On IP = PSPACE and Theorems with Narrow Proofs[*]

Juris Hartmanis Richard Chang[†] Desh Ranjan[‡]
Pankaj Rohatgi

Department of Computer Science, Cornell University
Ithaca, New York 14853, USA

Abstract

It has been shown that the class of languages with interactive proofs, IP, is exactly the class PSPACE. This surprising result elegantly places IP in the standard classification of feasible computations. Furthermore, the IP = PSPACE result reveals some very interesting and unsuspected properties of mathematical proofs.

In this column we define the *width* of a proof in a formal system $\mathcal{F}$ and show that it is an intuitively satisfying and robust definition. Then, using the IP = PSPACE result, it is seen that the width of a proof (as opposed to the length) determines how quickly one can give overwhelming evidence that a theorem is provable without showing the full proof.

1 On Proofs and Interactive Proofs

A mathematician has the most confidence in the truth of a theorem when he/she is given a complete proof of the theorem in a trusted formal system. Let $\mathcal{F}$ be such a formal system in which the correctness of a proof can be checked by a verifier in polynomial time. The class NP clearly captures all the theorems which have polynomially long proofs. The NP $=?$ P question is the question about the quantitative computational difference between finding a proof of a theorem and checking the correctness of a given proof.

Some years ago, theoretical computer scientists asked whether it is possible to give convincing evidence that a theorem is provable in $\mathcal{F}$ without showing a complete

[*]This research was supported in part by NSF Research Grant CCR 88-23053.

[†]Supported in part by an IBM Graduate Fellowship. Current Address: Department of Computer Science, University of Maryland, Baltimore County Campus, Baltimore, MD 21228, USA.

[‡]Current Address: Max Plank Institut für Informatik, Im Stadtwald, W 6600 Saarbrücken, Germany. On leave from Department of Computer Science, New Mexico State University, Las Cruces, NM 88003, USA.

proof. Clearly, if we do not give a complete proof to a verifier (that does not have the power or time to generate and check the proof), then we cannot expect the verifier to be completely convinced that the theorem is provable. This led to a very fascinating problem: how can a verifier be convinced with high probability that a given theorem is provable without seeing the whole proof? and how rapidly can this be done?

This problem has been formulated and extensively studied in terms of *interactive protocols* [Gol89]. Informally, an interactive protocol consists of a *Prover* and a *Verifier*. The Prover is an all powerful Turing Machine (TM) and the Verifier is a TM which operates in time polynomial in the length of the input. In addition, the Verifier has a random source (e.g., a fair coin) not visible to the Prover. In the beginning of the interactive protocol the Prover and the Verifier receive the same input string. Then, the Prover tries to convince the Verifier, through a series of queries and answers, that the input string belongs to a given language. The Prover succeeds if the Verifier accepts with probability greater than 2/3. The probability is computed over all possible coin tosses made by the Verifier. However, the Verifier must guard against imposters masquerading as the real Prover. That is, the Verifier must not be convinced to accept a string not in the language with probability greater than 1/3—even if the Prover lies.

Definition Let V be a probabilistic polynomial time TM and let P be an arbitrary TM. P and V share the same input tape and communicate via a communication tape. P and V form an interactive protocol for a language L if

1. $x \in L \Longrightarrow$ Prob[P-V accepts x] $> 2/3$.
2. $x \notin L \Longrightarrow \forall P^*$, Prob[P^*-V accepts x] $< 1/3$.

A language L is in IP if there exist P and V which form an interactive protocol for L.

Clearly, IP contains all NP languages, because in polynomial time the Prover can give the Verifier the entire proof. In such a protocol, the Verifier cannot be fooled and never accepts a string not in the language. To illustrate how randomness can generalize the concept of a proof, we look at an interactive protocol for a language not known to be in NP. Consider GNI, the set of pairs of graphs that are not isomorphic. GNI is known to be in co-NP and believed not to be in NP. However, GNI does have an interactive protocol [GMW86]. For small graphs, the Verifier can easily determine if the two graphs are not isomorphic. For sufficiently large graphs, the Verifier solicits help from the Prover to show that G_i and G_j are not isomorphic, as follows:

1. The Verifier randomly selects G_i or G_j and a random permutation of the selected graph. This process is independently repeated n times, where n is the number of vertices in G_j. If the graphs do not have the same number of vertices, they are clearly not isomorphic. This sequence of n randomly chosen, randomly permuted graphs is sent to the Prover. Recall that the Prover has not seen

the Verifier's random bits. This assumption is not necessary, but simplifies the exposition.

