$IP = PSPACE$

Ville Salo Ilkka Törmä

May 13, 2013

Abstract

In this essay, we prove that the complexity classes *PSPACE* and *IP* coincide. We also discuss some other models of interactive computation with varying prover strength and multiple provers.

1 Introduction

Consider a recursive language $L \subset \{0,1\}^*$ decided by the Turing machine M, for example the language of palindromes. We can view L as a property that any given word may or may not have (in this case, being a palindrome). Given a word $w \in \{0,1\}^*$, we can feed it into the machine M, which inspects it and, after some time, halts and announces whether it was a palindrome or not. This procedural view sees computational devices as black boxes that take objects in, process them and output a single bit indicating acceptance or rejection.

Another, more mathematically oriented, possibility is to see the computation of M on a word w as a proof that w is or is not a palindrome. With this intuition, recursively enumerable languages are those for which a proof for membership can always be found, and for recursive languages, so can a proof for non-membership. The polynomial class P consists of those languages for which a proof can be found in polynomial time, while in *NP*, a proof can be verified in polynomial time once it is found. In probabilistic classes, one can quickly produce a proof that is correct with some (usually arbitrarily high) probability.

This intuition of computation as proof motivates the following scenario. Suppose an agent P has obtained a truly marvelous proof for the fact that $w \in L$, and wants to convince another agent V of its correctness. Due to lack of computational resources (like the width of a margin), P cannot present the whole proof to V, but they can together discuss the main ideas until V is satisfied enough. We can model P as a computational device with unbounded resources (after all, she already has the proof and understands it completely) that interacts with a restricted device V, and together they should come to the conclusion that $w \in L$ with a high probability. But V should not be too gullible, so erroneous proofs should also be detected with a high probability.

It is a surprising result that such a computational model is equivalent in power to a class of algorithms bounded by computation space. This is the

important $IP = PSPACE$ theorem which we prove in the course of this essay. Our proof follows the one exhibited in [6]. Variants of the situation can be obtained by restricting the power of P (for example, by requiring it to be an oracle), or by adding multiple provers, either benevolent or malignant. Some of these models have been characterized in terms of space or time complexity, but not all, and we discuss them in the final section.

An interesting consequence of $IP = PSPACE$ is $IP = \text{co}-IP$, because obviously $PSPACE$ is closed under complementation. This is not obvious from the rather asymmetric definition of IP: If $w \in L$ for a language L in IP, then the honest prover P is able to convince the verifier V of this, but if $w \notin L$, no prover is likely to succeed. In the class $\text{co}-IP$, consisting of those languages whose complements are in IP, the prover tries to convince V that $w \notin L$, and words of L are accepted with a high probability regardless of the prover.

The asymmetry in the definition is important: If we only require that there exists an all-powerful prover P that can convince V with high probability of either $w \in L$ or $w \notin L$, whichever is the case, then the system can compute arbitrary functions, since P can simply give the correct answer to V, who always believes it. Similarly, if we only require that no prover can convince V of $w \in L$ or $w \notin L$ with high probability when the opposite holds, then V should simply ignore the prover altogether, and we obtain the probabilistic class BPP.

2 Definitions and Examples

We start with the definitions. An *interactive proof system* (P, V) consists of an arbitrary function $P : \{0,1\}^* \times (\{0,1\}^*)^* \to \{0,1\}^*$ (the *prover*) and a probabilistic Turing machine V (the *verifier*). The machine V has three tapes, the input tape (read-only, contains the input), the work tape (read-write, initially empty) and the communication tape (read-write, initially empty). It also has two special states q_{ask} and q_{rnd}. When the state q_{ask} is entered and the input and communication tapes contain the words w and $w^{(m)}$, respectively, the contents of the communication tape are replaced by $P(w, w^{(1)}, w^{(2)}, \ldots, w^{(m)})$ in one step, where the words $w^{(i)}$ are the previous messages sent to P. When the state q_{rnd} is entered, the read-write head of V that uses the work tape writes either 0 or 1 on the tape, with equal probability. An input word is *accepted* by the system if V eventually enters an accepting state, and rejected otherwise. A language $L \subset \{0,1\}^*$ is *accepted* by the system (P, V) if

- for all $w \in L$, (P, V) accepts it with probability at least $\frac{3}{4}$, and
- for all $w \notin L$ and all $P' : \{0,1\}^* \times (\{0,1\}^*)^* \to \{0,1\}^*$, the system (P', V) accepts w with probability at most $\frac{1}{4}$.

The probabilities are with respect to the random choices of V, and the values $\frac{3}{4}$ and $\frac{1}{4}$ could equally well be replaced by $1 - \epsilon$ and ϵ for an arbitrary $\epsilon \in (0, \frac{1}{2})$.

Intuitively, the prover P is an agent with unbounded resources that convinces V with high probability that $w \in L$ if this indeed is the case. The communication

tape is used to transfer messages between the two parties. However, if $w \notin L$, then with high probability, V can detect any erroneus proofs given by malicious agents P'. The agents cannot see the internal configuration of V, but can otherwise be any functions, even uncomputable ones. Thay can also remember all the previous messages of the verifier.

The class *IP* is the set of languages $L \subset \{0,1\}^*$ accepted by interactive proof systems (P, V) with polynomial time verifiers V. It was first introduced in [4].

Example 1 ([7]). *For an $n \times n$ matrix $A = (a_{ij})$, define the* permanent $\mathrm{per}(A)$ *as follows. If $n = 1$, then $\mathrm{per}(A)$ is the value of the only entry of A, and if $n > 1$, then $\mathrm{per}(A) = \sum_{j=1}^{n} a_{1j} \cdot \mathrm{per}(A_{1j})$, where A_{ij} is the (i,j)-minor of A (A without ith row and jth column). We sketch a proof for the fact that the $\#P$-complete language $\{(A, \mathrm{per}(A)) \mid A$ a 0-1 matrix$\}$ is in IP.*

The verifier V, upon receiving the input (A, a) where $a \in \mathbb{N}$ and A is an $n \times n$ matrix, first requests a large prime number $p \geq n!$ from the prover P. All calculations are then done modulo p, since $\mathrm{per}(A)$ cannot exceed $n!$. The verifier keeps a list $\langle (B_1, b_1), \ldots, (B_n, b_n) \rangle$ of matrices of the same size and numbers modulo p, starting with $\langle (A, a) \rangle$. The assumption here is that for all i, $b_i = \mathrm{per}(B_i)$. If the list contains a single pair (B, b) where $B = (b_{ij})$ has size $k \times k$, then V asks the prover for k numbers $c_1, \ldots, c_k$ (the permanents of the minors B_{1j}), checks that $\sum_{j=1}^{k} b_{1j} c_j = b$ and replaces the list with $\langle (B_{11}, c_1), \ldots, (B_{1k}, c_k) \rangle$.

