Interactive Proof Systems with Polynomially Bounded Strategies *

Anne Condon †
Dept. of Computer Science
University of Wisconsin
Madison, Wisconsin 53706
condon@cs.wisc.edu

Richard Ladner‡
Dept. of Comp. Sci. and Eng.
University of Washington
Seattle, Washington 98195
ladner@cs.washington.edu

February 26, 1992

Abstract

Interactive proof systems in which the Prover is restricted to have a polynomial size strategy are investigated. The restriction of polynomial size computation tree, visible to the Prover, or logarithmically bounded number of coin flips by the Verifier guarantee a polynomial size strategy. The additional restriction of logarithmic space is also investigated. A main result of the paper is that interactive proof systems in which the Prover is restricted to a polynomial size strategy are equivalent to MA, Merlin-Arthur games, defined by Babai and Moran [3]. Polynomial tree size is also equivalent to MA, but when logarithmic space is added as a restriction, the power of polynomial tree size reduces to NP. Logarithmically bounded number of coin flips are equivalent to NP, and when logarithmic space is added as a restriction, the power is not diminished. The proof that NP $\subseteq$ IP(log-space, log-random-bits) illustrates an interesting application of the new "fingerprinting" method of Lipton [18]. Public interactive proof systems which have polynomial size strategies are also investigated.

*Preliminary version of this paper appears in Seventh Annual Conference on Structure in Complexity Theory, June 1992

†Work supported by NSF, grant number CCR-9100886

‡Work supported by NSF, grant number CCR-9108314

1 Introduction

Interactive proof systems (IP) were introduced by Goldwasser, Micali, and Rackoff and independently by Babai as a formal model to investigate issues in cryptography and as a natural extension to NP, the class of problems solvable in nondeterministic polynomial time [16], [3]. The vision was that in an interactive proof system a *Prover* and a *Verifier* would interact in such way that the Prover could convince the Verifier with high probability of a *true* fact, but could only convince a Verifier with very low probability of a *false* fact. In this model, the Prover is all-powerful and the Verifier is limited to run in polynomial time. The Verifier has the additional ability to flip an unbiased coin.

The issue which we address in this paper is whether this original IP model can really serve as a practical basis for cryptography or as a practical extension of NP. Certainly, there are practical aspects of cryptography, but what do we mean by a practical extension of NP? How can we even say that NP is practical at all, given that it is still an open question whether or not there are polynomial algorithms for NP-complete problems? The class NP is a practical class in the sense that, given a language $L \in$ NP and a particular input $x \in L$ it is possible for a Prover to write down proof of membership of x in L which could be checked by a polynomial time bounded Verifier. Thus, for each $x \in L$ the Prover has a polynomial size *strategy* for convincing the Verifier of the fact that $x \in L$. In this case, the Prover's strategy consists of the polynomial length proof it would present to the Verifier. The practical aspect of NP is that the Prover is not all-powerful, but relies on a polynomial size strategy to convince the Verifier.

If one examines the early result of Goldreich, Micali, and Wigderson that graph nonisomorphism is in IP [15] one can intuitively see that the Prover, in proving that two graphs are not isomorphic, does not use a polynomial size strategy. More emphatically, the recent results of Lund, Fortnow, Karloff, and Nisan [19] and Shamir's striking result that IP = PSPACE [21], demonstrate that the all-powerful Prover is actually using a polynomial space computable strategy to convince the Verifier. There is no evidence that polynomial space computations are at all practical.

In this paper, we define restrictions which can be placed on both the Prover and the Verifier in order to make interactive proof systems potentially practical. By this we mean that our restrictions ensure that it is possible for the prover to write down a polynomial size strategy, which can be used to convince a verifier an input is in a language. Then, we prove a number of results about these restricted interactive proof systems. The principal restriction we consider is limiting the Prover to having a polynomial size strategy. The restriction to a polynomial size strategy is tantamount to saying that the interactive proof system could potentially be practically implemented. The formal definition of the *poly-strategy* restriction is given in Section 2.

If the interaction between Prover and Verifier can be represented by a polynomial size computation tree, visible to the Prover, then the Prover has a very simple polynomial size strategy. Alternatively, if the Verifier is limited to a logarithmic number of flips of a coin, then again the Prover has a simple polynomial size strategy. Thus, we define restrictions on interactive proof systems called *poly-tree-size* and *log-random-bits*, which are special cases of poly-strategy. If we restrict the Prover and Verifier to logarithmic space then the question arises as to whether a restricted Prover can convince such a limited Verifier of as much as it could when they each had no such space

limitation. It turns out that our most technically challenging results are about interactive proof systems where the Prover and Verifier limited to logarithmic space. We call this restriction *log-space*. Finally, interactive proof systems may be *public* or *private*, referring respectively to whether the Prover has full view of what the Verifier is doing or whether what the Verifier is doing can be partially hidden from the Prover. Although public and private interactive proof systems have been shown to be equivalent [17], this equivalence does not necessarily carry over to our restricted interactive proof systems.

1.1 Motivating Example

To motivate our study of restricted interactive proof systems, we use as an example the set 3-SAT, which is the set of satisfiable Boolean formulas in conjunctive normal form with at most three literals per clause. The traditional proof that 3-SAT is in NP shows that in polynomial time, a Prover can convince a Verifier that a formula is satisfiable, by simply listing the assignment of truth values to the variables. However, the Verifier needs linear space, in order to store this assignment. What if the Verifier has only $O(\log n)$ space? We show that in this case, the Prover can convince the Verifier with high probability that a formula is satisfiable, even if the Verifier has only $O(\log n)$ random bits.

Given a Boolean formula in 3 conjunctive normal form as input, the Prover sends information to the Verifier in two phases. In the first phase, the Prover sends an assignment of a truth value to each instance of a literal in the formula ordered by variable number. Thus, in the first phase all instances of a variable and its complement are sent before instances of the next variable and its complement are sent. In the second phase, the Prover sends the assignment to each instance of a literal in the formula, in the order in which the literals appear in the formula.

It may help to give a concrete example to explain what the Prover does. Consider the formula:

$$(x \vee y \vee z) \wedge (\bar{w} \vee \bar{y} \vee z) \wedge (w \vee \bar{x} \vee \bar{z}).$$

We represent the assignment of a truth value to an instance of a literal as a triple (l, i, v) where l is the literal, i is the number of the clause it appears in, and $v \in \{T, F\}$ is the assigned value. In the first phase the Prover sends the sequence:

$$(\bar{w}, 2, F), (w, 3, T), (x, 1, T), (\bar{x}, 3, F), (y, 1, F), (\bar{y}, 2, T), (z, 1, F), (z, 2, F), (\bar{z}, 3, T).$$

In the second phase the Prover sends the sequence:

$$(x, 1, T), (y, 1, F), (z, 1, F), (\bar{w}, 2, F), (\bar{y}, 2, T), (z, 2, F), (w, 3, T), (\bar{x}, 3, F), (\bar{z}, 3, T).$$

These sequences are sent by the Prover and received by the Verifier one character at a time, assuming a standard encoding of the sequences into a finite alphabet.

The Verifier does three checks. First, during the first phase of the Prover's protocol the verifier can check if each instance of a literal is given the same value and whether the complement of a literal is given the complementary value. Second, during the second phase of the Prover's protocol, the Verifier checks that each clause of the formula is satisfied. These first two checks can be done

by the Verifier deterministically using only $O(\log n)$ space. What remains to be checked is whether or not the information sent in the first phase is consistent with the information sent in the second phase. This third check is done in $O(\log n)$ space with $O(\log n)$ random bits, using a technique of Lipton [18] called "fingerprinting of multisets", which we describe in Section 4. Roughly, the idea is to reduce the problem of checking consistency of information sent in the two phases to that of checking if two multisets are equal. A short random fingerprint of each multiset is computed by the Verifier for this purpose, so that the multisets need not be stored explicitly. As the Verifier fingerprints the first multiset it is very important that the Prover not see what the Verifier is doing. If the Prover could see the fingerprint of the first multiset then it could potentially send a different multiset with the same fingerprint, thereby fooling the Verifier into believing two distinct multisets were equal. Thus, this interactive proof system is private and not public.

