INTERACTIVE PROOF SYSTEMS: 
PROVERS THAT NEVER FAIL AND RANDOM SELECTION

by

O. Goldreich and Y. Mansour

Technical Report #459

(JUNE 1987)
Interactive Proof Systems: 
Provers that never Fail and Random Selection

Oded Goldreich
Computer Science Dept.
Technion
Haifa, Israel

Yishay Mansour
IBM Scientific Center
Technion City
Haifa, Israel

ABSTRACT

An interactive proof system with Perfect Completeness (resp. Perfect Soundness) for a language $L$ is an interactive proof (for $L$) in which for every $x \in L$ (resp. $x \notin L$) the verifier always accepts (resp. always rejects). We show that any language having an interactive proof system has one with perfect completeness. On the other hand, only languages in $NP$ have interactive proofs with perfect soundness.

In the proof of the main result, we use a new protocol for proving approximately lower bounds and "random selection". The problem of random selection consists of a verifier selecting at random, with uniform probability distribution, an element from an arbitrary set held by the prover. Previous protocols known for approximate lower bound do not solve the random selection problem.

---

First author was partially supported by the Fund for Basic Research Administered by the Israel Academy of Sciences and Humanities. Second author is currently at the Laboratory for Computer Science, MIT, Cambridge, MA 02139.
1. INTRODUCTION

The two basic notions regarding a proof system are completeness and soundness. Completeness means that the proof system is powerful enough to generate "proofs" for all the valid statements. Soundness means that any statement that can be proved is valid (i.e., no "proofs" exist for false statements). Two computational tasks related to a proof system are generating a proof and verifying the validity of a proof. This naturally suggests the notions of a prover (a party able of generating proofs) and a verifier (a party capable of validating proofs). Typically, the verifier's task is easier than the prover's task. In order to focus on the complexity of the verification task it is convenient to assume that the prover has unlimited power.

For many years \( \mathbf{NP} \) was considered the formulation of "whatever can be efficiently verified". This stemmed from the association of deterministic polynomial-time computation with efficient computation. The growing acceptability of probabilistic polynomial-time computations as reflecting efficient computations is the basis of more recent formalizations of "whatever can be efficiently verified". In these formalizations, due to Goldwasser, Micali and Rackoff [GMR] and Babai [B], and shown to be equivalent by Goldwasser and Sipser [GS], the (polynomial time) verifier is allowed to toss coins and arbitrarily interact with the prover, furthermore he can accept or reject based on overwhelming statistical evidence. Ruling by overwhelming statistical evidence means relaxing the completeness and soundness conditions so that any valid statement can be proved with a very high probability while any false statement has only negligible probability to be proved. A bit more formally:

An interactive proof system for a language \( L \) is a two-party protocol for a prover and a verifier guaranteeing that, on common input \( x \), when the verifier follows his (polynomial-time) program, the following two conditions hold:

1. **Completeness** (in a probabilistic sense): For every \( x \in L \), if the prover follows his program then the verifier accepts \( x \) with probability \( \geq 1 - 2^{-|x|^1} \). In other words, the prover can "convince" the verifier of \( x \in L \).

2. **Soundness** (in a probabilistic sense): For every \( x \notin L \), no matter what the prover does, the verifier rejects \( x \) with probability \( \geq 1 - 2^{-|x|^1} \). In other words, no prover can "fool" the verifier.
We denote by $IP$ the class of languages for which there exists an interactive proof system. Clearly, $NP \subseteq IP \subseteq PSPACE$. It is believed that the class $NP$ is strictly contained in $IP$. Evidence for this can be derived from the fact that, relative to some oracle, interactive proofs are even not contained in the polynomial-time hierarchy, i.e. $\exists A \ s.t. \ IP^A \nsubseteq PH^A \neq \emptyset$ (see [AGH]). It is also interesting to note that natural languages as Graph Non-Isomorphism and Matrix Group Non-Membership, which are not known to be in $NP$, were shown to be in $IP$ (by [GMW] and [B], respectively).