If the list contains at least two elements (B, b) and (C, c), they are replaced by a single pair (D, d) as follows. The function $f(x) = \mathrm{per}(B + x(C - B))$ is a polynomial in $\mathbb{Z}_p$ of degree at most n, so V asks the prover for its formula, receiving a polynomial $g(x)$, and checks that $g(0) = b$ and $g(1) = c$. Then, V chooses a random element $e \in \mathbb{Z}_p$, and sets $D = B + e(C - B)$ and $d = g(e)$. Finally, when the list contains only a single 1×1 matrix (r) and a number q, the verifier checks that $r = q$, and rejects if not.

The idea here is that if the list contains a pair (B, b) such that $b \neq \mathrm{per}(B)$, then with very high probability, the list obtained after a reduction step also contains one. On the other hand, if the prover is honest, the reduction necessarily terminates with the correct list $\langle ((a), a) \rangle$.

3 Main Theorem

We now proceed to prove the remarkable result *IP = PSPACE*, which appeared first in [5].

Theorem 1. *IP = PSPACE*

Proof. The two inclusions are shown in Proposition 2 and Proposition 3. □

3.1 *PSPACE* $\subset$ *IP*

We first prove the inclusion *PSPACE* $\subset$ *IP* by defining an interactive proof system for the language of true quantified first-order boolean formulas, which is

known to be *PSPACE*-complete. This is done using a method called *arithmetization*, in which a logical formula is replaced by an arithmetic expression. In our case, a formula is transformed into an expression that evaluates to 0 iff the formula was false. Values of arithmetical formulas can in general be decided by *IP* protocols, similarly to how the permanent of a matrix was calculated in the example of the previous section.

Definition 1. *Let X be a set of variables. The quantified first-order formulas are defined inductively:* 0 *and* 1 *are formulas, as is x for each $x \in X$, and $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, $\neg\phi$, $(\forall x)\phi(x)$ and $(\exists x)\phi(x)$ are formulas when ϕ, ϕ_1 and ϕ_2 are. Freeness and boundedness of variables is defined as usual. We write QBF for the set of such formulas. When manipulated by machines, we assume any reasonable encoding for the formulas. We define the truth value of a formula in an obvious way, when a valuation $X \to \{0,1\}$ for the variables is given, the quantified variables ranging over $\{0,1\}$.*

Here, 'reasonable' in particular means that the length of the encoding is a linear function of the amount of paper one needs to write the formula down, and that one can quickly browse the inductive structure of the formula.

We first simplify our formulas, as this will lead to polynomials of low degree in the arithmetization phase that follows. We write $\phi' \sqsubset \phi$ when ϕ' is a subformula of ϕ.

Definition 2. *A QBF formula ϕ is* simple *if*

- *for all subformulas $\neg\psi$ in ϕ, we have $\psi \in X$.*
- *if $(\exists x)\psi_1 \sqsubset \phi$ or $(\forall x)\psi_1 \sqsubset \phi$, and $(\forall y)\psi_2 \sqsubset \psi_1$ and $(\forall z)\psi_3 \sqsubset \psi_2$, then ψ_3 does not contain x as a free variable.*

The latter condition means that after a variable is quantified, it can occur in the scope of at most one nested $\forall$-quantifier.

Lemma 1. *There exists a polynomial time algorithm* Simplify *such that ϕ is true if and only if* Simplify(ϕ) *is and* Simplify(ϕ) *is in simple form.*

Proof. Negations are handled by using De Morgan's laws to move them inward: $\neg(\phi \vee \phi') \mapsto \neg\phi \wedge \neg\phi'$ and $\neg(\exists x)\phi(x) \mapsto (\forall x)\neg\phi(x)$, and similarly for $\wedge$ and $\forall$.

The second property is obtained by recursively applying the transformation

$$(\forall x)\phi(x, y_1, \ldots, y_n) \mapsto$$

$$(\forall x)(\exists y'_1, \ldots, y'_n)(y'_1 \leftrightarrow y_1 \wedge \cdots \wedge y'_n \leftrightarrow y_n \wedge \phi(x, y'_1, \ldots, y'_n))$$

to subexpressions, top-down, where $\phi(x, y_1, \ldots, y_n)$ denotes that $x, y_1, \ldots, y_n$ are the free variables in ϕ, y'_i are new variables and $\phi_1 \leftrightarrow \phi_2$ is a shorthand for $(\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \neg\phi_2)$. □

We skip the exact definition of the algorithm, as this is a straightforward manipulation of the expression. The size of the resulting expression is $O(n^2)$, if n is the size of the original one. For example,

$$(\forall x_1)(\forall x_2)(\forall x_3)(x_1 \vee x_2 \vee x_3)$$

becomes

$$\begin{aligned}(\forall x_1)(\forall x_2)(\exists x_1')(x_1 \leftrightarrow x_1' \wedge \\ (\forall x_3)(\exists x_1'', x_2')(x_1'' \leftrightarrow x_1' \wedge x_2' \leftrightarrow x_2 \wedge \\ (x_1'' \vee x_2' \vee x_3)))\end{aligned}$$

in the transformation.

Next, from a simple QBF ϕ, we construct an arithmetical expression over natural numbers which evaluates to 0 if and only if ϕ is false (but can evaluate to a huge number if ϕ is true).