This interactive proof system is an example of one where Prover and Verifier are severely restricted, yet an NP-complete problem can still be proved and verified.

1.2 Related Work

Interactive proof systems with the log-space restriction have been studied in detail although not with the additional restrictions considered in this paper. (See for example [9], [10], [12] or [13]). Condon [7] previously studied log-space, "one-way" interactive proof systems, in which the Verifier never writes on the communication cell. A one-way interactive proof system is a special case of a private interactive proof system with the poly-tree-size restriction.

Feige, Fiat and Shamir [11] introduced a notion of interactive proofs of knowledge, where the prover wishes to convince the verifier that it "knows some secret" S about an input x, that satisfies a polynomial time computable predicate $P(x, S)$. For example, if x is a Boolean formula, S might be an assignment to the variables of the formula and $P(x, S)$ is true if and only if S is a satisfying assignment for x. In their model, like ours, the power of the prover is restricted; in fact the prover is a probabilistic, polynomial time bounded Turing machine which has S on a private tape. In this sense, their model is similar to our model of a prover with a polynomial-size strategy, where the strategy corresponds to the secret S. However, there are important differences between the two models, stemming from the fact that our model is a language acceptor, whereas the model of Feige, Fiat and Shamir is an interactive proof system of knowledge, not a language acceptor.

In a different context, Blum and Kannan [4] studied restricted interactive proof systems, where the communication of the Verifier and Prover is restricted in the following way. The Verifier sends strings x to the Prover and receives the response $f(x)$, for some fixed function f. Although this limits the strategy of the Prover, it does not seem to be closely related to our restrictions of poly-strategy or poly-tree-size.

Interactive proof systems with the restriction log-random-bits have recently been studied by Arora and Safra [2] and Arora et al. [1]. They show that any language in NP has an interactive proof system in which the verifier uses $O(\log n)$ random bits and examines only $O(1)$ bits sent by the prover. Strong new results on the nonapproximability of NP-hard problems follow from this characterization of NP.

Our notion of polynomial tree size was inspired by the Ruzzo's notion of polynomial tree size for alternating Turing machines [20]. The adaptation of polynomial tree size to interactive proof

systems is new to the best of our knowledge.

1.3 New Results

In stating our results, we denote by IP(<restriction 1>,..., <restriction k>) the class of languages accepted by interactive proof systems with <restriction 1>,..., <restriction k>. Thus, our example above shows that 3-SAT is a member of the set IP(log-space, log-random-bits). We assume that our interactive proof systems are private unless they are actually specified as public. Thus, in the example IP(log-space, log-random-bits) we are referring to private interactive proof systems. If we want to indicate public interactive proof systems then we write IP(public, <restriction 1>,..., <restriction k>). Our main results are as follows.

(1) We first characterize the class of languages accepted by interactive proof systems in which the Prover is restricted to having a polynomial size strategy, but there is no restriction on the space used by Prover or Verifier. It turns out that this class is exactly equivalent to the class of languages MA, defined by Babai and Moran [3]. The class MA stands for the languages recognized by so-called Merlin-Arthur games. Thus, we have the fundamental result:

$$\text{IP(poly-strategy)} = \text{MA}.$$

It turns out that some additional restrictions do not remove any more power. Thus we prove:

$$\text{MA} = \text{IP(poly-strategy)} = \text{IP(public, poly-strategy)} = \text{IP(poly-tree-size)}.$$

On the other hand, we show that if we add more restrictions we reduce the power of interactive proof systems to NP.

$$\text{NP} = \text{IP(public, log-random-bits)} = \text{IP(public, poly-tree-size)} = \text{IP(log-random-bits)}.$$

The techniques to prove these results are fairly straightforward, but revealing. These results are proved in Section 3.

(2) We next consider private, log-space interactive proof systems. We show that with either the poly-tree-size or log-random-bits restriction, such interactive proof systems accept the class NP. The principal results of Section 4 are:

$$\text{NP} = \text{IP(log-space, poly-tree-size)} = \text{IP(log-space, log-random-bits)}.$$

Several of the proofs are notable. Lipton's fingerprinting of multisets [18] is used help show NP $\subseteq$ IP(log-space, log-random-bits). The proof that IP(log-space, poly-tree-size) $\subseteq$ NP requires a new and interesting technique of constructing and evaluating a computation tree of an interactive proof system.

(3) Finally, we consider public, log-space interactive proof systems. Our results show that again, with either the restriction poly-tree-size or log-random-bits, such interactive proof systems are equivalent. However, they are apparently much weaker than the corresponding private interactive proof systems, since the class of languages they accept is contained in LOGCFL, the class of languages which are logarithmic space reducible to context-free languages [22]. The principal results of Section 5 are:

$$\text{IP(public, log-space, poly-tree-size)} = \text{IP(public, log-space, log-random-bits)}, \text{ and}$$

$$\text{NLOG} \subseteq \text{IP(public, log-space, poly-tree-size)} \subseteq \text{LOGCFL}.$$

Again several of the proofs are notable. The result that IP(public, log-space, poly-tree-size) $\subseteq$ IP(public, log-space, log-random-bits) uses a general tree pruning technique which allows a computation tree to be pruned back without sacrificing much in the acceptance probability. The result that IP(public, log-space, poly-tree-size) $\subseteq$ LOGCFL uses the equivalence of LOGCFL to the class of languages accepted by auxiliary pushdown automata which use $O(\log n)$ storage and polynomial time [22].

Figure 1 describes the basic relationships between the restricted interactive proof systems defined in this paper. In addition the figure describes which of these classes are equivalent to MA and NP.

2 Definitions

Because we are interested in limiting the power of the Prover in an interactive proof system we adopt a definition which facilitates the analysis of a weakened Prover, but is equivalent to the standard definition [16] when the Prover is unrestricted. Our definition follows the definition of so-called "game automata" given by Condon and Ladner [8].

An *interactive proof system* is a pair (P, V) of communicating Turing machines, P for Prover and V for Verifier. P is a nondeterministic Turing machine and V is a probabilistic Turing machine. Both P and V have access to a common input and V has access to a fair random coin. P and V also share a communication tape in order to communicate with each other. A run of the interactive proof system begins with P computing, then V computing, then back to P, and so on alternating back and forth, until V halts and either accepts or rejects the input. During a run of (P, V), P has nondeterministic choices it can make. We can think of the particular choices it make as its execution of a *strategy*. Also, during a run, V has random choices it can make, but otherwise executes deterministically. Generally speaking, V is allowed to compute *privately*, that is, V has its own work tape which is not visible to P and the results of V's coin flips are also not visible to P. A *configuration* of (P, V) is the standard object which describes all the components of the interactive proof system at a particular moment in a computation. A *visible configuration* consists of that part of a configuration which is visible to P, that is, all but what is private to V.

We define a *history* to be a sequence of configurations, starting with the initial configuration, such that each configuration follows from the previous one by a move of (P, V). A *visible history* is a sequence of visible configurations which P could have observed up to a particular moment in a computation. For more details on these definitions, see [5] or [6]. Thus, P's strategy depends only on what has been visible to it, namely, a visible history. We say that P has a move in visible history h if the last visible configuration in the sequence h is one in which it is P's turn to move.