Considering an interactive proof system, it seems that in some sense the prover is "responsible" for the completeness condition, while the verifier is "responsible" for the soundness condition. If this intuition is correct, and the prover has unrestricted power, why should the completeness condition be relaxed? Namely, can we modify the interactive proof such that the prover never fails in demonstrating the validity of true statements, while maintaining soundness. By perfect completeness we mean that the prover never fails to prove the membership of inputs that are indeed in the language, while perfect soundness means that the verifier never accepts inputs that are not in the language.

Perfect completeness and perfect soundness are not only theoretically interesting, but are also of practical importance. This is the case, since probabilistic completeness and soundness are defined with respect to ideal (unbiased) coin tosses and may not hold when using pseudo random sequences (even in the sense of [BM,Y]). On the other hand perfect completeness and soundness are independent of the quality of the verifier coin tosses.

Our main result is a transformation that given an interactive proof for a language $L$ yields an interactive proof with perfect completeness for $L$. We conclude that Interactive Proofs with Perfect Completeness are as powerful as Interactive Proofs. In particular, Arthur-Merlin interactive proofs with perfect completeness can be presented for Non-Membership in Matrix Groups, and for Graph Non-Isomorphism. Now, what about interactive proofs with perfect soundness? Unfortunately, we show that they are only as powerful as $NP$.

Looking at $IP$ as a randomized complexity class with two-sided error, one can draw an analogy between it and $BPP$. In this analogy, $IP$ is a randomized version of $NP$ while $BPP$ is a randomized version of $P$. Continuing with this analogy, interactive proofs with perfect completeness (i.e. no error on the Yes-instances) correspond to $Co-R$, while interactive proofs with perfect soundness (i.e. no error on the
No-instances) correspond to $R$. What we have shown is that the analogue of $BPP = \text{the analogue of Co-R}$, while the analogue of $R = \text{the analogue of P}$.

In the proof of the main result, we formulate and solve a new protocol problem called random selection. Suppose that the prover has in mind a large set $S \subseteq \{0,1\}^*$ of size $N$. The verifier, knowing $N$, wishes to select at random with uniform probability distribution an element in $S$. Since the set $S$ is known only to the prover, we can only require that every element selected by the verifier has probability $\frac{1}{N}$, and that there exists a prover such that for every $S$, all the elements in $S$ can be selected by the verifier. In such a random selection the prover is guaranteed that he is only giving away elements of $S$, while the verifier is guaranteed that no element is given to him with probability greater then $\frac{1}{N}$ (no matter how the prover plays).

Any protocol for random selection constitutes a protocol for proving approximate lower bound. The converse, however, is false. In fact, all the known protocols for proving approximate lower bounds [B,GS] do not solve the random selection problem. We present a protocol for random selection and proving approximate lower bound. The advantage of our protocol is that the prover never fails in proving a correct lower bound, whereas the previous protocols for this problem (see [B] and [GS]), do not have this property.

2. MODEL AND DEFINITIONS

Both the Interactive Proof systems (introduced by [GMR]) and the Arthur Merlin games (introduced by [B]) are based on interaction between a powerful prover and a probabilistic polynomial time verifier. The two parties have a common input $x$, and they interact in order to allow the verifier to reach a decision about the membership of the input in a specific language. The aim of the prover is to convince the verifier of the membership of the input in the language, while the verifier demands strong evidence to support this claim, but is willing to accept overwhelming statistical evidence. The difference between the two formalisms is that in an Arthur Merlin game the verifier (named Arthur) is limited to send to the prover the outcome of his coins, while in an interactive proof system the verifier can send to the prover a function of his coin tosses and does not necessarily reveal the outcome of the coins. Goldwasser and Sipser [GS] proved the equivalence between the two models; namely that any interactive proof, with $q(n)$ interactions, can be simulated by an Arthur Merlin game with $q(n)+2$ interactions.
We state and prove our main result for the Arthur Merlin games introduced by Babai [B]. Using the result of [GS] our main result applies also to the interactive proof systems of [GMR]. Since we are interested only in the complexity theoretic aspects of proof systems, we may assume that the prover (Merlin) uses an optimal strategy and therefore, with no loss of generality, is deterministic. In the following definition we assume that in all interactions of Arthur and Merlin, on inputs of the same length, the same number of messages are exchanged and that all these messages are of the same length. Clearly, this condition is immaterial and is only placed in order to facilitate the analysis.