Definition 3. *The quantified arithmetical formulas are defined inductively similarly to QBFs: the constants* 0 *and* 1 *are formulas, x is a formula for each $x \in X$, and $A_1 + A_2$, $A_1 \cdot A_2$, $-A$, $\prod_{x=0}^{1} A(x)$ and $\sum_{x=0}^{1} A(x)$ are formulas when A, A_1 and A_2 are. Freeness and boundedness of variables is defined as usual. We write QAF for the set of such formulas. We again assume a reasonable encoding and define the value of a formula in the obvious way. We write $Z(A)$ to denote the value of a QAF A, and if $A(x)$ contains the free variable x, we write $Z(A(x))$ for the polynomial $A(x)$ evaluates to.*

Lemma 2. *There exists a polynomial time algorithm* Arithmetize *mapping simple QBFs to QAFs such that ϕ is true if and only if $Z(\textsc{Arithmetize}(\phi)) \neq 0$, and $|\textsc{Arithmetize}(\phi)| = O(n)$.*

Proof. Let $\phi \in QBF$. We change $\vee$ into addition, $\wedge$ into multiplication, $\neg x$ into $1 - x$, $(\exists x)$ into $\sum_{x=0}^{1}$ and $(\forall x)$ into $\prod_{x=0}^{1}$.

Evaluating this expression with the obvious interpretations $\sum_{i=0}^{1} \phi(i) = \phi(0) + \phi(1)$ and $\prod_{i=0}^{1} \phi(i) = \phi(0)\phi(1)$ results in 0 if and only if ϕ was false: $f(x) : \mathbb{N} \to \{0, 1\}$ defined by $f(x) = 0 \iff x = 0$ is a lattice homomorphism from $(\mathbb{N}, \leq)$ to $(\{0, 1\}, \leq)$ and negations are applied only to variables in ϕ.

The claim $|\textsc{Arithmetize}(\phi)| = O(n)$ is trivial. □

There is a large upper bound for the value of a QAF depending on its length.

Lemma 3. *Let A be a QAF of size n. Then $Z(A) \leq 2^{2^n}$.*

Proof. If $A = B + C$, $A = BC$, $A = 1 - a$ or $A = a$ for $a \in \{0, 1\}$, the claim is clear. For $A = \sum_{x=0}^{1} B(x)$, we have $A = B(0) + B(1) \leq 2^{2^{n-1}} + 2^{2^{n-1}} \leq 2^{2^n}$ and for $A = \prod_{x=0}^{1} B(x)$, we have $A = B(0)B(1) \leq 2^{2^{n-1}} 2^{2^{n-1}} = 2^{2^n}$. □

We cannot give a much better bound since

$$Z(\textsc{Arithmetize}(\bigwedge_{x_1} \cdots \bigwedge_{x_n} (1 \vee 1))) = Z(\prod_{x_1=0}^{1} \cdots \prod_{x_n=0}^{1} (1 + 1)) = 2^{2^n},$$

so the verifier cannot really evaluate any subexpressions (the formula above is not simple, but making it simple would not change the value of its arithmetization). Thus, the protocol starts with the prover giving the verifier a suitable prime p, and evaluations will be done in the field $\mathbb{Z}_p$. We start by proving that suitable prime numbers always exist.

Lemma 4. *There exists a constant C such that for all $n \neq 0$ and $m \geq n$, we have $n \not\equiv 0 \mod p$ for some prime $\frac{1}{2}C \log_2 m \leq p \leq C \log_2 m$.*

Proof. We use some well-known number theoretical results. The first Chebyshev function is defined by

$$\vartheta(m) = \sum_{p \leq m} \log p,$$

where p ranges over primes. It is known that

$$|\vartheta(m) - m| = O(m^{\frac{1}{2}+\epsilon}),$$

for all $\epsilon > 0$. Thus, defining $v(m) = \vartheta(m) - \vartheta(m/2)$, there exists a constant E such that $v(m) \geq Em$ for all (large enough) m. Thus,

$$\prod_{\frac{1}{2}Dm \leq p \leq Dm} p = e^{v(Dm)} \geq e^m,$$

for some D and for all m, and then

$$\prod_{\frac{1}{2}C \log_2 m \leq p \leq C \log_2 m} p \geq m,$$

for some C and for all m.

Let $n_1, \ldots, n_k \in \mathbb{N}$ be pairwise coprime. Then $n \equiv 0 \mod \prod_i n_i$ if and only if $n \equiv 0 \mod n_i$ for all i (by the Chinese remainder theorem). In particular, if $\prod_i n_i \geq n$, then $n = 0$ if and only if $n \equiv 0 \mod n_i$ for all i.

Now, suppose $n \equiv 0 \mod p$ for some $m \geq n$ and for all primes $\frac{1}{2}C \log_2 m \leq p \leq C \log_2 m$, then choosing n_i to enumerate such primes, it follows from the Chinese remainder theorem that $m \equiv 0 \mod \prod_i n_i$. Since $\prod_i n_i \geq m \geq n$, we have $n = 0$. □

We say that a QAF is *simple* if each variable x is in the scope of at most one nested product, so that ARITHMETIZE(ϕ) is simple if ϕ is.

Lemma 5. *Let $\sum_{x=0}^{1} A(x)$ be a simple QAF of size n. Then the polynomial $Z(A(x))$ in the single variable x has degree at most $2n$.*

Proof. We show by induction that for any subexpression $B(x)$ of $A(x)$, the degree of $B(x)$ is at most $|B(x)|$ if the variable x does not occur in $B(x)$ in the scope of a product quantifier, and at most $2|B(x)|$ otherwise. The first kind of subexpressions are called *light* in the following.

If $|B_1(x)| = m_1$ and $|B_2(x)| = m_2$, then the degree of $B_1(x) + B_2(x)$ and the degree of $B_1(x)B_2(x)$ are at most $m_1 + m_2$, and the degree of $\sum_{y=0}^{1} B_1(x, y)$ is at most m_1 (where y is free in B_1), so that the inductive assumptions are satisfied in these operations for both light and non-light $B_1(x)$ and $B_2(x)$.

The degree of $\prod_{y=0}^{1} B_1(x, y)$ is at most m_1 if x does not occur in $B_1(x, y)$. If it does occur, then the degree is at most $2m$, but then $B_1(x, y)$ must be light, so that $2m_1 \leq 2|B_1(x, y)|$, which concludes the inductive claim since $\prod_{y=0}^{1} B_1(x, y)$ is not light. □

In particular, $Z(A(x)) \equiv d(x)$ for some $d(x) \in \mathbb{Z}_p[x]$ of degree at most $2n$.