Unless otherwise stated we always assume that our interactive proof system runs in polynomial time as a function of the input length. Both the Prover and the Verifier are limited to polynomial time and the number of alternations is limited to polynomial. As such, any strategy of P can be represented as an exponential size object σ, which lists for every potential visible history in which

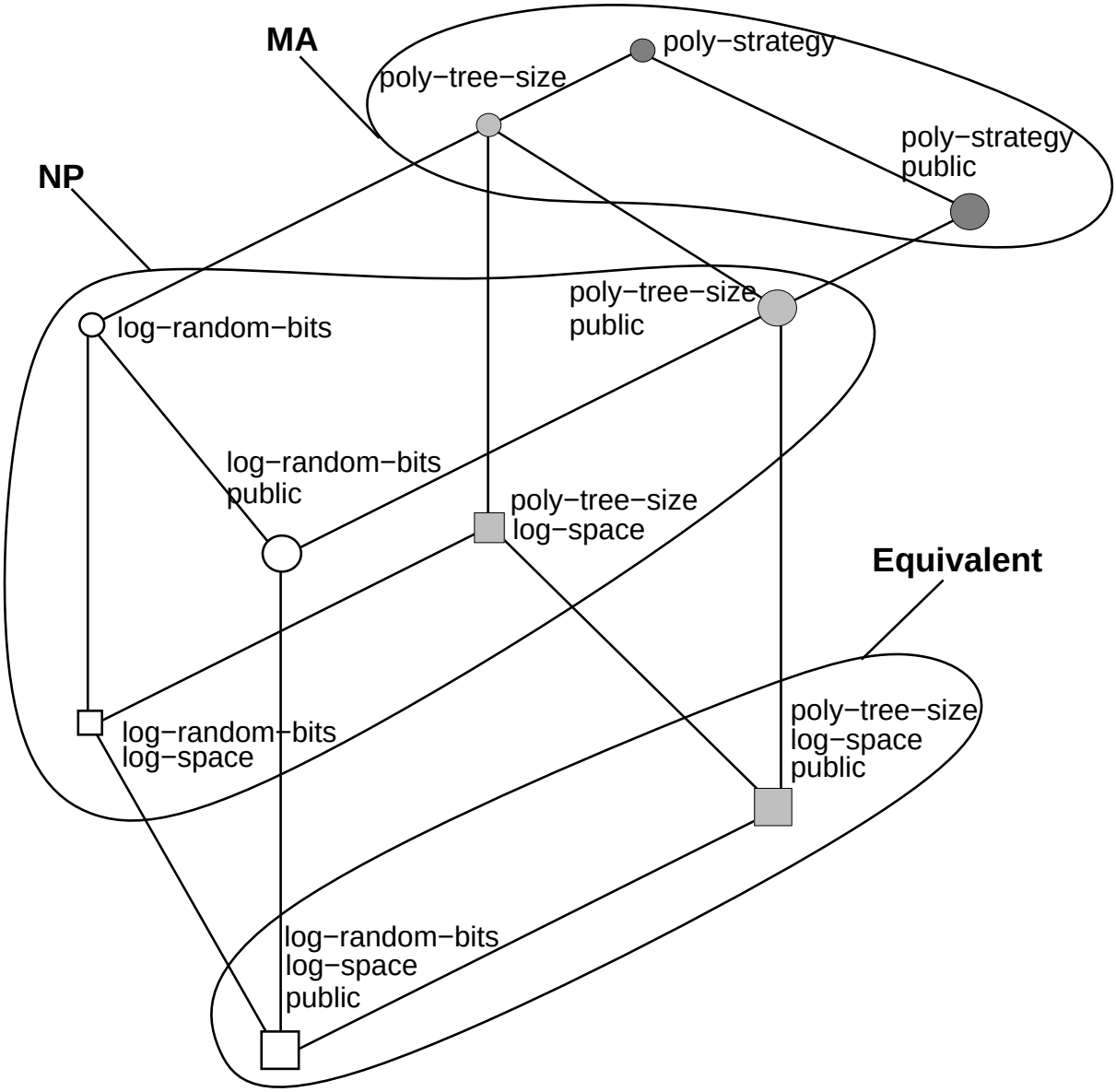

Figure 1: There are three dimensions in the restrictions with each dimension ranging from least restrictive to most restrictive: (i) poly-strategy – poly-tree-size – log-random bits (shaded from dark to light in the figure), (ii) private – public (small versus large), and (iii) unrestricted-space – log-space (circle versus square). The figure describes the relationships between the restricted interactive proof systems with the top-to-bottom direction indicating less restrictive to more restrictive. Equivalent classes are encircled.

P must move, exactly what nondeterministic choice P should make next. That is, σ is a list of pairs of the form (h, m) where h is a visible history and m is a move of P. Given a strategy σ, a computation tree $T(\sigma)$ can be constructed which records all the possible moves of V and the particular choices that P would make in reaction to these moves. Thus, the nodes of $T(\sigma)$ are labeled with configurations. Because V can move privately, P's choices must be identical for branches of $T(\sigma)$ with the same visible history. Given an input x and strategy σ, let (P, V, x, σ) be the probabilistic Turing machine which consists of P executing using the strategy σ and V running randomly on input x. We say that (P, V) accepts the language L if:

1. If $x \in L$ then for some σ, $\Pr[(P, V, x, \sigma) \text{ accepts}] \geq \frac{2}{3}$,
2. If $x \notin L$ then for all σ, $\Pr[(P, V, x, \sigma) \text{ accepts}] \leq \frac{1}{3}$.

We define IP to be the class of languages which are accepted by interactive proof systems. In the following we will be presenting a number of restrictions of interactive proof systems. The class of languages accepted by interactive proof systems with k of these restrictions is denoted by IP(<restriction 1>, <restriction 2>,..., <restriction k>).

2.1 Public Interactive Proof Systems

If the Verifier's random coin and work tapes are visible to the Prover then the interactive proof system (P, V) is said to be *public*. IP(public) is the class of languages accepted by public interactive proof systems. When the interactive proof system is public then a strategy of P is a slightly simpler object, although still potentially exponential in size. In particular, P's strategy can be assumed to be Markov, that is, the strategy need only depend on the current configuration of the interactive proof system, not on the entire visible history. Goldwasser and Sipser have shown that IP = IP(public) [17]. However, as we pointed out earlier, when other restrictions are included this equivalence may break down. We can state unequivocally that IP(public, <restriction 1>,...,<restriction k>) is always a subset of IP(<restriction 1>,...,<restriction k>).

2.2 Polynomial Size Strategy

Intuitively, if there is a polynomial p such that for any $x \in L$, the Prover's strategy can be expressed as a string τ with $|\tau| \leq p(|x|)$, then we say that (P, V) is *poly-strategy*. We can imagine that τ is written on an "advice tape" provided to P. Given a visible history and the advice τ, P can calculate in deterministic polynomial time what move it should make from the visible history to continue its strategy. More formally, we say that a polynomial time computable function f is a *strategy evaluator* if whenever f is provided a string τ and a visible history h, produces $f(\tau, h)$, a move for P from the visible history h. Given a strategy evaluator f and a string τ, define $\sigma(f, \tau)$ to be the strategy in which P always makes the move $f(\tau, h)$ in response to the visible history h. We say that $L \in$ IP(poly-strategy) if there is an interactive proof system (P, V), strategy evaluator f, and polynomial p with the properties:

1. If $x \in L$ then for some τ, $|\tau| \leq p(|x|)$ and $\Pr[(P, V, x, \sigma(f, \tau)) \text{ accepts}] \geq \frac{2}{3}$,

2. If $x \notin L$ then for all σ, $\Pr[(P, V, x, \sigma) \text{ accepts}] \leq \frac{1}{3}$.

Please note that if $x \notin L$ then we maintain the strong requirement that the Verifier accepts with probability $\leq \frac{1}{3}$ even if the Prover has an arbitrary strategy. On the other hand, if $x \in L$ then the Prover only needs a polynomial amount of information produce a strategy for which the Verifier will accept with probability $\geq \frac{2}{3}$.