2. The Verifier asks the Prover to determine, for each graph in the sequence, which graph, G_i or G_j, was the one selected. If the Prover answers all the queries correctly, then the Verifier accepts.

Suppose the two original graphs are not isomorphic. Then, only one of the original graphs is isomorphic to the permuted graph. The Prover simply answers by picking that graph. If the graphs are isomorphic, then the Prover has at best a 2^{-n} chance of answering all n questions correctly. Thus, the Verifier cannot be fooled with high probability. Therefore, GNI $\in$ IP.

Note that GNI is believed to be incomplete for co-NP. So, the preceding discussion does not show that co-NP $\subseteq$ IP. For a while, it was believed that co-NP is not contained in IP, because there are oracle worlds where co-NP $\not\subseteq$ IP [FS88]. In fact, the computational power of interactive protocols was not fully appreciated until Lund, Fortnow, Karloff and Nisan [LFKN90] showed that IP actually contains the entire Polynomial Hierarchy. This result then led Shamir [Sha90] to completely characterize IP by showing that

$$\mathrm{IP} = \mathrm{PSPACE}.$$

Babai, Fortnow and Lund [BFL90] characterized the computational power of multi-prover interactive protocols

$$\mathrm{MIP} = \mathrm{NEXP}.$$

In both cases, it is interesting to see that interactive proof systems provide alternative definitions of classic complexity classes. Thus, they fit very nicely in the overall classification of feasible computations. Furthermore, both of these problems have contradictory relativizations [FS88]. That is, there exist oracles A and B such that

$$\mathrm{IP}^{\mathrm{A}} = \mathrm{PSPACE}^{A} \quad \text{and} \quad \mathrm{IP}^{\mathrm{B}} \neq \mathrm{PSPACE}^{B},$$

and similarly for the multi-prover case. Thus, these results provide the first *natural* counterexamples to the belief that problems with contradictory relativizations are beyond our proof techniques. The IP = PSPACE result also provides a very dramatic counterexample to the already battered Random Oracle Hypothesis. In [HCRR90], we showed that

$$\mathrm{Prob}_A[\ \mathrm{IP}^{\mathrm{A}} \neq \mathrm{PSPACE}^{A}\ ] = 1.$$

2 On Theorems with Polynomially Wide Proofs

In this section, we define the notion of the *width* of a proof of a theorem in a formal system. The notion of "formal system" goes back to Hilbert who wanted to develop a complete system to formalize all of mathematics. There are several equivalent ways

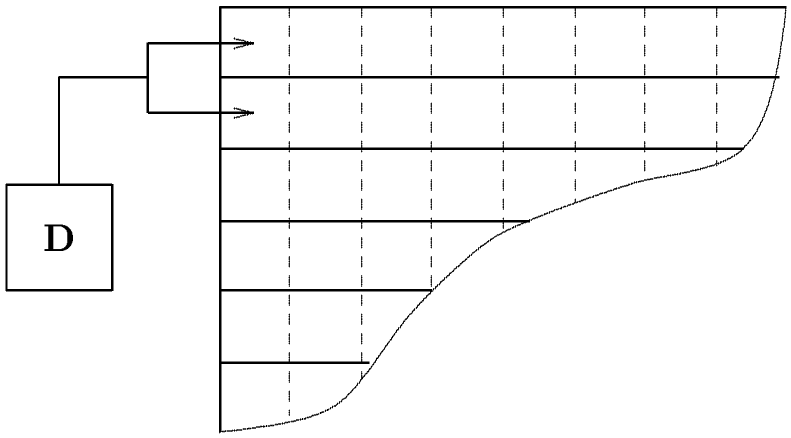

Figure 1: A DFA proof checker reading adjacent lines of the proof.