Proposition 1. *There exists an IP protocol for solving QBF. In fact, if the input formula ϕ is true, the protocol returns true with probability* 1.

Proof. The protocol is the function QBFSat in Algorithm 1 for large enough $n = |A|$; small cases can be handled separately by a truth table, obtaining exact answers. We show the correctness of the algorithm.

First, suppose the function Zero in Algorithm 3 used as a subroutine works correctly, that is, it runs in polynomial time in the size of $|A|$ and p, returns true if $Z(A) \equiv 0 \mod p$, and otherwise returns true with probability at most $\frac{mk}{p}$ where k is the number of quantifiers in A and m the maximum degree of polynomials encountered on line 2 in the whole course of recursion. Note that we always have $k \leq n$ and $m \leq 2n$ if A is simple, by Lemma 5.

The function QBFSat is deterministic, so we only need to check that the function NonZero in Algorithm 2 works correctly. First, if $Z(A) \neq 0$ then an honest prover will convince the verifier running NonZero: The prover simply gives the verifier p and a such that $Z(A) \equiv a \mod p$, $\frac{C}{2}2^n \leq p \leq C2^n$, p is prime, and $a \not\equiv 0 \mod p$ (such values are guaranteed by Lemma 4). All these properties can be checked in polynomial time, except the first one (or primality can be checked in *NP* style by requesting a proof, if the reader does not believe that primality testing is in P). If $Z(A) = 0$, then for any p and $a \not\equiv 0 \mod p$ we have $Z(A) \not\equiv a \mod p$. As we are assuming Zero works correctly, Zero returns true with probability at most $\frac{3mk}{p}$. Since $p \geq \frac{C}{2}2^n$, the probability of failure is then at most

$$\frac{3mk}{p} \leq \frac{3mk}{\frac{C}{2}2^n} \leq \frac{1}{4},$$

for large n, since $k \leq n$ and $m \leq 2n$. It is easy to see that the algorithm runs in polynomial time if Zero does.

Thus, we only have to check that Zero works correctly. First, it is clear that it runs in polynomial time: In the course of recursion, Zero is called k times, once for every quantifier in A, and the lengths of numbers and loops are polynomial in either m or k. If $Z(A) \equiv 0 \mod p$, then the prover simply gives the polynomials $d_i(x)$ such that $Z(C_i(x)) \equiv d_i(x)$. The check

$$Z(B(Q^1_{x_1} d_1(x_1), \ldots, Q^m_{x_m} d_m(x_m))) \equiv 0 \mod p$$

then succeeds, and so does the check $\textsc{Zero}(C_i(a_i) - d_i(a_i), p)$, by induction on k.

The nontrivial part is to bound the probability that $\textsc{Zero}(A, p)$ returns true when $A \not\equiv 0 \mod p$. This is a function of the number of quantifiers and the maximal degree of a polynomial encountered in the decomposition on line 2. We prove by induction on the number of quantifiers k in A that the probability of returning true if $Z(A) \not\equiv 0 \mod p$ is at most $\frac{mk}{p}$. So, for a fixed m, we assume this is the case for smaller k, and prove the claim for A with k quantifiers. First, we note that the dishonest prover *must lie* when choosing one of the $d_i(x)$, that is, $Z(C_i(x)) \not\equiv d_i(x)$ for some i. Otherwise, the check on line 6 guarantees that

$$\begin{aligned} Z(A) &= Z(B(Q^1_{x_1} C_1(x_1), \ldots, Q^m_{x_m} C_m(x_m)) \\ &\equiv Z(B(Q^1_{x_1} d_1(x_1), \ldots, Q^m_{x_m} d_m(x_m))) \\ &\equiv 0 \mod p, \end{aligned}$$

so that in fact $Z(A) \equiv 0 \mod p$.

Denote the event that $\textsc{Zero}(A, p)$ returns true even though $Z(A) \not\equiv 0 \mod p$ by RT (short for 'returns true'). We denote the event that $Z(C_i(a_i) - d_i(a_i)) \equiv 0 \mod p$ by BG_i (short for 'bad guess') and the event that $\textsc{Zero}(C_i(a_i) - d_i(a_i), p)$ returns true by SRT_i (short for 'subroutine returns true').[1] Now, let j be such that $Z(C_j(x)) \not\equiv d_j(x)$. The polynomial $Z(C_j(x)) - d_j(x)$ has degree at most m and is not identically zero. Thus, it can have at most m roots, so that $\mathbb{P}(\mathrm{BG}_j) \leq \frac{m}{p}$. If, on the other hand, $Z(C_i(a_i) - d_i(a_i)) \not\equiv 0 \mod p$, then the check $\textsc{Zero}(C_i(a_i) - d_i(a_i), p)$ returns true with probability at most $\frac{m(k-1)}{p}$ by the induction hypothesis. So, we know $\mathbb{P}(\mathrm{BG}_j) \leq \frac{m}{p}$ and $\mathbb{P}(\mathrm{RT}_j \mid \neg \mathrm{BG}_j) \leq \frac{m(k-1)}{p}$. Due to the loop at line 11, in order for $\textsc{Zero}(A, p)$ to return true, all the events SRT_i must happen, so

$$\begin{aligned} \mathbb{P}(\mathrm{RT}) \leq \mathbb{P}(\bigwedge_i \mathrm{SRT}_i) &\leq \mathbb{P}(\mathrm{SRT}_j) \\ &\leq \mathbb{P}(\mathrm{BG}_j \vee (\mathrm{SRT}_j \wedge \neg \mathrm{BG}_j)) \\ &= \mathbb{P}(\mathrm{BG}_j) + \mathbb{P}(\mathrm{SRT}_j \mid \neg \mathrm{BG}_j)\mathbb{P}(\neg \mathrm{BG}_j) \\ &\leq \mathbb{P}(\mathrm{BG}_j) + \mathbb{P}(\mathrm{SRT}_j \mid \neg \mathrm{BG}_j) \\ &\leq \frac{m + mk - m}{p} \\ &\leq \frac{mk}{p}. \end{aligned}$$

□

Since the problem of finding out whether a given QBF is true is *PSPACE*-hard, we in fact have $PSPACE \subset IP$, since *IP* is trivially closed by polynomial time reductions.

