By induction one shows that the root value is the maximal number a strategy for the prover can have; and this determines the maximal probability for which the verifier accepts x. In the figure, these numbers are represented in blue and the optimal strategy is represented in red.

The tree is exponentially large, but we can compute the value of the root in polynomial space. We traverse the tree in a depth-first way. We only need to store the values in all nodes of the current branch, and each such value is bounded by $2^{t(|x|)}$. Because the depth is polynomial, the algorithm runs in polynomial space.

Exercise 4. In the definition we assumed that P is any deterministic device (that might produce non-computable answers). We can generalize the definition to non-deterministic provers where the probabilities in Definition 1 also include randomness over the prover. Show that this would not change the definition of the class IP, i.e., languages that have proof systems with non-deterministic provers also have proof systems with deterministic provers.

4 #SAT $\in$ PSPACE

To prove that PSPACE $\subseteq$ IP, we will show in the next section that $TQBF \in$ IP, i.e., there is an interactive proof system for the set of true quantified Boolean formula. In this section we show an easier result using the same proof technique.

Proposition 4. *The set #SAT (pronounced as sharp-SAT) defined by*

$$\{\langle k, \phi\rangle : \phi \textit{ is a Boolean formula with precisely } k \textit{ different satisfying assignments}\},$$

is in IP*.*

This result is already non-trivial, for example it implies that coNP $\subseteq$ PSPACE. Indeed, the elements of this set with $k = 0$ determine precisely the set of unsatisfiable formula ϕ. This set is complete for coNP, (because its complement is complete for NP).

We prove the proposition using the technique of arithmetization, which we already used when we studied branching programs: we reduce our problem to equality of polynomials with low degree modulo some large prime number. Then we use the fundamental theorem of algebra.

Lemma 5. *If f is a polynomial with integer coefficients and q is a prime, then the equation $f(x) = 0 \bmod q$ has at most d/q different solutions x modulo q.*

From this theorem it follows that by checking equality of the polynomials in a random r in the interval $[0, q-1]$, we can decide equality with high probability. In this section we encode polynomials only using multiplications and additions, we do not use powers. We allow any such coding, for example: $(x + y) \cdot (x \cdot x + 1)$. This implies that polynomials always have polynomially bounded degree in there input sizes. We first explain how Proposition 4 can be formulated in terms of polynomials. More precisely, we show how the proposition follows from the following result.

Corollary 6. *Let d_p be the maximal degree of p in a variable. The following set is in* IP*:*

$$\begin{aligned} \textit{SUM-POLY} = \Big\{ \langle q, k, p\rangle :\ & p \textit{ is a polynomial in } m \textit{ variables,} \\ & q \textit{ is a prime that exceeds } 3md_p, \\ & \sum_{x_1\in\{0,1\}} \sum_{x_2\in\{0,1\}} \cdots \sum_{x_m\in\{0,1\}} p(x_1, x_2, \ldots, x_m) = k \bmod q \Big\}. \end{aligned}$$

Moreover, there is a protocol that accepts all inputs in SUM-POLY with probability 1*.*

In the proof of Proposition 4 we use a result about prime number that is called Bertrand's postulate: for every n there is a prime number q with $n < q \leq 2n$. We also use the result from a few lectures ago where we proved that the set of prime numbers is in BPP, moreover our probabilistic algorithm answers always correctly on input a prime number.

Proof of Proposition 4 using Corollary 6. We convert a Boolean formula to a polynomial such that if the formula is true for some values of the variables, then the polynomial is one, and otherwise the polynomial is zero. This is done by the following replacement of logical operators:

$$\begin{aligned} x \wedge y &\to x \cdot y \\ x \vee y &\to x + y - x \cdot y \\ \overline{x} &\to 1 - x. \end{aligned}$$

Let p be the polynomial obtained from a Boolean formula ϕ. Assume that ϕ has m arguments. The condition $\langle k, \phi \rangle \in \#SAT$ is equivalent to

$$k = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_m \in \{0,1\}} p(x_1, x_2, \ldots, x_m)$$

The sum at the right is at most 2^m, so it suffices to check the above equation modulo some prime q with $2^m < q \leq 2^{m+1}$. Such a prime exists by Bertrand's postulate. The idea is that the prover first sends this prime, and the verifier checks this is indeed a prime using a decider that satisfies the conditions of the class RP. The protocol is as follows:

Data: k and Boolean formula ϕ.
Result: Accepts if ϕ has precisely k different satisfying assignments, otherwise, accepts with probability at most 1/3.