The most common case of a polynomial size strategy is when the number of visible histories in which P must move is bounded by a polynomial. In this case τ is just a strategy of P and the strategy evaluator, f, applied to the argument (τ, h) simply looks up a pair (h, m) in the list τ and returns m. Thus, $\sigma(f, \tau) = \tau$.

2.3 Polynomial Tree Size

As noted earlier, given a strategy σ there is an associated computation tree $T(\sigma)$. This computation tree represents the entire computation by the Prover following strategy σ and the Verifier. In order to assess the Prover's view of the computation tree we coalesce nodes in the tree which represent the same visible history. The result is the computation tree as viewed by the Prover, which we call the *visible computation tree*, $PT(\sigma)$. Move formally, two nodes in $T(\sigma)$ are equivalent if the visible histories leading to the two nodes are identical. The nodes of $PT(\sigma)$ are the equivalence classes of nodes in $T(\sigma)$ under this equivalence relation. The tree structure of $PT(\sigma)$ is inherited from $T(\sigma)$ in the natural way. The label at a node of $PT(\sigma)$ is the visible configuration common to all the nodes in the equivalence class of nodes of $T(\sigma)$ which make up that node. We call the restriction when the visible computation tree $PT(\sigma)$ has polynomial size, the *poly-tree-size* restriction.

We say $L \in$ IP(poly-tree-size) if there is a polynomial p such that:

1. If $x \in L$ then for some σ, $|PT(\sigma)| \leq p(|x|)$ and $\Pr[(P, V, x, \sigma) \text{ accepts}] \geq \frac{2}{3}$,
2. If $x \notin L$ then for all σ, $\Pr[(P, V, x, \sigma) \text{ accepts}] \leq \frac{1}{3}$.

Even though $PT(\sigma)$ is of polynomial size, the computation tree $T(\sigma)$ may still be of exponential size. However, from P's point of view the computation tree appears to have polynomial size, since it only views a polynomial number of potential visible histories by following its strategy. In the case that (P, V) is a public interactive proof system, $PT(\sigma) = T(\sigma)$. Given a visible computation tree as the representation of a strategy, it is easy to calculate a Prover's move from the tree. Thus, we immediately have

(i) IP(poly-tree-size) $\subseteq$ IP(poly-strategy),

(ii) IP(public, poly-tree-size) $\subseteq$ IP(public, poly-strategy).

(We note that an alternative reasonable definition for the size of a visible computation tree is to exclude nodes representing moves of the Verifier made after the last move of the Prover, since the description of the Prover's strategy does not depend on these moves. With this definition, the result of Theorem 2, which states that IP(public, poly-tree-size) = NP, no longer holds. Instead, with the alternative definition, the result is that IP(public, poly-tree-size) = MA.)

2.4 Logarithmic Random Bits

One way to guarantee polynomial tree size is to insist that the Verifier may only use $O(\log n)$ random bits in any run, where n is the length of the input. We call this restriction *log-random-bits*. In this case, any computation tree $T(\sigma)$ has polynomial size regardless of whether (P, V) is public or not. Thus any strategy of P need only list reactions to a polynomial number of potential visible histories. Define IP(log-random-bits) to be the class of languages accepted by interactive proof systems where V uses only $\log n$ random bits in any run on an input x where $n = |x|$. We have immediately:

(i) IP(log-random-bits) $\subseteq$ IP(poly-tree-size) $\subseteq$ IP(poly-strategy),

(ii) IP(public, log-random-bits) $\subseteq$ IP(public, poly-tree-size) $\subseteq$ IP(public, poly-strategy).

2.5 Logarithmic Space

One final restriction is to place a limit on the space which the Prover and Verifier may use in their computations. In this case we also place a limit on the size of the communication tape so that it cannot be used to supplant the work tapes of the Prover and Verifier. We say that the interactive proof system (P, V) is *log-space* restricted if for any input of length n both P and V use $O(\log n)$ work tape space and $O(1)$ communication tape space. A language L is a member of IP(log-space) if there is a log space bounded interactive proof system which accepts L. This definition of the log-space restriction is consistent with that of Condon and Ladner [8] and Condon [6], where log space bounded interactive proof systems were first studied in depth. Alternative definitions of logarithmic space have also been considered (see for example [10] or [7]), where only the verifier is restricted to log space, but not the prover. All our results hold for this alternative definition also.

We may easily combine the log-space restriction with any of the restrictions public, log-random-bits, and poly-tree-size. We immediately have the following:

(i) IP(log-space, log-random-bits) $\subseteq$ IP(log-space, poly-tree-size),

(ii) IP(public, log-space, log-random-bits) $\subseteq$ IP(public, log-space, poly-tree-size).

We do not try to combine log-space with poly-strategy because the definition of poly-strategy may be sensitive to the representation of the strategy on the "advice tape" when we only allow limited space by the Prover. Recall that the definition of poly-size requires that the Prover's strategy can be represented by a string τ of polynomial length. Moreover, there is a polynomial time computable strategy evaluator f, such that the prover can compute the next move from the current configuration using f and τ. This definition is robust with respect to the representation of τ if we are considering polynomial time computation by the Prover. That is, as long as one representation τ is polynomial time computable from another, τ', the class of languages does not depend on which string τ or τ' represents the strategy. However, if the prover is restricted to log space, two representations for a strategy might be computable from each other in polynomial time, but *not* in log space. In this case, poly-size is not well defined unless the representation of the strategy is specified. For example, in the definition of poly-tree-size the representation of the

strategy is made explicit. We have chosen to make poly-size as robust as possible by not restricting the representation of a strategy τ in the definition. It would be an interesting exercise to look at alternative possibilities for defining poly-size, log-space IP systems.

2.6 Amplification

All the restrictions mentioned in this section are robust with respect to "constant amplification". That is, given an interactive proof system which includes any of the restrictions (poly-strategy, polynomial-tree-size, log-random-bits, log-space, public) a new interactive proof system can be constructed satisfying the same restrictions by repeating the original interactive proof a constant number of times with the acceptance condition being that the majority of the trials are accepting. Thus, if in the original interactive proof system the probability of acceptance for inputs in the language is at least $\alpha > \frac{1}{2}$ and the probability of acceptance for inputs not in the language at most $1-\alpha$, then for any $\beta < 1$ there is an amplified interactive proof with acceptance probability at least β for inputs in the language and acceptance probability at most $1-\beta$ for inputs not in the language. The values α and β are independent of the actual input and are parameters of the interactive proof systems solely. The restrictions of polynomial size strategy, logarithmic space, and public Verifier are also robust with respect to "polynomial amplification". However, amplification by more than a constant for either polynomial tree size or logarithmic random bits interactive proof systems will lead to interactive proofs which do not satisfy the same restriction.

3 Unrestricted Space

In this section we consider interactive proof systems in which the Prover and Verifier may use unlimited space. By definition both are restricted to polynomial time, so that unrestricted space essentially amounts to polynomial space. By a series of straightforward observations, we classify all of the interactive proof systems, both public and private, with the restrictions poly-strategy, poly-tree-size, and log-random-bits.

In the next two theorems, we show that three of the resulting complexity classes are equal to MA and three are equal to NP. The class MA was introduced by Babai and Moran [3]. In our notation, the class MA consists of those languages accepted by public interactive proof systems (P, V), where every possible run of the system is restricted so that first P computes, then V computes, and the system halts. Thus, P never computes further once V starts its computation. Note that NP is equivalent to the class of languages accepted by interactive proof systems where the Verifier is deterministic, that is, an interactive proof system with nondeterministic, but no random, moves. Hence, NP is clearly contained in MA, and it is not known if this containment is proper.