Definition 1 (Arthur Merlin games):

An Arthur Merlin game is a pair of programs A and M (i.e. a two-party protocol) and a function \( p \) such that:

1. On common input \( x \), exactly \( 2q(1x1) \) messages of length \( m(1x1) \) each are exchanged, where \( q \) and \( m \) are fixed polynomials.

2. Arthur (A) goes first, and at iteration \( 1 \leq i \leq q(1x1) \) chooses at random a string \( r_i \) of length \( m(1x1) \), with uniform probability distribution.

3. Merlin's reply in the \( i \)-th iteration, denoted \( y_i \), is a function of all the previous choices of Arthur and the common input \( x \). More formally, \( y_i = M(x, r_1, \ldots, r_i) \).

4. For every program \( M' \), a conversation between A and M' on input \( x \) is a string \( r_y \), \( y_i = M'(x, r_1, \ldots, r_i) \), where for every \( 1 \leq i \leq q(1x1) \) \( y_i = M'(x, r_1, \ldots, r_i) \). We denote by \( CONV_{x}^{M'} \) the set of all conversations between A and M' on input \( x \). Note that \( |CONV_{x}^{M'}| = 2^{q(1x1)m(1x1)} \).

5. The function \( p \) is a polynomial-time computable mapping of the input \( x \) and the conversation \( r_y \) to a value \( p(x, r_y) \). This value is called the value of the conversation.

Originally, Arthur Merlin games were introduced to discuss language recognition. In that setting the value of a conversation is either accept or reject (i.e. \( p \) is a Boolean predicate). We adopt this convention when discussing language recognition problems. The generalization allows us to discuss a new problem called random selection (see Section 3). For the rest of this section we discuss Arthur Merlin games for language recognition.
Notation: Let A and M' be programs of an AM game as above, $ACC^M$ denotes the set
\[ \{ r_1, \ldots, r_e(x) \mid \exists y_1, \ldots, y_e(x) \ S.L. \ r_1 y_1 \ldots r_e(y_e(x)) \in CONV^M \} \]

Intuitively, $ACC^M$ is the set of all the random choices leading A to accept x, when interacting with $M'$. Note that $ACC^M$ depends only on Merlin ($M'$), since we assume that Arthur follows the protocol.

The ratio $\frac{|ACC^M|}{|CONV^M|}$ is the probability that Arthur accepts x when interacting with $M'$.

Definition 2 (Arthur Merlin proof systems): An AM proof system for language L is an AM game satisfying the following two conditions:

1. For M (as specified in the game), for all $x \in L$, $\frac{|ACC^M|}{|CONV^M|} > \frac{2}{3}$. (This condition is hereafter referred to as probabilistic-completeness.)

2. For every $M'$ and for any $x \not\in L$, $\frac{|ACC^M|}{|CONV^M|} \leq \frac{1}{3}$. (This condition is hereafter referred to as probabilistic-soundness.)

An equivalent definition is obtained by replacing $1/3$ by $2^{-p(x)}$ and $2/3$ by $1 - 2^{-p(x)}$, where $p(\cdot)$ is an arbitrary polynomial satisfying $p(n) > 1$ (for $\forall n > 1$).

Definition 3 (perfect completeness): An AM proof system with perfect-completeness for a language L is an AM proof system for L satisfying:

\[ \forall x \in L \quad |ACC^M| = |CONV^M| \]

Perfect-completeness, of an AM proof system, means that an honest Merlin always succeeds in convincing Arthur to accept inputs in the language.