P: Sends a prime number q with $2^m < q < 2^{m+1}$

V: Checks the bounds for q and runs the algorithm to check that q is prime. If this fails, the verifier rejects. Otherwise he computes the arithmetization p of ϕ.

P, V: Run the interactive procedure of the corollary to check that $(q, k, p) \in \mathit{SUM\text{-}POLY}$, using a protocol that satisfies the conditions of from Corollary 6.

It is easy to verify that if $(k, \phi) \in \#SAT$ the protocol always accepts. Assume $(k, \phi) \notin \#SAT$. If the prover sends a q that is not prime, the verifier reject with probability 1/3 in the 2nd step. Otherwise, he will reject with probability 1/3 in the last step by the conditions of Corollary 6. □

Corollary 6 follows from Lemma 7 below.

Lemma 7. *Let ℓ_p be the number of additions and multiplications needed to evaluate a polynomial p. There exists an interactive proof system (P, V) and a polynomial t, such that*

- *If $\langle q, k, p \rangle \in \mathit{SUM\text{-}POLY}$, then $(V \leftrightarrow P)$ always accepts.*
- *If $\langle q, k, p \rangle \notin \mathit{SUM\text{-}POLY}$, then for any prover P', $(V \leftrightarrow P')$ accepts with probability at most md_p/q.*
- *For any prover P' and any input, V performs at most $(m+1)t(\log q, \ell_p)$ computation steps.*

Exercise 5. What is the problem with the following protocol to prove Lemma 7?

V: Verifies that q is a prime that exceeds $3md_p$ using the algorithm mentioned above. Reject if this check fails.

P: Sends a polynomial $f(X)$ in the form $a_0 + a_1X + \cdots + a_dX^d$ with coefficients in $[0, q-1]$, such that
$$f(X) = \sum_{x_2} \cdots \sum_{x_m} p(X, x_2, \ldots, x_m) \mod q.$$

V: Verifies that $k = f(0) + f(1) \mod q$ and rejects if this is false.

V, P: Use the induction hypothesis to verify that $f(0)$ and $f(1)$ are correct, i.e., that $f(i) = \sum_{x_2} \cdots \sum_{x_m} p(i, x_2, \ldots, x_m) \mod q$ for $i \in \{0, 1\}$.

Proof. The idea of the protocol is as follows: the prover sends
$$f(X) = \sum_{x_2} \cdots \sum_{x_m} p(X, x_2, \ldots, x_m) \bmod q.$$
the verifier checks that $k = f(0) + f(1) \mod q$. If this is false he rejects. Otherwise, he will believe the result if he is convinced that the polynomial f is equal to the sum above. In other words, he needs to believe in the equality of both polynomials modulo q. For this, he selects a random r and checks the equality for $X = r$. He knows the left side $f(r)$. Thus he needs to believe that
$$f(r) = \sum_{x_2} \cdots \sum_{x_m} p(r, x_2, \ldots, x_m) \bmod q.$$
This is precisely the same problem as before, but with one variable less.

V: Verifies that q is a prime that exceeds $3md_p$ using the algorithm mentioned above. Reject if this check fails.
If $m = 0$: directly check that $p = k \bmod q$. Accept or reject accordingly.

P: Sends a polynomial $f(X)$ in the form $a_0 + a_1X + \cdots + a_dX^{d_p}$ with coefficients in $[0, q-1]$, such that
$$f(X) = \sum_{x_2} \cdots \sum_{x_m} p(X, x_2, \ldots, x_m) \bmod q.$$

V: Verifies that $k = f(0) + f(1) \mod q$ and rejects if this is false. Otherwise, he sends a random number $r \in [0, q-1]$.

V, P: Use recursion to verify that $f(r) = \sum_{x_2} \cdots \sum_{x_m} p(r, x_2, \ldots, x_m) \bmod q$,
i.e., that $\langle q, f(r), p(r, \cdot, \ldots, \cdot)\rangle \in \mathit{SUM\text{-}POLY}$.

If q is not a prime that exceeds $3md_p$ this will be detected with probability $1/3$ in the first step. In this case the protocol satisfies all conditions. Hence, we assume q satisfies the condition. The lemma is proven by induction on the number m of variables in p. If $m = 0$, then p evaluates to a constant and the verifier can directly check whether this constant equals k modulo q. The verifier answers correctly with probability 1. This requires a time polynomial in $\log q$ and ℓ_p.