Proposition 2. $PSPACE \subset IP$

[1] The events are conditioned on the computation history up to the choice of the a_i. The fact that we send a_i to the prover does not affect the probabilities of these events, as the polynomials $d_i(x)$ are fixed at that point.

Algorithm 1 Check a QBF instance. More precisely, given a QBF ϕ, return true with probability 1 if ϕ is true, and false with probability at least $\frac{3}{4}$ if ϕ is false.

1: **function** QBFSAT(ϕ)
2: $\quad\phi' \leftarrow$ SIMPLIFY(ϕ)
3: $\quad A \leftarrow$ ARITHMETIZE(ϕ')
4: $\quad$**return** NONZERO(A)

Algorithm 2 Given simple QAF A of size n, check $Z(A) \neq 0$. More precisely, given a simple QAF A, return true with probability 1 if $Z(A) \neq 0$, and false with probability at least $\frac{3}{4}$ if $Z(A) = 0$.

1: **function** NONZERO(A)
2: $\quad$**send** A
3: $\quad$**receive** prime $\frac{C}{2}2^n \leq p \leq C2^n$ ▷ receive p such that $Z(A) \not\equiv 0 \mod p$
4: $\quad$**receive** $a \in \mathbb{Z}_p$, $a \neq 0$ ▷ receive a such that $Z(A) \equiv a \mod p$
5: $\quad$**return** ZERO($A - a, p$)

Algorithm 3 Given simple QAF A of size n and p, check $Z(A) \equiv 0 \mod p$. More precisely, given a simple QAF A and prime p, return true with probability 1 if $Z(A) \equiv 0 \mod p$, and false with probability at least $1 - \frac{mk}{p}$ where k is the number of quantifiers in A and m the maximum degree of polynomials encountered on line 2 in the course of recursion, if $Z(A) \not\equiv 0 \mod p$.

1: **function** ZERO(A, p)
2: $\quad$**let** $A = B(Q^1_{x_1} C_1(x_1), \ldots, Q^m_{x_m} C_m(x_m))$
3: $\quad$**send** A
4: $\quad$**for all** $i \in \{1, \ldots, m\}$ **do**
5: $\quad\quad$**receive** $d_i(x) \in \mathbb{Z}_p[x]$ of degree $2n$ ▷ supposedly $Z(C_i(x)) \equiv d_i(x)$
6: $\quad$**if** $Z(B(Q^1_{x_1} d_1(x_1), \ldots, Q^m_{x_m} d_m(x_m))) \not\equiv 0 \mod p$ **then**
7: $\quad\quad$**return** false
8: $\quad$**for all** $i \in \{1, \ldots, m\}$ **do**
9: $\quad\quad$**randomize** $a_i \in \mathbb{Z}_p$
10: $\quad\quad$**send** a_i ▷ just because we can (see Section 4)
11: $\quad$**for all** $i \in \{1, \ldots, m\}$ **do**
12: $\quad\quad$**if** ZERO($C_i(a_i) - d_i(a_i), p$) **then**
13: $\quad\quad\quad$**return** false
14: $\quad$**return** true

3.2 *IP* ⊂ *PSPACE*

We now show that every language $L \in IP$ is also in *PSPACE*. The basic method for converting a probabilistic algorithm into a deterministic one is to simply calculate the probability of acceptance by cycling through every computation that the system may perform. In general, the time complexity of this approach exponential in the number of random bits used, but since a *PSPACE* algorithm may use exponential time, this is of no concern to us. First, if the *prover* runs in polynomial space, there is an easy reduction.

Lemma 6. *Let the language L be accepted by the system (P, V), where V runs in time n^k for some $k \in \mathbb{N}$. If P is PSPACE-computable, then L is in PSPACE.*

Proof. Consider Algorithm 4. It cycles through all the sequences of random bits that V might use during its computation on the input string $w \in \{0,1\}^n$, and simulates the system (P, V) with each of them, keeping count of the number c of accepting computations. If $c \geq 2^{n^k-1}$, then the probability that V accepts the input string w is at least $\frac{1}{2}$. This happens if and only if $w \in L$, by the definition of interactive proof systems. Thus the algorithm recognizes L.

Algorithm 4 Decide membership of L, assuming P is *PSPACE*-computable.

```
input w ∈ {0,1}*
N ← |w|^k
c ← 0
for all r ∈ {0,1}^N do
    simulate V using r as source of random bits, answering requests with P
    if V accepts then
        c ← c + 1
if c ≥ 2^(N−1) then
    accept
else
    reject
```

It remains to be shown that Algorithm 4 runs in polynomial space, but this is easy, since we only need to remember the sequence $r \in \{0,1\}^N$ of polynomial length, the internal state of V of polynomial size, the number c of polynomial length and whatever space the algorithm P requires. By assumption, P is *PSPACE*-computable, so the total space requirements are of polynomial size. □

In general, the prover P may not be *PSPACE*-computable, or even computable for that matter, and this may seem to complicate things immensely. However, we next show that P can always be replaced by a *PSPACE*-computable function without altering L. The proof relies on the intuition that interactive proof systems are actually games between the players P and V, where the goal of P is to maximize the probability of V accepting each input string $w \in L$. Thus the class *IP* contains games of polynomially many rounds, and *PSPACE*

is the natural computational class for most natural problems concerning such objects, like the computation of optimal strategies. Since we need to calculate probabilities, we define some notation for it.