Definition 4 (perfect soundness): An AM proof system with perfect-soundness for a language L is an AM proof system for L satisfying:

\[ \forall M' \quad \forall x \not\in L \quad ACC^M = \emptyset \]

Perfect-soundness, of an AM proof system, means that no matter what Merlin does Arthur never accepts an input not in language.
3. RANDOM SELECTION AND APPROXIMATE LOWER BOUND

Definition 5 (Random Selection): Let $L \subseteq \{0,1\}^*$, $L_0 := L \cap \{0,1\}^*$, and $L_1 = \{1\}^k$. A random selection from $L$ is an Arthur Merlin game (i.e. programs $A$ and $M$ and a function $\rho$) satisfying the following:

1) The value of a conversation between $A$ and $M$, on input $n$ (in unary) and $L_\alpha$, is uniformly distributed in $L_\alpha$. (That is, for every $w \in L_\alpha$, the probability $\text{Prob}(\rho(\text{conv}) = w) = \frac{1}{|L_\alpha|}$, where conv is selected with uniform probability distribution from $CONV^M_{(\alpha,L_\alpha)}$.)

2) For every $M'$, the value of a conversation between $A$ and $M'$, on input $n$ and $L_\alpha$, is either a $n$-bit string or a (special string called) error. Furthermore, for every $w \in \{0,1\}^*$, the probability $\text{Prob}(\rho(\text{conv}) = w)$ is either 0 or $\frac{1}{|L_\alpha|}$, where conv is selected with uniform probability distribution from $CONV^{M'}_{(\alpha,L_\alpha)}$.

Notation: Let $S(M',L)$ be the set of the values of conversations between $A$ and $M'$ (on input $n$ and $L$), excluding the error value. (That is, $S(M',L) = \rho(\text{CONV}^{M'}_{(\alpha,L_\alpha)}) - \{\text{error}\}$, where $\rho$ is appropriately extended to sets.) Whenever clear from the context, we use $SM'$ for $S(M',L)$.

Note that if Arthur has an efficient procedure for testing membership in $L$, then he can check whether Merlin tried to cheat him. In any case, Merlin can not enhance the probability of a single string to occur as the value of a conversation. The existence of an efficient procedure for sampling $L$, yields a trivial solution. However, this is not the case in general, and furthermore, is not the case in the applications that we are interested. The reader should note that using a protocol for random selection one can prove approximate lower bounds. (Random selection implies that if $|S(M',L)| \leq 2^{k-2}$ then $\text{Prob}(\text{the value is error}) \geq \frac{3}{4}$.)

3.1 A Protocol for Random Selection

For the design of the protocol, we consider ordered complete binary trees (i.e. each internal node has a left and right son) of depth $k$. The following notations are used: (1) $\text{Succ}(T,v,d)$ - a function that maps the node $v$, in the tree $T$, to its successor using the outgoing edge marked $d$ (e.g. $\text{Succ}(T,v,\text{right}) = v$'s right son). (2) valued tree - a tree with integers associated to each node, where $\text{value}(v)$ denotes the
integer associated with node $v$. (3) \textit{consistent path} - a path $(v_0, \ldots, v_k)$ from the root to a leaf in a valued tree, $T$, such that for every node $v_i$ if $v_{i+1} = \text{Succ}(T, v_i, \text{right})$ then value($v_i$) is smaller than value($v_j$) for all $j > i + 1$ and if $v_{i+1} = \text{Succ}(T, v_i, \text{left})$ then value($v_i$) is greater or equal to value($v_j$) for all $j > i + 1$. (4) \textit{sorted tree} - a valued tree in which all the paths are consistent.

An overview of the protocol

For sake of simplicity, we present a random selection protocol for the special case that $|L_a| = 2^k$. The extension to the general case is easy (see Remark at the end of section 3.1).