Assume $m \geq 1$. The computation time of the algorithm is bounded by a polynomial in $\log q$ and ℓ_p. Let t be this polynomial. The total computation time is at most $t(\log q, \ell_p) + ((m-1)+1) \cdot t(\log q, \ell_p)$ by the induction hypothesis.

For the prover P described above, the verifier always accepts inputs $\langle q, k, p\rangle \in \mathit{SUM\text{-}POLY}$. Indeed, this last assumption implies $f(0)+f(1) = k$ and $f(r) = \sum_{x_2} \cdots \sum_{x_m} p(r, x_2, \ldots, x_m) \bmod$

q. This implies that $\langle q, f(r), p(r, \cdot, \ldots, \cdot)\rangle \in \mathit{SUM\text{-}POLY}$. Indeed, the condition $q \geq 3md_p$ is true for this new recursive call, because m decreases by one and d_p can not increase. By the induction hypothesis, V always accepts.

It remains to show that if $\sum_{x_1} \cdots \sum_{x_m} p(x_1, \ldots, x_m) \neq k \bmod q$, the probability of acceptance is at most md_p/q. If in the second step, the prover sends a correct polynomial, then $f(0) + f(1) \neq k \bmod q$ and the verifier immediately rejects. Thus he must send a wrong polynomial $\tilde{f}$. Now there are again two cases: If the verifier selects an r where $\tilde{f}(r) = f(r)$ he will accept, and this can happen with probability at most d_p/q by Lemma 5. Otherwise, $\tilde{f}(r) \neq f(r)$ and by the induction hypothesis the probability at most $(m-1)d_p/q$. The total probability of acceptance is at most md_p/q. $\square$

5 PSPACE $\subseteq$ IP

Proposition 8. PSPACE $\subseteq$ IP.

In Chapter 8, we learned that the set *TQBF* of true quantified Boolean formulas is PSPACE-complete. Consider the set of formulas F given by

$$\exists x_1 \exists x_2 \ldots \exists x_m \phi(x_1, \ldots, x_m),$$

for some Boolean formula ϕ. This set is in NP, so it is easy to come up with an interactive proof system. But we can also use the protocol from Corollary 6: A verifier can arithmetize ϕ and replace all $\exists$ symbols by $\sum$ symbols. We ask the prover to evaluate the expression. If he tells the result is zero, we reject, and otherwise we verify the result with the protocol from Corollary 6.

Now, we would like to do something similar with expressions that also contain $\forall x$ quantifiers, for example:

$$\exists x_1 \forall x_2 \ldots \exists x_{m-1} \forall x_m \phi(x_1, \ldots, x_m).$$

A natural idea is to replace the $\forall x$ quantifiers by $\prod_x$. Then the idea would be to prove a version of Corollary 6 that also has products in it, something like:

$$\sum_{x_1 \in \{0,1\}} \prod_{x_2 \in \{0,1\}} \cdots \sum_{x_{m-1} \in \{0,1\}} \prod_{x_m \in \{0,1\}} p(x_1, \ldots, x_m) \bmod q.$$

Unfortunately, there is a problem. Taking products rather than sums increases the degree of the polynomial that needs to be send by the prover. In the above example, the polynomial for the variable of the first quantifier, i.e.,

$$f(X) = \prod_{x_2 \in \{0,1\}} \cdots \sum_{x_{m-1} \in \{0,1\}} \prod_{x_m \in \{0,1\}} p(X, x_2, \ldots, x_m) \bmod q.$$

can have exponential degree and the prover might not be able to send it in polynomial time.

The solution is to use a third operator in the arithmetization, which we call linearization operator. Given a polynomial p in one variable, the polynomial

$$y \to y \cdot p(0) + (1-y) \cdot p(1)$$

is linear and has the same values in 0 and 1. Inserting such operators inside the chain of summations and multiplications does not affect the outcome.