Theorem 1 *MA = IP(poly-tree-size) = IP(public, poly-strategy) = IP(poly-strategy).*

Proof: We first show that MA $\subseteq$ IP(public, poly-strategy). Suppose $L \in$ MA. Let (P, V) be the public interactive proof system accepting L, where P computes and then V computes and halts.

Suppose input $x \in L$, and suppose that on strategy σ of P, $\Pr[(P, V, x, \sigma) \text{ accepts}] \geq \frac{2}{3}$. We show that σ is of polynomial size. To see this, let $c_1, c_2, \ldots c_k$ be the configurations of the computation, when P computes with strategy σ. Then σ is the list of tuples (h_i, m_i), $1 \leq i < k$, where h_i is the visible history $c_1, \ldots, c_i$ and m_i is the move of P from c_i to c_{i+1}. Since there are polynomially such tuples in the list, σ is clearly a strategy of polynomial size. If $x \notin L$ then by the definition of MA, every strategy σ of P has $\Pr[(P, V, x, \sigma) \text{ accepts}] \leq \frac{1}{3}$. Hence L is in IP(public, poly-strategy).

A similar proof shows that MA $\subseteq$ IP(poly-tree-size). Again, let (P, V) be the public interactive proof system accepting some language $L \in$ MA, where P computes and then V computes and halts. Let (P, V') be the interactive proof system which differs from (P, V) only in that all moves of V' are private. Then on any strategy σ of P, the visible computation tree $PT(\sigma)$ is of polynomial size, since all of the computation of V' is coalesced into one node of $PT(\sigma)$. This does not affect the set of strategies of P, since P makes all of its moves before V makes any move. Hence, $L \in$ IP(poly-tree-size).

We observed in Sections 2.1 and 2.3 that IP(public, poly-strategy) $\subseteq$ IP(poly-strategy) and IP(poly-tree-size) $\subseteq$ IP(poly-strategy). Hence, to complete the proof, we show that IP(poly-strategy) $\subseteq$ MA.

Let L be a language in IP(poly-strategy) and let interactive proof system (P, V) accept L, where (P, V), together with strategy evaluator f and polynomial p, satisfy the properties of Section 2.2. We show that $L \in$ MA, by constructing a new public interactive proof system (P', V') for L. On input x, P' writes on the communication tape a string τ of length at most $p(|x|)$. Once this is done, P' never moves further. Then V' simulates the computation of (P, V) on the strategy $\sigma(f, \tau)$. To do this, whenever it is P's turn and the visible history so far is h, V' computes $f(\tau, h)$. Since f is polynomial time computable, V' can simulate a move of P in deterministic polynomial time. Moreover, all of V''s moves are visible to P'.

If the input $x \in L$, then there is a τ of polynomial length, such that $\Pr[(P, V, x, \sigma(f, \tau)) \text{ accepts}] \geq \frac{2}{3}$. Hence P' can guess this string τ to ensure that x is accepted by (P', V'). If $x \notin L$, then no guess of P' will cause V' to accept with probability greater than $\frac{1}{3}$. Also, note that the strategy guessed by P' cannot depend on V''s moves, even though they are visible to P', because P' never moves once V' has made a move. Hence, $L \in$ MA. $\square$

Theorem 2 *NP = IP(public, poly-tree-size) = IP(public, log-random-bits) = IP(log-random-bits).*

Proof: Trivially, NP $\subseteq$ IP(public, log-random-bits), since NP can be modeled as an interactive proof where the Verifier is deterministic. We have already noted that IP(public, log-random-bits) $\subseteq$ IP(public, poly-tree-size) and that IP(public, log-random-bits) $\subseteq$ IP(log-random-bits). To demonstrate that IP(public, poly-tree-size) $\subseteq$ NP, let L be accepted by a public, poly-tree-size interactive proof system (P, V) where p is the polynomial which bounds the tree size. The NP algorithm to accept L on input x, is to guess a computation tree $T(\sigma)$ of size $\leq p(|x|)$, then evaluate it. By evaluate the tree we mean assign a value to each node in the tree which represents the probability of acceptance if P follows the strategy σ starting at the configuration which labels the node. Thus, the value of the root is simply $\Pr[(P, V, x, \sigma) \text{ accepts}]$. If the value of the root is at least 2/3 then the NP algorithm accepts x. The evaluation of the computation tree $T(\sigma)$ is done bottom up, with accept leaves evaluated to 1 and the reject leaves evaluated 0. A node

which represents a random move of the Verifier is evaluated as the average of the values of its two children. All other nodes are simply given the value of its sole child.

Finally, we show that IP(log-random-bits) $\subseteq$ NP. Let L be accepted by an interactive proof system (P, V) restricted to log-random-bits. The NP algorithm to accept L on input x, guesses a computation tree $T(\sigma)$ and evaluates it as above. In addition, the algorithm deterministically checks that the tree corresponds to a valid strategy of the Prover, in that on two paths with the same visible history, the Prover's move is the same. All of this can be done in polynomial time since the Verifier uses only $O(\log n)$ random bits, and hence the computation tree is of polynomial size. $\square$

4 Private, Logarithmic Space

In this section, we consider log-space bounded interactive proof systems with the restrictions poly-tree-size or log-random-bits. The main results of this section is stated in the following theorem.

Theorem 3 *NP = IP(log-space, log-random-bits) = IP(log-space, poly-tree-size).*

To prove this, we first show in Lemma 1 that IP(log-space, poly-tree-size) $\subseteq$ NP. In Lemma 2, we show that NP $\subseteq$ IP(log-space, log-random-bits). The theorem follows immediately from these two lemmas and observation (i) of Section 2.5. All of the proofs in this section are valid even without the assumption that the interactive proof system runs in polynomial time, as long as it halts with probability 1 on any strategy of the Prover.

Lemma 1 *IP(log-space, poly-tree-size) $\subseteq$ NP*

Proof: Let L be in IP(log-space, poly-tree-size) and let (P, V) be the restricted interactive proof system accepting L. We describe a nondeterministic polynomial time algorithm that accepts L.

To decide if an input x is in L, this algorithm guesses a strategy of the Prover and computes the probability that the Verifier accepts, when the Prover uses that strategy. If $x \in L$, the Prover has a strategy σ which can be represented by a visible computation tree, $PT(\sigma)$, of size polynomial in $|x|$. This tree can be guessed in polynomial time. The main part of the proof is to show how to compute the probability that the Verifier accepts, when the Prover uses strategy σ.

Let m be the number of configurations of (P, V) on input x, and assume that the configurations are numbered from 1 to m. Note that $m = poly(n)$, since the computation is $O(\log n)$ space bounded. For convenience, assume that the initial configuration is numbered 1 and that there is a unique accepting configuration, which is numbered m.

We associate a probability vector $\bar{u} = (u(1), \ldots, u(m))$ with each node N of the tree $PT(\sigma)$. Entry $u(j)$ is defined to be the probability that configuration j is reached given that the computation of (P, V, x, σ) follows the path from the root to the node N in $PT(\sigma)$. Thus, $u(j) = 0$ unless the visible part of j is identical to the visible configuration which labels node N. Clearly, the vector associated with the root is $(1, 0, \ldots, 0)$, since we are assuming that the initial configuration is numbered 1. If N is a leaf, then entry $u(m)$ is the probability that V accepts, given that the visible

part of the computation is the sequence of visible configurations which labels the path from the root of $PT(\sigma)$ to N. Hence, the probability that V accepts, given that the Prover uses strategy σ, is the sum, over the leaves of $PT(\sigma)$, of $u(m)$.