Definition 4. *Let (P, V) be an interactive proof system, and let $w, w^{(i)}, v^{(i)} \in \{0,1\}^*$ be arbitrary. We denote by $\mathbb{P}_V(w^{(m+1)} \mid w, (w^{(i)})_{i=1}^m, (v^{(i)})_{i=1}^m)$ the conditional probability that the $(m+1)$st request of V has value $w^{(m+1)}$, given that the input is w and the m first requests have the values $w^{(i)}$ and the answers $v^{(i)}$. We also denote by $\mathbb{P}_{(P,V)}(\mathrm{acc} \mid w, (w^{(i)})_{i=1}^m, (v^{(i)})_{i=1}^m)$ the probability that V eventually accepts using the prover P, given the same conditions as above.*

Definition 5. *An* optimal prover *for the verifier V is a prover function P_{opt} such that for all $w \in \{0,1\}^*$, all message histories $(w^{(i)})_{i=1}^m, (v^{(i)})_{i=1}^m$ and all provers P, we have*

$$\mathbb{P}_{(P_{\mathrm{opt}},V)}(\mathrm{acc} \mid (w^{(i)})_{i=1}^m, (v^{(i)})_{i=1}^m) \geq \mathbb{P}_{(P,V)}(\mathrm{acc} \mid (w^{(i)})_{i=1}^m, (v^{(i)})_{i=1}^m).$$

The following lemma is clear by the definition of optimality, when we consider the case $m = 0$.

Lemma 7. *If the language L is accepted by the interactive proof system (P, V) and P_{opt} is an optimal prover for V, then L is accepted by the system (P_{opt}, V).*

An optimal prover P_{opt} always exists for a given language L and verifier V: For each input $w \in \{0,1\}^*$, there is a bound on the number of requests V can send, so there are finitely many behaviors a prover can exhibit. For the first message $w^{(1)}$ that V sends, we choose any prover P that maximizes the probability of V accepting w, and define P_{opt} to answer as P. For the second request $w^{(2)}$, we choose another prover $P_{w^{(1)}}$ that behaves as P on the first request, and maximizes the probability of V eventually accepting, given that it first sent the request $w^{(1)}$. We then define P_{opt} to answer as $P_{w^{(1)}}$ to the request $w^{(2)}$. Continuing inductively, an optimal prover for L and V is obtained, though it may be highly uncomputable. In the next lemma, we show that for *IP*-systems a *PSPACE*-computable optimal prover can be constructed.

Lemma 8. *Suppose that a verifier V runs in time n^k for some $k \in \mathbb{N}$. Then there exists a PSPACE-computable optimal prover for V.*

Proof. The idea of the algorithm is as follows. Given an input string and a request history, the prover P_{opt} first recalls its previous answers. It then cycles through the answers it may possibly give, and for each of them, calculates the probability that V eventually accepts if that answer is given, assuming that P_{opt} behaves optimally from that point on. This involves simulating V with all possible random choices it might make. When the calculation is done, P_{opt} simply chooses the answer that maximizes the probability of acceptance.

We make the following simplifying assumptions about V. First, all the requests of V and the answers it excepts are of length one. This can be achieved by V first sending its message to the prover one bit at a time, indicating that the

message has ended, and then requesting the answer one bit at a time. Second, we assume that the number of requests that V sends during its computation on an input word w is a polynomial function of $|w|$ only, and we denote it by $M(|w|)$. Third, the last message of V is 1 if and only if it accepts the input.

With these simplifications, the optimal prover P_{opt} is described in Algorithm 5, together with the auxiliary functions of Algorithm 6.

Algorithm 5 The optimal prover P_{opt} for the verifier V.

```
function P_opt(w, w^(1), ..., w^(m))
    for all i ∈ {1, ..., m − 1} do
        v^(i) ← P_opt(w, w^(1), ..., w^(i))          ▷ Compute previous answers
    p_0 ← P_acc(w, w^(1), ..., w^(m), v^(1), ..., v^(m−1), 0)
    if p_0 ≥ 1/2 then
        return 0
    else
        return 1
```

We now prove that P_{opt} is indeed an optimal prover for V, and for that, let $w \in \{0,1\}^*$ be arbitrary. First, we claim that the function $\mathbb{P}_0$, given inputs w and $w^{(1)}, \ldots, w^{(m)}, v^{(1)}, \ldots, v^{(m)}$, correctly calculates $\mathbb{P}_V(0 \mid w, (w^{(i)})_{i=0}^m, (v^{(i)})_{i=0}^m)$. The algorithm works by going through every possible sequence $r \in \{0,1\}^{|w|^k}$ of random bits that V might use in its computation, and simulating V from the initial configuration on each of them. Its m first requests are answered by the $v^{(i)}$, provided that they have the correct values $w^{(i)}$; otherwise, the computation is discarded. Then, we obtain the conditional probability of V requesting 0 by counting the number of computations where this happens, and dividing it by the number of all non-discarded computations.

Second, we show the correctness of $\mathbb{P}_{\text{acc}}$, which is supposed to calculate, on the inputs w and $w^{(1)}, \ldots, w^{(m)}, v^{(1)}, \ldots, v^{(m)}$, the conditional probability $\mathbb{P}_{(P,V)}(\text{acc} \mid w, (w^{(i)})_{i=0}^m, (v^{(i)})_{i=0}^m)$ for an optimal prover P (which is independent of the choice of P, as long as it is optimal). The algorithm is recursive, so we prove the claim by induction on $M(|w|) - m$. First, if $m = M(|w|)$, the function simply returns $w^{(m)}$, which has value 1 if V accepts, and 0 otherwise. Since the value of $w^{(m)}$ is known, the base case is correct. Let then $m < M(|w|)$. By the induction hypothesis and the previous paragraph, we have

$$p_b = \mathbb{P}_V(b \mid w, (w^{(i)})_{i=0}^m, (v^{(i)})_{i=0}^m)$$

and

$$p_{bc} = \mathbb{P}_{(P,V)}(\text{acc} \mid w, w^{(1)}, \ldots, w^{(m)}, b, v^{(1)}, \ldots, v^{(m)}, c)$$

for $b, c \in \{0,1\}$. Now, since P is optimal, it answers the next query of V maximizing the probability of acceptance, which is then exactly

$$p = p_0 \cdot \max(p_{00}, p_{01}) + p_1 \cdot \max(p_{10}, p_{11}),$$

since the two possibilities for the next request are disjoint random events.