The protocol proceeds as follows: The common input is a pair $(n, 2^k)$. We refer to the program of an honest Merlin which incorporates a set $S$ of size $2^k$ as an additional input. Merlin creates a complete sorted binary tree of depth $k$ with the leaves containing the elements of the set $S$. Arthur's aim is to traverse along a random path from the root to a leaf. Arthur creates the path in $k$ steps: At each step Arthur chooses at random a direction to continue the path (left or right), and Merlin replies with the value of the appropriate node in the tree. This process continues until Arthur has a path of length $k$. Arthur checks whether the path is consistent. If it is inconsistent then the value of the conversation is error, else the value of the conversation equals the value of the last node in the path (i.e. the leaf). This protocol is called \textit{Choose}, and is composed of two interactive programs \textit{Arthur\_Choose} $(n, 2^k)$ (Arthur's program) and \textit{Merlin\_Choose} $(S, n, 2^k)$ (Merlin's program). The reader may either think of Merlin as getting the set $S$ as an auxiliary input, or think of Merlin's program as incorporating this set.

\textbf{Program for Honest Merlin:}

Suppose that $|S| = 2^k$ then Merlin acts as follows. Merlin constructs a complete binary sorted tree, $T_{S_k}$, whose leaves have values in the set $S$. He sets the value of $node_0$ to the root of $T_{S_k}$, and initialize the interaction with Arthur by sending him the value of $node_0$ (the root). In step $i$ ($1 \leq i \leq k$), Merlin preforms the following actions:

Receive \textit{Direction},

$node_i \leftarrow \text{Succ}(T_{S_k}, node_{i-1}, \text{Direction})$

Send value($node_i$)
If $|S| = 2^k$ then Merlin sends Arthur, at each iteration, an error message.

**Program for Arthur:**

Initially Arthur receives from Merlin the value of the root and saves it in $val_0$. In step $i$ ($1 \leq i \leq k$), Arthur performs the following actions:

- Choose at random $Direction_i \in \{left, right\}$
- Send $Direction_i$
- Receive $val_i$

The value of a conversation

The value of a conversation is defined as $val_k$ in case the path is consistent and contains no error messages, and as error otherwise. A path is consistent and contains no error messages if and only if

$$\forall i, j : 0 \leq i < j \leq k, \ (Direction_i = left) \Rightarrow (val_i \leq val_j) \ & \ (val_i \neq error)$$

Clearly, the value of a conversation can be computed in polynomial-time.

Remark: If $|S|$ (which is input to both A and M) is not a power of two then the programs are modified as follows. Merlin adds to $S$ dummy values (larger than any original element in $S$) to form $S'$ so that $|S'|$ is a power of 2, and runs his program using $S'$. At each step, Arthur chooses at random a direction, with probability proportional to the number of non-dummy leaves in the different subtrees. Note that with an unbiased binary coin such a probability can only be approximated (with exponentially decreasing error), and thus the definition of Random Selection should be slightly relaxed.

### 3.2 Analysis of the Protocol

First we consider the case that Merlin is honest. Let $S$ be the input set of Merlin.

**Lemma 1:** If $|S| = 2^k$, and Merlin follows his program Choose_Merlin $(S, n, 2^k)$, then the value of the conversation is uniformly distributed in $S$:

**Proof:** Merlin creates a complete binary sorted tree, and perform the moves, that Arthur requested, on that tree. The value of the conversation is the value of the leaf at the end of the path. All the paths in the tree are consistent, therefore the path that Arthur choose is consistent. All the paths in the tree have the same probability to be chosen by Arthur, hence the value of the conversation is uniformly distributed in $S$. \[\blacksquare\]
We show that no matter how Merlin plays, the probability of a single element to be output is either $2^{-k}$ or zero. This condition ensures that Merlin can not enhance the probability of a single element. For every Merlin, $M'$, we define a binary tree $T^{M'}$. The tree $T^{M'}$ describes Merlin strategy, and is not necessarily kept by $M'$. Each node, at level $i$, corresponds to a possible prefix of $i$ steps in a conversation between $M'$ and $A$. The left (resp. right) son of a node at the $i$-th level, corresponds to the continuation of the conversation when Arthur’s response is $Next=left$ (resp. $Next=right$). The value of a node, $v$, is Merlin’s response to Arthur at that iteration and is denoted by $value(v)$. Note that $T^{M'}$ is a valued tree.