of defining what one means by "formal system." The most common one says that a formal system is a set of axioms and a set of recursive rules for obtaining new statements from the axioms. All the axioms and the statements obtained by the repeated application of these rules are the theorems of the formal system. The proof of a theorem is the documentation of how the theorem was derive using the systematic application of rules to the axioms and other theorems. The notion of theorems and proofs is central to the concept of a formal system. The idea behind restricting the rules to be recursive is that one wants the proof-checking to be recursive. Although the rules are only restricted to be recursive, they are expected to be fairly simple so the proofs of the theorems can be checked and understood with ease. In what follows we give an equivalent definition of "formal system" which renders proof-checking very simple. This formal system is better suited for defining measures of computational complexity on proofs in a formal system. (This is analogous to the fact that Turing machines are better suited for the study of computational complexity than the Lambda Calculus, even though both are models of computable functions.)

We define a formal system to be a set of proofs and a corresponding set of theorems. The key concept in the definition of "formal system" is how a proof is presented and verified. In our first definition of "formal system" we insist that the proof must be presented as a sequence of horizontal lines written on a two dimensional page. Each line is a string over a fixed alphabet and is written in a left-justified form directly below the preceding line. Moreover, we insist that the proof can be checked by a deterministic finite automaton (DFA) in the following way. The DFA proof checker starts at the top-left corner of the page and reads two adjacent symbols at a time from the first two lines of the proof. The DFA scans the input in an oblivious manner—it reads the first two lines, scanning from left to right, and returns to the left margin to read the second and third lines, left to right, and so forth. (See Figure 1.) When the DFA reaches the bottom right corner, it accepts or rejects. The proofs of the formal system are exactly those sequences of lines accepted by the DFA proof checker, and the theorems of the formal system are the first lines of the proofs. That is, we assume that all proofs start with the statement of a theorem and end with "Q.E.D."

Let $\mathcal{F}$ be a formal system and $D_{\mathcal{F}}$ be the corresponding proof checker, as described above.

Definition The *width* of a proof in $\mathcal{F}$ is the width of the longest line. The *width* of a theorem T in $\mathcal{F}$, written $width(D_{\mathcal{F}}, T)$, is the width of the narrowest proof of T in $\mathcal{F}$.

Definition We say that a language L is in PWT, that is, L is a set of *polynomially-wide theorems*, if there exist a $k \geq 1$ and a formal system $\mathcal{F}$, such that

$$T \in L \iff T \text{ is a theorem in } \mathcal{F} \text{ and } width(D_{\mathcal{F}}, T) < |T|^k + k.$$

At first glance, our definition of "formal system" may seem too restrictive. After all, the DFA proof checker cannot even verify if the statement of the theorem has balanced parentheses. However, the DFA proof checker can verify a proof that the theorem has balanced parentheses. We claim that our definition of "formal system" is robust by showing that replacing the DFA proof checker with more powerful devices does not change the class PWT.

In our definition of a formal system with a DFA proof checker, we tried to capture the idea that it should be "easy" to check if one line of the proof follows another. This notion of "easy" can be extended to recognition in polynomial time. For example, let the verifier be a Turing machine with a separate work tape. The size of the work tape is bounded by a polynomial in the size of the theorem. The verifier still scans the proof obliviously, but now it can copy the lines of the proof onto its work tape and spend an additional amount of time (polynomial in the length of the line it is reading) to check the proof. After this allotted time, the verifier continues to read the proof obliviously. Let PWT_{poly} be defined as above, replacing the DFA proof checker by a polynomial time proof checker. We show that the more powerful proof checker does not change the class of polynomially wide theorems.

Theorem 1 $\mathrm{PWT} = \mathrm{PWT}_{poly}$.

Proof: Clearly, $\mathrm{PWT} \subseteq \mathrm{PWT}_{poly}$. To see that $\mathrm{PWT}_{poly} \subseteq \mathrm{PWT}$, let $L \in \mathrm{PWT}_{poly}$. Then, there is a formal system $\mathcal{F}$ and a Turing machine $M_{\mathcal{F}}$ such that

$$T \in L \iff T \text{ is a theorem in } \mathcal{F} \text{ and } width(M_{\mathcal{F}}, T) < |T|^k + k.$$

We can construct a new formal system $\mathcal{F}'$ where the proofs can be verified by an oblivious finite automaton. To do this, simply annotate the proofs in $\mathcal{F}$ by inserting, between the lines, the instantaneous descriptions (ID) of the computation of $M_{\mathcal{F}}$.