Algorithm 6 Helper functions for P_{opt}.

function $\mathbb{P}_{\text{acc}}(w, w^{(1)}, \ldots, w^{(m)}, v^{(1)}, \ldots, v^{(m)})$
  if $m = M(|w|)$ **then**
    return $w^{(m)}$ ▷ Last request indicates acceptance
  else
    $p_0 \leftarrow \mathbb{P}_0(w, w^{(1)}, \ldots, w^{(m)}, v^{(1)}, \ldots, v^{(m)})$ ▷ Prob. of $w^{(m+1)} = 0$
    $p_1 \leftarrow 1 - p_0$
    $p_{00} \leftarrow \mathbb{P}_{\text{acc}}(w, w^{(1)}, \ldots, w^{(m)}, 0, v^{(1)}, \ldots, v^{(m)}, 0)$
    $p_{01} \leftarrow 1 - p_{00}$
    $p_{10} \leftarrow \mathbb{P}_{\text{acc}}(w, w^{(1)}, \ldots, w^{(m)}, 1, v^{(1)}, \ldots, v^{(m)}, 0)$
    $p_{11} \leftarrow 1 - p_{10}$
    return $p_0 \cdot \max(p_{00}, p_{01}) + p_1 \cdot \max(p_{10}, p_{11})$

function $\mathbb{P}_0(w, w^{(1)}, \ldots, w^{(m)}, v^{(1)}, \ldots, v^{(m)})$
  $a \leftarrow 0$ ▷ Total number of computations
  $b \leftarrow 0$ ▷ Number of computations with $w^{(m+1)} = 0$
  for all $r \in \{0,1\}^{|w|^k}$ **do** ▷ Sequence of random bits
    $C \leftarrow \text{start}(V)$ ▷ The initial configuration of V
    $i \leftarrow 0$ ▷ Counter for requests
    $j \leftarrow 0$ ▷ Counter for random bits
    while $i \leq m$ **do**
      $C \leftarrow \text{next}(C, V)$ ▷ Simulate one step of V
      if state in C is q_{rnd} **then**
        $C \leftarrow \text{write}(C, r_j)$ ▷ Supply a random bit
        $j \leftarrow j + 1$
      if state in C is q_{ask} **then**
        if $i \leq m$ **then**
          if communication tape in C contains $w^{(i)}$ **then**
            $C \leftarrow \text{answer}(C, v^{(i)})$ ▷ Supply an answer to request
          else
            skip to next r
        else
          $a \leftarrow a + 1$
          if communication tape in C contains 0 **then**
            $b \leftarrow b + 1$
        $i \leftarrow i + 1$
  return $\frac{b}{a}$

Third, we show that P_{opt} is an optimal prover, and for that, let P be any optimal prover, and consider an input word w and request and answer histories $(w^{(i)})_{i=0}^{m}, (v^{(i)})_{i=0}^{m-1}$, respectively. We show that in this situation, V is not less likely to accept the word w using P_{opt} than P. If $m = M(|w|)$, then $\mathbb{P}_{(P,V)}(\text{acc} \mid w, (w^{(i)})_{i=0}^{m}, (v^{(i)})_{i=0}^{m-1})$ is independent of P, so we suppose $m < M(|w|)$. If P and P_{opt} give a different answer to $w^{(m)}$, then necessarily $\mathbb{P}_{(P,V)}(\text{acc} \mid w, w^{(1)}, \ldots, w^{(m)}, v^{(1)}, \ldots, v^{(m-1)}, 0) = \frac{1}{2}$, and P_{opt} answered with 0 and P with 1. But by the induction hypothesis, P_{opt} acts as an optimal prover after the first m requests, so by the correctness of $\mathbb{P}_{\text{acc}}$, the acceptance of w is equally likely with P_{opt} as with P in this situation.

Finally, we prove that P_{opt} is computed in *PSPACE*. The function P_{opt} calls itself recursively at most $M(|w|)$ times, and inside each call, the helper function $\mathbb{P}_{\text{acc}}$ recursively calls itself at most $M(|w|)$ times. Further, each call to $\mathbb{P}_{\text{acc}}$ includes a single call to $\mathbb{P}_0$. In total, at any given time, the algorithm needs to remember at most the following data:

- In each of the at most $M(|w|)$ calls to P_{opt}, the inputs w and $w^{(i)}$, the previous answers $v^{(i)}$ and the index i.
- In each of the at most $M(|w|)$ calls to $\mathbb{P}_{\text{acc}}$, the inputs w, $w^{(i)}$ and $v^{(i)}$, and the probabilities p_i and p_{ij}.
- In a call to $\mathbb{P}_0$, the inputs w, $w^{(i)}$ and $v^{(i)}$, the counters a, b, i and j, the random bits r and the configuration C.

Each of the data objects above is polynomial in size, and there are at most a polynomial number of them at any given time. Thus, P_{opt} operates in polynomial space, and we are done. □

Combining Lemma 6, Lemma 7 and Lemma 8, we obtain the desired result.

Proposition 3. *IP* ⊂ *PSPACE*

4 Variants and Extensions

We now discuss some variants of the class *IP*, and show how they relate to it and other important complexity classes.

We start with two apparent weakenings of the class *IP*, which turn out not to diminish its power.

Definition 6. *Let (P, V) be an interactive proof system that accepts the language L. We say (P, V)* has perfect completeness *if for all $w \in L$, the probability of the system accepting w is* 1. *We say that the system has* public coins, *if V sends the random bits he reads to P.*

In systems with perfect completeness, the verifier must always accept words in the language. Note that in a system with public coins, P can of course deduce which messages are random bits and which are other messages, because he is omnipotent.

Proposition 4. *Every language $L \in IP$ is accepted by an interactive proof system (P, V) with perfect completeness and public coins, where V runs in polynomial time.*

Proof. Since $IP \subset PSPACE$, there is a polynomial time reduction of L to the *PSPACE*-complete language QBF. Furthermore, QBF is accepted by a system with perfect completeness and public coins because the protocol in the proof of Proposition 2 has both properties. □

Conversely, whenever *less* information is sent to the prover in the model, it may be become more powerful: the verifier now has the choice whether to send the information or not. For example, we could change the definition of *IP* so that the input word is not sent to the prover.