The following technical Lemma plays a central role in the analysis.

Lemma 2: In any valued tree, at each level the values of nodes with consistent paths from the root, are distinct.

Proof: Assume on the contrary that there is a value that appears in some level on two distinct consistent paths. That is, $\exists u_1$ and $u_2$ such that $value(u_1) = value(u_2)$ and both $u_1$ and $u_2$ are on consistent paths and at the same level. Consider the least common ancestor of $u_1$ and $u_2$, denoted $v$. Without loss of generality, we assume that $u_1$ is in the left subtree of $v$ and $u_2$ is in the right subtree. In order for the paths to be consistent, $value(v) \geq value(u_1)$ and $value(v) < value(u_2)$. Clearly this implies that $value(u_1) < value(u_2)$, contradicting the hypothesis that $value(u_1) = value(u_2)$.

Lemma 3: For every $v \in S^{M'}$: $\text{Prob}($the value of an A-M' conversation is $v$$) = 2^{-k}$.

Proof: Recall that $S^{M'}$ is the set of A-M' conversation values. The value of a conversation is not error if and only if the values of $Direction_i$ and $val_i$ correspond to a consistent path (with no error messages) in $T^{M'}$. Since, $T^{M'}$ is a valued tree, by Lemma 2, there is a unique leaf (node at level $k$) with value $v$ and a consistent path from the root. By the one-to-one correspondance between conversations with value in $S^{M'}$ and consistent paths, the Lemma follows.

Combining Lemmas 1 and 3, we get

Theorem 4: The protocol $\text{Choose}$ (of section 3.1) is a random selection Arthur Merlin game.

3.3 Approximate Lower Bound
Following Babai [B] and Goldwasser and Sipser [GS], we use the following definition.

**Definition 6 (Approximate Lower Bound):** Let \( L \subseteq \{0,1\}^* \), \( L_a := L \cap \{0,1\}^a \), and \( l_a = |L_a| \). An approximate lower bound for \( L \) is an Arthur Merlin game (i.e. programs \( A \) and \( M \) and a function \( p \)), with an oracle to membership in \( L \), satisfying the following:

1) The value of a conversation between \( A \) and \( M \), on input \( n \) (in unary) and \( l_a \), is accept with probability at least 2/3.

2) For every \( M' \) and \( l' < \frac{l_a}{2} \), the value of a conversation between \( A \) and \( M' \), on input \( n \) and \( l' \), is accept with probability at most 1/3.

3) The value of a conversation can be computed in polynomial-time when having access to an oracle for membership in \( L \).

An approximate lower bound with perfect completeness is an approximate lower bound for \( L \) in which the value of a conversation between \( A \) and \( M \), on input \( n \) (in unary) and \( l_a \), is always accept.

**Proposition 5:** Let \( (A, M, p) \) be a random selection from a language \( L \), and \( p'(\text{conv}) = \text{accept} \) if and only if \( p(\text{conv}) \in L \). Then \( (A, M, p') \) is an approximate lower bound with perfect completeness for \( L \).

**Corollary 6:** The protocol Choose together with an oracle for membership in \( L \) is an approximate lower bound with perfect completeness for \( L \).

### 4. ARTHUR MERLIN PROOF SYSTEM WITH PERFECT COMPLETENESS

In this section we transform an AM proof system into an AM proof system with perfect completeness.

#### 4.1 The Protocol

We denote the original Arthur by \( \hat{A} \) and Merlin by \( \hat{M} \). Let \( \epsilon \) be the error probability, i.e. for \( x \in L \) the \( \text{Prob}(\hat{A} \text{ accepts}) > 1-\epsilon(|x|) \), and for \( x \notin L \) the \( \text{Prob}(\hat{A} \text{ accepts}) < \epsilon(|x|) \). On input of size \( n \), \( q(n) \) iterations are performed, at each iteration Arthur sends a message of length \( m(n) \). When clear from the text we use \( \epsilon, q, m \) for \( \epsilon(n), q(n), m(n) \), respectively. Without loss of generality we assume that \( \epsilon < 2^{-n^2} \).