We claim that the vectors can be computed in polynomial time; hence the probability that V accepts can be computed in polynomial time. The vectors are computed level by level, starting with the root, which is known initially. Suppose $\bar{u}$ labels node N. We show how to compute the vector $\bar{u}'$ labeling a child N' of N.

This is done using an $m \times m$ matrix M, defined as follows. If i and j are configurations with distinct visible parts, then M_{ij} is the probability of reaching configuration j from configuration i, while only going through configurations whose visible part is the same as i. Otherwise, if i and j are configurations with the same visible parts, then $M_{ij} = 0$. Let $\bar{u}'' = \bar{u}M$. For $1 \leq j \leq m$, let $u'(j) = u''(j)$, if configuration j is a configuration whose visible part equals the visible configuration labeling N', and let $u'(j) = 0$ otherwise. Let $\bar{u}' = (u'(1), \ldots, u'(m))$. Then $\bar{u}'$ is the vector associated with node N'.

We next show that all vectors can be computed in polynomial time. To prove this, we will prove that the matrix M can be computed in polynomial time, and its entries are rational numbers of polynomial length. Given this, it follows that all the vectors $\bar{u}$ labeling the nodes of the tree have polynomial length. This is because if $\bar{u}'$ labels a child of a node labeled by $\bar{u}$, then each entry of $\bar{u}'$ is either 0 or is an entry of $\bar{u}M$, and so the maximum length of the entries of $\bar{u}'$ is at most the maximum length of an entry of $\bar{u}$ plus the maximum length of an entry of M plus 1. Since the root is labeled $(1, 0 \ldots, 0)$, it follows that the length of an entry of a vector labeling any node is bounded by the distance of the node from the root times the maximum length of an entry of M plus 1. Since the tree has polynomial depth, the entries of all vectors labeling nodes of the tree have polynomial length.

The proof that M can be computed in polynomial time is similar to a proof of Gill ([14], Theorem 6.4) for probabilistic Turing machines. We are assuming throughout that P uses strategy σ. The jth column of M is computed as follows. Without loss of generality, suppose that the non-halting configurations with distinct visible parts from configuration j are numbered $1, \ldots, k$. Then clearly entries M_{lj}, $k < l \leq m$, are 0. Let $M_j = (M_{1j}, M_{2j}, \ldots M_{kj})^T$. Let Q be the $k \times k$ matrix, where Q_{rs} is the 1-step transition probability from configuration r to configuration s, if r has the same visible part as s, and Q_{rs} is 0 otherwise. Let $\bar{b}$ be the k-vector where $b(i)$ is the probability that configuration j is reached from configuration i in 1 step. Then $M_j = QM_j + \bar{b}$, or equivalently, $2(I - Q)M_j = 2\bar{b}$.

The matrix Q and vector $\bar{b}$ can be computed directly from the definition of (P, V) and σ, and all of their entries are from the set $\{0, \frac{1}{2}, 1\}$. We now show that $(I - Q)$ (and hence $2(I - Q)$) is invertible. To do this, we show that $\lim_{l \to \infty} Q^l = 0$. From this, it follows that 1 cannot be an eigenvalue of Q, and thus $I - Q$ is invertible. To see that $\lim_{l \to \infty} Q^l = 0$, note that entry rs of Q^l is the probability that in exactly l steps, configuration s is reached from configuration r. Since r, s are non-halting configurations and the Verifier halts with probability 1, the probability that s is reached in exactly l steps approaches 0 as l goes to infinity.

Hence, by Cramer's rule, each entry e of M_j can be expressed as the quotient of two integers N_e/D where D is the determinant of $2(I - Q)$ and $N_e \leq D$. Also, from the definition of Q it follows

that each row of $2(I-Q)$ has a constant number of non-zero entries, which are integers whose absolute values sum to at most 4. Using this fact and expansion by minors, it can be shown by induction that the determinant of $2(I-Q)$ is at most 2^{k+1}. Thus each entry of M_j can be written in the form p/q where p and q are integers of length polynomial in n, and these entries can be computed from Q and $\bar{b}$ in polynomial time. □

In previous work, Condon [6] showed that NP $\subseteq$ IP(log-space), even when the Verifier runs in polynomial time. However, the Verifier could use polynomially many random bits. In the next lemma, we show how the number of bits is reduced to $O(\log n)$.

Lemma 2 *NP $\subseteq$ IP(log-space, log-random-bits).*

Proof: The proof of Lemma 2 was actually begun in the introduction, Section 1.1, where we explained why 3-SAT is in IP(log-space, log-random-bits). We next complete the proof that 3-SAT is in IP(log-space, log-random-bits). Once this is established, we can conclude that NP $\subseteq$ IP(log-space, log-random-bits) because IP(log-space, log-random-bits) is closed under log-space reductions and 3-SAT is NP-complete for log-space reductions.

Let f be an instance of the 3-SAT problem. We showed in Section 1.1 that to test whether f is satisfiable, the Verifier must test the equality of two multi-sets, which have size linear in the size of the formula, f. The first and second multisets consist of the triples (l,i,v), sent by the Prover to the Verifier in the first and second stages, respectively, of the interactive proof system for 3-SAT described in Section 1.1, on input f. Lipton showed how to test for equality of two multisets using very few random bits. This is done by comparing a *fingerprint* of each multiset. A fingerprint of a multiset $\{x_1, x_2, \ldots, x_k\}$ is defined to be $\prod_{i=1}^{k}(x_i + r) \bmod p$, where p is a prime and $r \in \{1, \ldots, p-1\}$. Lipton proved the following:

Fingerprinting Lemma (Lipton [18]) *The probability that $\{x_1, \ldots, x_k\}$ and $\{y_1, \ldots, y_l\}$ are unequal and get the same fingerprints is at most*

$$O\left(\frac{\log(b) + \log(m)}{bm} + \frac{1}{b^2 m}\right)$$

where all numbers are b bit numbers and $m = \max(k,l)$ provided the prime p is selected randomly and uniformly from the interval $[(bm)^2, 2(bm)^2]$ and r is selected randomly and uniformly from the interval $[1, p-1]$.

If n is the length of the formula then the triples (l,i,v) in each multiset can be represented as $O(\log n)$ bit numbers and the size of each set is $O(n)$. Thus the prime p is selected from a range of $O(\log n)$ bit numbers, as is the number r. Once p and r are selected, the Verifier can compute a fingerprint in $O(\log n)$ space as follows. The Verifier repeatedly receives a triple (l,i,v) from the Prover and computes the term $N(l,i,v) + r$, where $N(l,i,v)$ is the number representing (l,i,v). Then the Verifier multiplies this new term with the partial fingerprint obtained so far, and takes the result mod p to obtain a new partial fingerprint. (The initial partial fingerprint is 1).

It remains to show that p and r can be chosen using $O(\log n)$ random bits. This is done simply as follows. To choose p, the Verifier first counts the number of primes in the range, say $[s, 2s]$,

specified by the Fingerprinting Lemma. Since all numbers in the range have $O(\log n)$ bits, the Verifier can do this deterministically in $O(\log n)$ space and polynomial time. Suppose that there are i primes in this range. Then, the Verifier randomly and uniformly chooses an integer between 1 and i and selects the ith prime in the range $[s, 2s]$ as the prime p. To select an integer between 1 and i, the Verifier selects $\lceil \log i \rceil$ bits randomly and uniformly, which determines a number in the range $0, \ldots, 2^{\lceil \log i \rceil} - 1$. By adding 1 to this number, the Verifier obtains a number in the range $1, \ldots, 2^{\lceil \log i \rceil}$. If the number happens to be in the range $1, \ldots, i$, the Verifier is done; else the Verifier repeats this procedure a second time, and then a third time if there is no success the second time. If in all of the three trials, the Verifier chooses a number outside of the range $1, \ldots, i$, the Verifier halts and accepts the input. The probability of this occurring is at most 1/8, since at least half of the numbers in the range $1, \ldots, 2^{\lceil \log i \rceil}$ are in the range $1, \ldots, i$. Otherwise, if the prime p is successfully chosen, the Verifier chooses a number r randomly and uniformly from the range $1, \ldots, p-1$ in the same way that a number was chosen from the range $1, \ldots, i$. With probability at least 3/4, the Verifier is successful in finding p and r, using $O(\log n)$ random bits.