Since the ID's change only near the tape heads, a finite automaton that has $M_{\mathcal{F}}$'s transition table can verify that $M_{\mathcal{F}}$ accepted the proof. Thus, $L \in \mathrm{PWT}$. $\square$

Even the concept of explicit lines are not essential in these definitions. Consider a formal system whose proofs are presented as a single line and the proof checker is a two-headed finite automaton, scanning the proof from left to right. Here the width of the proof is the maximum separation of the heads. Define PWT_{1-line} to be the class of languages containing polynomially wide theorems under this new definition of width. Again, this model yields the same class PWT.

Theorem 2 $\mathrm{PWT} = \mathrm{PWT}_{1-line}$.

Proof: Let $L \in \mathrm{PWT}$. Let $\mathcal{F}$ and $D_{\mathcal{F}}$ be the formal system and the proof checker for the language L. To give a one-dimensional proof that $T \in L$, simply take the two-dimensional proof and concatenate the lines to form one long line. A DFA D' with two heads can simulate $D_{\mathcal{F}}$ reading two adjacent symbols of the two dimensional proof by keeping the two heads in the correct positions along the single-line proof. Since $D_{\mathcal{F}}$ reads symbols only from adjacent lines, D' can simulate $D_{\mathcal{F}}$ without moving its heads further apart than the width of the original proof. So, $L \in \mathrm{PWT}_{1-line}$.

On the other hand, if $L \in \mathrm{PWT}_{1-line}$, then for any $T \in L$, let w be the width of the proof of T. To give a two-dimensional proof that $T \in L$, simply divide the single-line proof into segments of length w and arrange the segments into a two-dimensional proof by listing the segments one below the other. A polynomial time proof checker M can simulate the two-headed DFA, D, reading a single-line proof, because M can record the last two lines of the proof in its work tape. Since D never moves its heads greater than w tape cells apart, M only needs the last two lines of the two-dimensional proof to simulate D. Thus, $\mathrm{PWT}_{1-line} \subseteq \mathrm{PWT}_{poly} \subseteq \mathrm{PWT}$. $\square$

Finally, in both models with finite automata verifiers, we can allow the verifier to have freer movements. In the two dimensional case, we can remove the restriction that the verifier must read the proof in an oblivious manner and allow it to move up, down, left or right at any time. Define $\mathrm{PWT}_{\updownarrow\leftrightarrow}$ to be the class of languages containing polynomially wide theorems under this non-oblivious DFA verifier model.

Theorem 3 $\mathrm{PWT} = \mathrm{PWT}_{\updownarrow\leftrightarrow}$.

Proof: Clearly, $\mathrm{PWT} \subseteq \mathrm{PWT}_{\updownarrow\leftrightarrow}$. To see that $\mathrm{PWT}_{\updownarrow\leftrightarrow} \subseteq \mathrm{PWT}$, note that the non-oblivious DFA proof checker is deterministic, so it cannot be in the same state when it revisits a location in the two dimensional page. If it does, it will loop and never accept the proof. Since the number of states is constant, the DFA can only visit each location a constant number of times. Thus, to convert a proof for a non-oblivious DFA verifier into one that an oblivious DFA proof verifier can check, one simply has to include, at each location on the page, a list describing state and direction of head movement of the non-oblivious DFA during each visit to that location. The oblivious DFA proof checker can verify that the list at each location is consistent with the lists at neighboring locations, because there is only a constant number of possible lists. It can also verify that the non-oblivious DFA entered the proof at the top left corner, and accepted in the bottom right corner. Hence, $\mathrm{PWT}_{\updownarrow\!\!\!\!\leftrightarrow} \subseteq \mathrm{PWT}$. $\square$

Similarly, for the formal system where the proof is presented in one line and the verifier is a two-headed finite automaton, we can allow both heads to move left and right. Define $\mathrm{PWT}_{\leftrightarrow}$ to be the class of languages containing polynomially wide theorems under this new definition of width.

Theorem 4 $\mathrm{PWT} = \mathrm{PWT}_{\leftrightarrow}$.