For easier notation in the induction, we define a more general operator[1]

Definition 9. *The linearization operator* $\mathrm{L}_{x\to y}$ *maps a polynomial* p *with variables* x *and* y *to a polynomial* $\mathrm{L}_{x\to y}[(x,y,\dots)\to p(x,y,\dots)]$ *to a polynomial*

$$(y,\dots)\to(1-y)\cdot p(0,y,\dots)+y\cdot p(1,y,\dots).$$

Proof of Proposition 8. We show that $TQBF \in \mathrm{IP}$. We describe now how the verifier and the prover in the definition of IP operate. The input is a fully quantified Boolean formula

$$\exists\!\!\!\forall x_1 \exists x_2 \dots \exists\!\!\!\forall x_m \phi(x_1,\dots,x_m),$$

where $\exists\!\!\!\forall$ is either $\exists$ or $\forall$. We convert this formula to an arithmetical expression the evaluates to one if the formula is true and to zero if it is false. We convert ϕ to a polynomial using the same operations as before:

$$\begin{array}{rcl} x\wedge y & \to & xy \\ x\vee y & \to & x+y-xy \\ \overline{x} & \to & 1-x. \end{array}$$

Then we replace the quantifiers using the operations:

$$\begin{array}{rcl} \forall x\,(B(x)) & \to & \prod_x B(x) \\ \exists x\,(B(x)) & \to & \widetilde{\sum}_x B(x) \text{ which evaluates to } B(0)+B(1)-B(0)B(1) \end{array}$$

and obtain a formula

$$\prod\!\!\!\!\sum_{x_1\in\{0,1\}} \prod\!\!\!\!\sum_{x_2\in\{0,1\}} \cdots \prod\!\!\!\!\sum_{x_m\in\{0,1\}} p(x_1,\dots,x_m),$$

where $\prod\!\!\!\!\sum \in \left\{\prod, \widetilde{\sum}\right\}$. It follows directly that this expression evaluates to 1 if the original formula is true, and to zero otherwise.

In the next step we insert linearization operators at the left of each $\prod\!\!\!\!\sum$ operation for all free variables. At the same time, we create dummy variables in the original polynomial and rename the variables to obtain an expression:

$$\prod\!\!\!\!\sum_{y_1} \mathrm{L}_{y_2\to y_1} \prod\!\!\!\!\sum_{y_3} \mathrm{L}_{y_4\to y_2}\, \mathrm{L}_{y_5\to y_3} \prod\!\!\!\!\sum_{y_6} \mathrm{L}_{y_7\to y_4}\, \mathrm{L}_{y_8\to y_5}\, \mathrm{L}_{y_9\to y_5} \prod\!\!\!\!\sum_{y_{10}} \cdots \prod\!\!\!\!\sum_{y_{m(m+1)/2}} p(y_1,\dots,y_{m(m+1)/2}).$$

These linearization operators make sure that at each step of the evaluation of these operators from right to left, (as is done by the prover), the degree of each variable is always at most 2, except in the beginning where it might be at most $2d_p$. We write this formula formally as $Q_{y_1,\dots,y_e} p(y_1,\dots,y_e)$ where Q represents a sequence of operators. These operators destroy the arguments of the polynomials in the same order of appearance from left to right. Linearisation operators only depend on variables that are destroyed at the left of them.

Now choose the smallest prime q that is at least $3\cdot 2d_p\cdot m(m+1)/2$, and apply the protocol from the following corollary (which generalizes Corollary 6) with $m \leftarrow m(m+1)/2$ and $k=1$ (which means that the original quantified Boolean formula is true).

[1]The name "linearization operator" does not cover this term anymore. On the other hand, "reduction operator" is also uninformative because reduction can have many meanings in theoretical computer science.

Corollary 10. *Let d_{Qp} be the maximal degree of a variable that appears in the stepswise evaluation of a sequence of operators Q on p. The set SUM-PROD-POLY given by*

$$\Big\{\langle q,k,Q,p\rangle : \ p \text{ is a polynomial in } m \text{ variables,}$$
$$Q = Q^{(1)}Q^{(2)}\dots Q^{(m)} \text{ such that } \forall i \le m : Q^{(i)}_{y_1\dots y_m} \in \left\{\widetilde{\sum_{y_i}}, \prod_{y_i}, \underset{y_i\to y_1}{\mathrm{L}}, \dots, \underset{y_i\to y_{i-1}}{\mathrm{L}}\right\}$$
$$q \text{ is a prime that exceeds } 3d_{Qp}m,$$
$$Q_{y_1,\dots,y_m} p(y_1,y_2,\dots,y_m) = k \bmod q\Big\}$$

is in IP. *Moreover, there is a protocol that accepts all inputs in SUM-POLY with probability* 1.