Clearly, if the formula is satisfiable there is a strategy for the Prover which leads to acceptance with probability one. If the formula is not satisfiable then the interactive proof system accepts with probability at most 1/3. One source of error is due to the fact that the Verifier may incorrectly accept a formula which is not satisfiable, if the random numbers p and r are not successfully chosen. We have just shown that this happens with probability at most 1/4. The other source of error is due to the fact that even if two multisets are not equal, they may get the same fingerprint. The Fingerprinting Lemma guarantees that this happens with probability at most $O(1/n)$, where n is the size of the formula. Hence, the protocol erroneously accepts with probability at most 1/3.

This completes the proof that 3-SAT $\in$ IP(log-space, log-random-bits). Hence, NP $\subseteq$ IP(log-space, log-random-bits). $\square$

5 Public, Logarithmic Space

In this section we consider public interactive proof systems in which the Verifier is restricted to log-space. Let NLOG be the class of languages accepted by a nondeterministic $O(\log n)$-space bounded Turing machine and let LOGCFL be the class of languages which are $O(\log n)$-space reducible to context-free languages [22]. We have the following relationships.

Theorem 4 *NLOG $\subseteq$ IP(public, log-space, poly-tree-size) =*
IP(public, log-space, log-random-bits) $\subseteq$ LOGCFL.

The inclusion NLOG $\subseteq$ IP(public, log-space, poly-tree-size) follows because NLOG can be modeled as a log-space interactive proof system with a deterministic Verifier. We have already noted the inclusion IP(public, log-space, log-random-bits) $\subseteq$ IP(public, log-space, poly-tree-size). To complete the proof we need to show IP(public, log-space, log-random-bits) $\subseteq$ LOGCFL (Lemma 3) and IP(public, log-space, poly-tree-size) $\subseteq$ IP(public, log-space, log-random-bits) (Lemma 4).

Lemma 3 *IP(public, log-space, log-random-bits) $\subseteq$ LOGCFL.*

Proof: To argue that IP(public, log-space, log-random-bits) $\subseteq$ LOGCFL we use the equivalence of LOGCFL and the class of languages accepted by nondeterministic pushdown automata which have $O(\log n)$ auxiliary storage and run in polynomial time [22]. Suppose $L \in$ IP(public, log-space, log-random-bits) and let x be an input to a public, log-space, log-random-bits, interactive proof system (P, V) which accepts L. We will construct an auxiliary (nondeterministic) PDA M which accepts L. Generally speaking, M evaluates a computation tree $T(\sigma)$ for some σ. The nondeterministic moves of the PDA can simulate the strategy σ of the Prover. The pushdown store of the PDA is used to search and store partial evaluations of $T(\sigma)$. The PDA runs in polynomial time because the tree it has to search is of polynomial size. The PDA only uses log-space because the probabilities it has to calculate can be represented in $O(\log n)$ bits.

To evaluate $T(\sigma)$ for some σ, M starts with the unevaluated initial configuration of (P, V) (except for the input) in its storage and an empty pushdown store. In a general situation M has a configuration (evaluated or unevaluated) in its storage and the sequence of configurations (partially evaluated or unevaluated) leading to that configuration on its pushdown store. At termination the evaluated initial configuration will be in its storage and the pushdown store will be empty again. If the value of the evaluated initial configuration is $\geq \frac{2}{3}$ then M accepts. What is left to explain is what M does in a general situation.

1. If the configuration C in the storage is unevaluated then
   (a) if C is accepting configuration then give it the value 1,
   (b) if C is a rejecting configuration then give it the value 0,
   (c) if C is a configuration in which it is P's turn to move then nondeterministically choose a successor configuration to replace C in the storage and push C onto the pushdown store,
   (d) if C is a configuration in which it is V's turn to move randomly then choose the successor configuration which corresponds to the random choice 0 to replace C in the storage and push C onto the pushdown store,
   (e) if C is a configuration in which it is V's turn to move deterministically then choose the successor configuration to replace C in the storage and push C onto the pushdown store.
2. If the configuration C in the storage has a value then, if the pushdown store is empty then C is the evaluated initial configuration, otherwise, let D be the configuration on the top of the pushdown store and
   (a) if D is a configuration in which it is P's turn to move then then remove D from the pushdown store and have it replace C in the storage and D's value becomes C's value,
   (b) if D is a configuration in which it is V's turn to move randomly and C is the successor configuration which corresponds to the random choice 0 then replace C with the successor of D which corresponds to the random choice 1 and leave D at the top of the pushdown store with the partial value equal to the value of C,

(c) if D is a configuration in which it is V's turn to move randomly and C is the successor configuration which corresponds to the random choice 1 then D must have a partial value; compute D's value as the average of D's partial value and C's value, then remove D and its value from the pushdown store and have them replace C and its value in the storage,

(d) if D is a configuration in which it is V's turn to move deterministically then remove D from the pushdown store and have it replace C in the storage and D's value becomes C's value.

The proof that M accepts L is tedious, requiring the proof of an invariant which describes the relationship between the evaluation of a computation tree $T(\sigma)$ and the computation of M. Roughly speaking the invariant states that the pushdown store of M contains the labels of a root to node path in $T(\sigma)$ for some σ and the contents of the storage the label of a successor of the last node in the path. If a configuration in the pushdown store which corresponds to a node N in $T(\sigma)$ has a partial value then the configuration is random and the partial value is the value of the subtree of $T(\sigma)$ rooted the successor of N which corresponds to the random choice 0. If the configuration in the storage has a value then it is the value of the subtree of $T(\sigma)$ rooted at the node labeled with that configuration. The statement of the complete invariant and its proof is left to the reader. Thus, M accepts the language L.

What is left is to demonstrate that M uses $\log n$ auxiliary storage and polynomial time. Notice that if an evaluated configuration is one in which it is P's turn to move or V's turn to move deterministically, then its value is identical to its successor. If on the other hand, an evaluated configuration is one in which it is V's turn to move randomly, then its value requires at most one bit more to represent it the maximum number of bits to represent one of its successors. Thus, at most $O(\log n)$ bits are needed to represent any of the values in an evaluated computation tree which has at most $O(\log n)$ random nodes on any branch. Furthermore, since (P, V) must also have polynomial tree size an accepting computation of M on input x will run in polynomial time as a function of $|x|$. □

Lemma 4 *IP(public, log-space, poly-tree-size)* $\subseteq$ *IP(public, log-space, log-random-bits).*

Proof: The basic idea is to use tree pruning to reduce the amount of the randomness while reducing the probability of acceptance only slightly. Let T be a binary branching tree with $N > 0$ nodes. In a binary branching tree every internal node has exactly two children. Define T_i to be T pruned to nodes of height i or less. Thus, T_0 has one node, namely, the root of T, T_1 contains the root of T and its two children if it has children, and $T_m = T$ for all $m \geq$ the height of T. Define $w(i)$ to be the probability that a leaf of T is reached in traversing a root-to-leaf path in T_i assuming that each of the two children of an internal node are chosen with equal probability. In traversing a root-to-leaf path in T_i either the leaf reached is also a leaf of T because the length of the path is $< i$, or the leaf reached is an internal node of T. In the former case the traversal is a success and in the latter a failure.