Proof: By Theorem 2, $\mathrm{PWT} \subseteq \mathrm{PWT}_{1-line} \subseteq \mathrm{PWT}_{\leftrightarrow}$. For any language L in $\mathrm{PWT}_{\leftrightarrow}$, let $\mathcal{F}$ and $D_{\mathcal{F}}$ be the formal system and the two-headed DFA proof checker for $\mathcal{F}$. Suppose $D_{\mathcal{F}}$ is checking if a string x is a proof in $\mathcal{F}$. Let w be the greatest distance separating the two tape heads of $D_{\mathcal{F}}$ while verifying x. A deterministic Turing machine M with one input head and a separate work tape can simulate $D_{\mathcal{F}}$ using no more than $\log w$ tape cells on its work tape. M will simply use the work tape to keep track of the distance between the two tape heads of $D_{\mathcal{F}}$. Since M has only polynomially many work tape configurations, it can only visit each symbol of x polynomially many times without looping. Then, as in the proofs of Theorems 2 and 3, we can convert the one line proof into a two dimensional proof and annotate the proof with M's crossing sequence. This will change the width of the proof by at most a polynomial. Also, since the depth of each crossing sequence is polynomial, an oblivious polynomial time proof checker can verify that the crossing sequence at each location is consistent with the crossing sequences of its neighbors. Hence, $\mathrm{PWT}_{\leftrightarrow} \subseteq \mathrm{PWT}_{poly} \subseteq \mathrm{PWT}$. $\square$

As the reader may have suspected, there is a good reason why all these classes turn out to be PWT. The reason is that the verifiers in these formal systems have enough power to check valid computations. Hence, restricting the width of the proof to be polynomial in the size of the theorem is equivalent to restricting the size of the instantaneous descriptions of a Turing machine computation to be polynomial in the size of the input string. But, this is just PSPACE. (See [Fis69] for related results.)

Theorem 5 $\mathrm{PWT} = \mathrm{PSPACE}$.

Proof: Let $L \in \mathrm{PWT}$. Then, using our first definition of "formal system", there exist a constant k and a formal system with a DFA proof checker, $D_{\mathcal{F}}$, such that

$$T \in L \iff T \text{ is a theorem in } \mathcal{F} \text{ and } width(D_{\mathcal{F}}, T) < |T|^k + k.$$

On input T, a nondeterministic Turing machine M can guess a width m, guess successive lines of length m, and use $D_{\mathcal{F}}$ to check if the lines form a proof of T in $\mathcal{F}$. If such a proof is found, M accepts. Clearly,

$$T \in L \iff M(T) \text{ accepts.}$$

Since NPSPACE $\subseteq$ PSPACE, $L \in$ PSPACE.

Conversely, suppose that $L \in$ PSPACE. Then, there is a Turing machine M which recognizes L. We can define a formal proof system that uses the instantaneous descriptions (ID) of M as the lines of a proof. Since the ID's only change near the tape head, a DFA can check if one ID follows another according to the transition table of M. Moreover, the length of the ID's are polynomial in the size of the input string T, so the width of the proof is polynomial in $|T|$. Thus, there is a formal system which proves exactly the strings in L using polynomially-wide proofs. So,

$$\text{PWT} = \text{PSPACE}.$$

$\square$

Note that the role of *polynomial* width was not essential in proving these results. We can easily define Exponentially Wide Theorems and show similar robustness properties. These observations show that the definition of the width of a proof is robust, and we believe it properly captures the intuitive idea of width. In essence, the width of a proof quantifies how much information a verifier needs check a proof and how far this information can be separated in the proof.

3 Conclusion

From the previous consideration, we see that

$$\text{IP} = \text{PSPACE} = \text{PWT}.$$

Recall that NP is the class of languages with polynomially long proofs. PWT is the class of languages with polynomially wide proofs whose length may not be polynomially bounded. By comparing IP and PWT we can see how interaction and probabilistic acceptance can shorten the amount of time required to check a proof.

For example, if NP $\neq$ PSPACE, then there are formal systems where all of the theorems have polynomial width, but some of the proofs have lengths that are not bounded by any polynomial (in the size of the theorem). Thus, in these formal systems the proof cannot always be presented in polynomial time. However, one can present a probabilistic proof in polynomial time, because IP = PSPACE. If NP $\neq$ PSPACE, then the whole range of languages in PSPACE $-$ NP are of this type.