If this protocol accepts, then also accept, otherwise reject. It is easy to see that this protocol satisfies the conditions of the proposition. Hence, to finish the proof it remains to prove the corollary above. □

Note that by definition the first operator in Q of an element of *SUM-PROD-POLY* can not be a linearization operator, because such an operator requires two arguments, and the second argument should appear strictly before the first argument, which is impossible. For induction purposes, we need a forth operator. This is a variant of the linearization operator and it can be used with induction. For a constant r, the operator $\underset{y_i\to r}{\mathrm{L}}$ maps a polynomial with a variable y_i to

$$(1-r)\cdot p(\dots,0,\dots) + r\cdot p(\dots,1,\dots).$$

(The value r does not appear inside the polynomial, the idea is that in the protocol at the moment when the variable y_j of $\underset{y_i\to y_j}{\mathrm{L}}$ is destroyed, the value of y_j is fixed to r both in the polynomial and in the linear operator that contains y_j.) We consider now the set *SUM-PROD-POLY'* which is the same as in Corollary 10, but the sequence Q can take operators also such values, i.e.,

$$Q^{(i)}_{y_1\dots y_m} \in \left\{\widetilde{\sum_{y_i}}, \prod_{y_i}, \underset{y_i\to y_1}{\mathrm{L}}, \dots, \underset{y_i\to y_{i-1}}{\mathrm{L}}, \quad \underset{y_i\to 0}{\mathrm{L}}, \dots, \underset{y_i\to q-1}{\mathrm{L}}\right\}.$$

Corollary 10 follows directly from the following lemma.

Lemma 11. *There exists an interactive proof system (P,V) and a polynomial t, such that*

- *If the input belongs to SUM-PROD-POLY', then $(V \leftrightarrow P)$ always accepts.*
- *For any input outside SUM-PROD-POLY' and any other prover P', $(V \leftrightarrow P')$ accepts with probability at most md_p/q.*
- *For any prover P' and any input, V performs at most $(m+1)\cdot t(\log q, \ell_p)$ computation steps.*

Proof. The proof system (P,V) operates as follows.

V: Verifies that q is a prime that exceeds $3md_p$ using the algorithm mentioned above. Reject if this check fails.
If $m = 0$: directly check that $p = k \bmod q$. Accept or reject accordingly.
Otherwise, let $Q = Q^{(1)}\tilde{Q}$ (i.e., $\tilde{Q}$ contains all operators except for the first).

P: Sends a polynomial that maps Y to

$$\tilde{Q}_{Y,y_2,\dots,y_m} p(Y, y_2, \dots, y_m) \bmod q.$$

V: Verifies that

$$k \bmod q = \begin{cases} f(0)f(1) & \text{if } Q^{(1)}_{y_1,\dots,y_m} = \prod_{y_1} \\ f(0) + f(1) - f(0)f(1) & \text{if } Q^{(1)}_{y_1,\dots,y_m} = \widetilde{\sum}_{y_1} \\ (1-\tilde{r}) \cdot f(0) + \tilde{r} \cdot f(1) & \text{if } Q^{(1)}_{y_1,\dots,y_m} = \mathrm{L}_{y_1 \to \tilde{r}}. \end{cases}$$

He rejects if this is false. Otherwise, he sends a random number $r \in [0, q-1]$.

V, P: Use the induction hypothesis to verify that

$$f(r) = \tilde{Q}_{r,y_2,\dots,y_m} p(r, y_2, \dots, y_m) \bmod q.$$

The argument why this protocol satisfies the conditions is very similar as the proof of Lemma 7. The lemma, and hence the main result is proven. □

Exercise* 6. In this exercise you are asked to adapt the argument that #SAT is in IP. A black box contains a polynomial p in a field of exponential size over n variables. The degree in each variable is bounded by n. This polynomial is a very complex object and any description of it has exponential length. On input $x_1, \dots, x_n$, this black box evaluates this polynomial.

A *verifier* wants to check n values of the polynomial:

$$\begin{aligned} p(u_1, \dots, u_n) &= a \\ p(v_1, \dots, v_n) &= b \\ &\dots \\ p(z_1, \dots, z_n) &= e, \end{aligned}$$

but he can use the black box at most once. Fortunately, a *prover* claims to know p and has unlimited computational power to convince the verifier the values are correct. Is there a protocol such that a prover can convince the verifier in polynomial time that the values are correct?

Note that the following does not work: the prover sends the polynomial, then the verifier checks all values, and finally he checks whether the sent polynomial p is correct by sampling the device on a single random point. This would work if p had a description of polynomial size, but this is not the case.