Tree Pruning Lemma. *For all* $k \geq 1$, $w(\lceil \log N \rceil + k - 1) > 1 - 2^{-k}$.

Given the tree pruning lemma we can argue that $O(\log n)$ random bits are sufficient. Let L be accepted by an interactive proof system (P, V) which is public, log-space, and poly-tree-size. We construct (P', V') which is a simple modification of (P, V). Let x be an input of length n and let $N = poly(n)$ be a bound on the tree size. The new interactive proof system (P', V') will simulate move-for-move (P, V) except if V' makes more than $\lceil \log N \rceil + 2$ random moves then V' immediately halts and rejects x. By the tree pruning lemma, choosing $k = 3$, if σ is any strategy for P, then σ is also a strategy for P' with the property that

$$\Pr[(P, V, x, \sigma) \text{ accepts}] \geq \Pr[(P', V', x, \sigma) \text{ accepts}] \geq \tfrac{7}{8}\Pr[(P, V, x, \sigma) \text{ accepts}]$$

Technically, the tree pruning lemma is applied to the tree T which is formed by collapsing maximal sequences of non-random nodes of $T(\sigma)$ into single nodes of T. The tree T is then binary branching and has $\leq N$ nodes. If $x \in L$ then there is a σ such that $|T(\sigma)| \leq N$ and $\Pr[(P', V', x, \sigma) \text{ accepts}] \geq \frac{7}{8}\Pr[(P, V, x, \sigma) \text{ accepts}] \geq \frac{14}{24}$. If $x \notin L$ then for all σ, $\Pr[(P', V', x, \sigma) \text{ accepts}] \leq \Pr[(P, V, x, \sigma) \text{ accepts}] \leq \frac{1}{3}$. The interactive proof system (P', V') uses the same space as (P, V) plus the space needed to count to $\log N$. This space is bounded by $O(\log n)$. Finally, (P', V') must be amplified to bring the acceptance probability for inputs in the language up to $\frac{2}{3}$.

What remains is to prove the Tree Pruning Lemma. Let T be a tree with $N > 0$ nodes and $k \geq 1$. Define n_i be the number of nodes of T which are exactly at height i in T. It is a simple observation that $w(i) = 1 - n_{i+1}2^{-(i+1)}$. Assume, for purposes of leading to a contradiction, that for all $m \leq \lceil \log N \rceil + k - 1$, $n_{m+1}2^{-(m+1)} \geq 2^{-k}$. Then, we have

$$\begin{aligned} N - 1 &= \textstyle\sum_{j=0}^{\infty} n_{j+1} \\ &\geq \textstyle\sum_{j=0}^{\lceil \log N \rceil + k - 1} n_{j+1} \\ &\geq \textstyle\sum_{j=0}^{\lceil \log N \rceil + k - 1} 2^{j+1-k} \\ &= 2^{1-k}(2^{\lceil \log N \rceil + k} - 1) \\ &\geq 2^{\lceil \log N \rceil} \\ &\geq N. \end{aligned}$$

This is a contradiction. Thus, we have for some $m \leq \lceil \log N \rceil + k - 1$, $n_{m+1}2^{-(m+1)} < 2^{-k}$. Hence, we can conclude

$$w(\lceil \log N \rceil + k - 1) \geq w(m) = 1 - n_{m+1}2^{-(m+1)} > 1 - 2^{-k}.$$

□

6 Conclusion

We have identified what we feel captures the notion of a practical interactive proof system, namely, an interactive proof system in which the Prover can only rely on a polynomial size strategy. We explored several additional restrictions such as polynomial tree size, logarithmic random bits, and logarithmic space. For the most part we were able to classify fully what the effects of these restrictions are.

One important problem left is to identify problems which have the full power of IP(poly-strategy) which is identical to Babai and Moran's class MA [3]. Such problems could serve as a stronger cornerstone for cryptography than problems that are currently used such as factoring, quadratic residuosity, and discrete logarithms which in some sense are all "NP-like" problems.

Another intriguing problem is to classify IP(public, log-space, log-random-bits) more fully. Could this class be equal to LOGCFL? Fortnow and Lund have shown that NC is contained IP(public, log-space), but their simulations use $\log^{O(1)} n$ coin flips [13]. When their simulation is adapted to LOGCFL, which is contained in NC^2 [20], then the number of coin flips is $O(\log^2 n)$. Reducing the number of coin flips to $O(\log n)$ appears to require a new technique.

Finally, there is the question of how this theory of restricted interactive proofs systems could be brought into the theory of zero-knowledge interactive proof systems. It is our impression that many researchers working in zero-knowledge, which is closely related to cryptography, are already somewhat practically motivated. Thus, they are, for the most part, already implicitly using restricted interactive proof systems.

7 Acknowledgement

We would like to thank Osamu Watanabe for pointing out the relationship between our model and the work of Feige, Fiat and Shamir. Thanks also to the anonymous reviewers for suggesting many improvements to the presentation.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, Proof verification and hardness of approximation problems. Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, pp.14-23, 1992.

[2] S. Arora and M. Safra, Probabilistic checking of proofs. Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, pp.2-13, 1992.

[3] L. Babai and S. Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computing and System Sciences*, 36, pp. 254-276, June 1988.

[4] M. Blum and S. Kannan. Designing programs that check their work. Proceedings of the Twenty-first Annual Symposium on the Theory of Computing, pp. 86-97, 1989.

[5] A. Condon. Computational Models of Games. MIT Press, 1989.

[6] A. Condon. Space Bounded Probabilistic Game Automata. *Journal of the ACM*, 38(2):472-494, April 1991.

[7] A. Condon. The complexity of the max word problem and the power of one-way interactive proof systems. Proceedings of 8th Symposium on Theoretical Aspects of Computer Science, pp. 456-465, February, 1991.

[8] A. Condon and R. E. Ladner. Probabilistic game automata. *Journal of Computer and System Sciences*, 36(3), pp. 452-489, June 1988.

[9] A. Condon and R. E. Lipton. On the complexity of space bounded interactive proofs. Proceedings of the 30th Annual Symposium on the Foundations of Computer Science, pp. 462-467, 1989.

[10] C. Dwork and L. Stockmeyer. Finite state verifiers I: the power of interaction. *Journal of the ACM*, 39(4):800-828, October 1992.

[11] U. Feige, A. Fiat and S. Shamir. Zero knowledge proofs of identity. *Journal of Cryptology*, 1(2):77-94, 1988.

[12] L. J. Fortnow. Complexity-theoretic aspects of interactive proof systems. Ph. D. Thesis, Technical Report Number TR-447, Laboratory for Computer Science, MIT.

[13] L. Fortnow and C. Lund. Interactive proof systems and alternating time-space complexity. Proceedings of 8th Symposium on Theoretical Aspects of Computer Science, pp. 263-274, February, 1991.

[14] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, **6, No. 4**, pp. 675-695, 1977.

[15] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3), pp. 691-729, July, 1991.

[16] S. Goldwasser, S. Micali and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18, pp. 186-208, 1989.

[17] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. *Advances in Computing Research 5: Randomness and Computation*, JAI Press, Greenwich, CT, 1989.

[18] R. J. Lipton. Efficient checking of computations. Proceedings of the 7th Symposium on Theoretical Aspects of Computer Science, February, pp. 207-215, 1990.

[19] C. Lund, L. Fortnow, H. Karloff, N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859-868, October 1992.

[20] W. L. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21(2), pp. 218-235, 1980.

[21] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869-877, October 1992.

[22] I.H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3), pp. 405-114, 1978.