Note that the languages in PSPACE may require exponentially long proofs. So if PSPACE $\neq$ NEXP, then there are languages which have exponentially long proofs

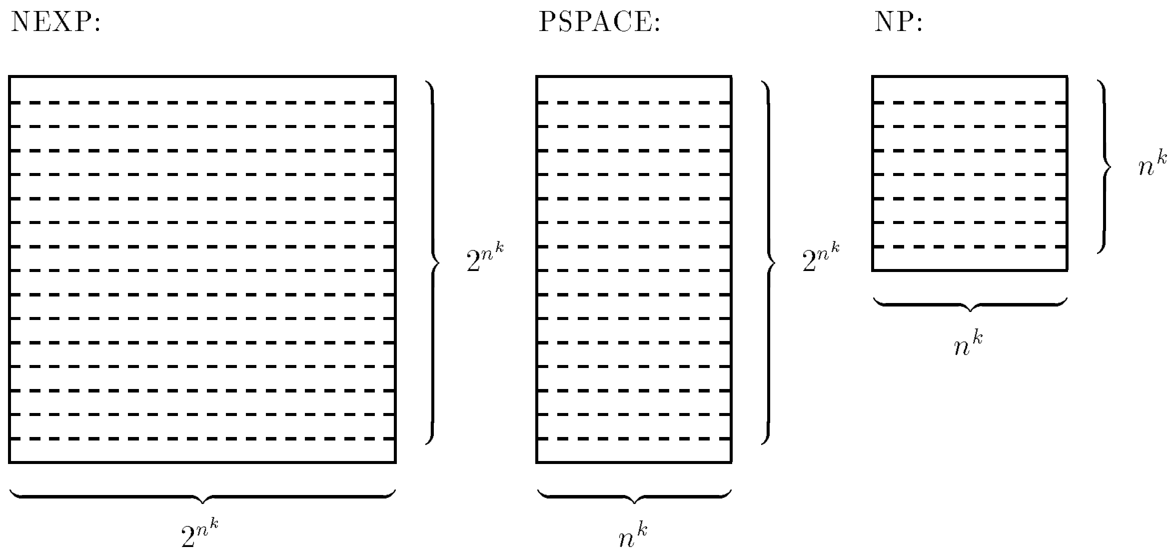

Figure 2: The shapes of proofs for languages in NEXP, PSPACE and NP.

that do not have (proof systems) with polynomially wide proofs. Figure 2 illustrates the various shapes of these proofs.

We believe that these results, in particular the IP = PSPACE result in the IP = PWT interpretation, reveal new and interesting insights about the quantitative nature of mathematical proofs and thus about the fundamental nature of mathematics. The two dimensional shape of the proof determines how rapidly one can give overwhelming evidence that there is a proof without showing the whole proof. Intuitively, this is because the width of the proof reflects the maximum distance that separates the critical pieces of the proof which are used to verify the correctness of the proof.

We are impressed by the rapid development of the theory of interactive proofs and delighted by the research culminating in the IP = PSPACE result. IP = PSPACE is indeed a beautiful result contributing to the elegance and relevance of complexity theory and leading to new insights in the quantitative nature of mathematics.

References

[BFL90] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 16–25, 1990.

[Fis69] M. J. Fischer. Two characterizations of the context sensitive languages. In *IEEE Conference Record of the Symposium on Switching and Automata Theory*, pages 149–156, 1969.

[FS88] L. Fortnow and M. Sipser. Are there interactive protocols for co-NP languages? *Information Processing Letters*, 28(5):249–251, August 1988.

[GMW86] O. Goldreich, S. Micali, and A. Widgerson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 174–187, 1986.

[Gol89] S. Goldwasser. Interactive proof systems. In *Computational Complexity Theory*, Proceedings of Symposia in Applied Mathematics, Volume 38, pages 108–128. American Mathematical Society, 1989.

[HCRR90] J. Hartmanis, R. Chang, D. Ranjan, and P. Rohatgi. Structural complexity theory: Recent surprises. In *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory*, volume 447 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 1990.

[LFKN90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 2–10, 1990.

[Sha90] A. Shamir. IP = PSPACE. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 11–15, 1990.