Lemma 9. *If L has an IP protocol where the prover cannot see the input word, then it has an IP protocol.*

Proof. We can think of the usual definition of *IP* as

$$L \in IP \iff (\exists V)(\forall w)(\exists P)(\forall P')\phi(V, w, P, P', L),$$

where ϕ is a statement about probabilities in the cases $w \in L$ or $w \notin L$, and P and P' are quantified over functions from messaging histories to responses. Suppose first that the honest prover cannot see the input word but the dishonest one can, so that we obtain the definition

$$L \in IP_1 \iff (\exists V)(\exists P)(\forall w)(\forall P')\phi(V, w, P, P', L).$$

Now, P' in essense sees the input anyway, since *in particular* P' takes the value of the most evil oracle for the input $w \notin L$. Thus, an optimal V could just as well send w to the prover, so that these models are equivalent. Since $\forall$-quantifiers commute, the definition

$$L \in IP_2 \iff (\exists V)(\exists P)(\forall P')(\forall w)\phi(V, w, P, P', L)$$

is also equivalent. But this is just the model of *IP* where the provers do not see the input. □

Next, we consider the extensions of *IP* involving multiple provers.

Definition 7. *An* interactive proof system with multiple provers *is an $(n+1)$-tuple $(P_1, \ldots, P_n, V)$, where the P_i are provers and V is a polynomial time verifier with n communication tapes, one for each prover. Each prover can only see w and the requests V sends to it, not the requests sent to the other provers (but it can remember what it has seen sofar). A language L is* accepted *by $(P_1, \ldots, P_n, V)$ if*

- *for all $w \in L$, the probability that V accepts w using the provers $P_1, \ldots, P_n$ is at least $\frac{3}{4}$, and*

- *for all $w \notin L$ and all provers $P'_1, \ldots, P'_n$, the probability that V accepts w using the provers $P'_1, \ldots, P'_n$ is at most $\frac{1}{4}$.*

For all $n \geq 1$, we denote by MIP_n the class of languages accepted by interactive proof systems with n provers.

In particular, $MIP_1 = IP$ by definition. The presence of more than one prover makes the system more powerful, as the verifier can cross-check the answers of one verifier against those of the others. We present the following result without proof.

Theorem 2 ([1])**.** *For all $n \geq 2$, we have $MIP_n = NEXPTIME$.*

Thus, if multiple honest provers are required for acceptance, the system possesses immense computational power. But what if there are two provers, only one of which is known to be honest? This situation resembles the classic puzzle of two gatekeepers, one of which always speaks the truth, while the other always lies, and the task is to deduce which is which.

Definition 8. *An* interactive proof system with unknown identities *is a tuple (P, V), where P is a prover function and V is a polynomial time verifier with two communication tapes. A language L is* accepted *by (P, V) if*

(i) for each $w \in L$ and all provers P', the probability that V accepts w using the provers P and P' in any *order is at least $\frac{3}{4}$, and*

(ii) for each $w \notin L$ and all provers P', P'', the probability that V accepts w using the provers P', P'' is at most $\frac{1}{4}$.

The language L is symmetrically accepted *by (P, V) if item (ii) is replaced by*

(ii') for each $w \notin L$ and all provers P', the probability that V accepts w using the provers P and P' in any *order is at most $\frac{1}{4}$.*

We denote by UIP (SIP) the class of languages accepted (symmetrically accepted, respectively) by systems with unknown identities.

The prover P represents the truth-telling gatekeeper, while P' and P'' can answer whatever they wish. We can easily deduce at least the following.

Proposition 5. *We have $IP = UIP$ and $IP \subset SIP$.*

Proof. Let $L \in IP$ using the interactive proof system (P, V) with perfect completeness (which exists by Proposition 4), and construct the system with unknown identities (P, V') as follows. The verifier V' runs the protocol of V twice for each prover, and accepts if both runs on either prover result in acceptance. Since (V, P) has perfect completeness, for every prover P' and input $w \in L$, the verifier V' always accepts w using the provers P and P' in any order. For any two provers P' and P'' and an input $w \notin L$, the probability of two consecutive accepts by V using either of them is at most $\frac{1}{16}$, so the total probability of acceptance is at most $\frac{1}{8}$. This shows that $L \in UIP$ and $L \in SIP$.

Suppose then that $L \in UIP$ using the system (P, V) with unknown identities, and construct the IP system (P', V') as follows. The system (P', V') simulates the verifier V whose first communication tape is connected to P and the second to P_0, the trivial prover that always answers with 0. The verifier V' simulates V step by step, and if V would send a request, then V' sends it to P' together with a single bit indicating whether it was meant for P or P_0. The prover P' then answers the request accordingly. It is clear that L is accepted by the system (P', V'), and thus $L \in IP$. □

There is also an alternative (and perhaps more natural) proof for $PSPACE \subset SIP$, found in [2], where this class was first defined (but not explicitly named). The result is also stronger, in that the verifier can be assumed deterministic. Let G be a determined two-player game of polynomial length for which the problem of finding a winning strategy is $PSPACE$-complete, and consider the following SIP protocol for it. First, the verifier asks the provers whether the first player has a winning strategy in G. If the provers' answers are equal, the verifier believes them, and otherwise, it lets the provers take turns in playing the game against each other, using the winning strategies they claimed to have. The eventually winning prover must be the honest one.

Finding upper bounds for SIP is less obvious, and the following result from [3] gives an exponential time bound. The exact characterization of SIP in terms of space or time complexity seems to remain an open problem.

Theorem 3. $SIP \subset EXPTIME$

References

[1] László Babai, Lance Fortnow, and Carsten Lund. Nondeterministic exponential time has two-prover interactive protocols. *Comput. Complexity*, 1(1):3–40, 1991.

[2] U. Feige, A. Shamir, and M. Tennenholtz. The noisy oracle problem. In *Advances in cryptology—CRYPTO '88 (Santa Barbara, CA, 1988)*, volume 403 of *Lecture Notes in Comput. Sci.*, pages 284–296. Springer, Berlin, 1990.

[3] Joan Feigenbaum, Daphne Koller, and Peter Shor. A game-theoretic classification of interactive complexity classes. In *In Proceedings of the Tenth Annual IEEE Conference on Computational Complexity*, pages 227–237, 1995.

[4] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[5] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, pages 59–68, New York, NY, USA, 1986. ACM.

[6] Dexter C. Kozen. *Theory of computation*. Texts in Computer Science. Springer-Verlag London Ltd., London, 2006.

[7] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. Assoc. Comput. Mach.*, 39(4):859–868, 1992.