This can be achieved by performing sufficiently many copies of the original Arthur Merlin game in parallel, and ruling by the majority (see [B], [GS] and [BHZ]).
An overview of the protocol

The main idea is to simulate the original AM game, in a manner avoiding the conversations in which Merlin fails to convince Arthur of $x \in L$. This is done by Merlin restricting Arthur's choices to subsets favorable to Merlin. Rather than choosing from the set $\{0,1\}^m$, Arthur chooses a $m$-bit string from a subset of size $2^{m-1}$. Arthur's choice is determined by the value of the \textit{Choose} protocol, and is thus guaranteed that any string has exactly probability $2^{m-1}$ (provided that the value is not error). This choice is known to both Arthur and Merlin. Merlin now responds to this choice. At the end Arthur has a conversation, and decides to accept or reject essentially by evaluating $\hat{\rho}$ on the conversation (namely his decision is the same as the one of the original $\hat{\Lambda}$'s). We need to show that: (1) For any $x \in L$ Merlin can choose subsets such that he never fails, and (2) For every $x \in L$, no Merlin can fool Arthur into accepting (except for a low probability).

The Computation Tree of an Arthur Merlin Game

Let $A$ and $M$ be programs of an Arthur Merlin game so that on input $x$ they exchange at most $2h$ messages each of length $l$. The computation tree of $A$ and $M$ on input $x$ is a complete valued tree with out-degree $2^l$ and height $h$. The nodes are labeled by Merlin's messages and the edges are labeled by Arthur's messages. Each path in the tree correspond to a possible conversation between $A$ and $M$. The conversation corresponding to the path $v_0 \rightarrow v_1 \cdots \rightarrow v_k$ is $r_1y_1 \cdots r_k y_k$, where node $v_i$ is marked by $y_i$ and the edge $v_{i-1} \rightarrow v_i$ is marked by $r_i$. We number the levels of the tree beginning at the root (level zero), and ending at the leaves (level $h$).

Program for an honest Merlin:

Merlin's program consists of two stages. First, Merlin computes the sets of choices (for Arthur) that are favorable to him. The second stage is a simulation of the original Arthur Merlin game (denoted by $\hat{A}\hat{M}$) when Arthur's choices are restricted to these subsets.

Preprocessing stage

Let $T$ be the computation tree of the original $\hat{A}\hat{M}$ game on input $x \in L$. The aim of Merlin is to find a subtree of $T$ such that there are no leaves that represent conversations that cause Arthur to reject and each internal node has out-degree $2^{m-1}$. The subtree is computed in the following manner. First Merlin deletes
from $T$ all the leaves that represents conversation that cause Arthur to reject. Next, Merlin performs node pruning level by level, going from the leaves to the root. At each level $i$ Merlin prunes all the nodes with out-degree less than $2^{m-1}$. The preprocessing is said to have failed, if after the pruning of nodes the final tree is empty, in such a case Merlin enters an error mode (and sends error messages to Arthur in response to any future message). If the tree is not empty, Merlin chooses a subtree of $T$, such that all the internal nodes has out-degree equal $2^{m-1}$, and saves it as $T$ (This is done by a second pruning.)

**Simulation stage**

Assuming that Merlin did not enter the error mode the simulation of the original protocol proceeds as follows. The protocol follows the iterations of the original $(Ahat, M)$ game. The current node, is the node in $T$ that represents the prefix of the conversation of $AhatM$, up to the present iteration. At iteration $i$, Merlin restricts Arthur choice to a set of values, denoted by $S_i$. These values are the labels of the ($2^{m-1}$) outgoing edges of the current node. After receiving Arthur's random choice $r_i$, Merlin updates the current node, computes his response, $y_i$, and sends it to Arthur. Formally, at each iteration $i (1 \leq i \leq \lg n)$ Merlin preforms:

\[
S_i \leftarrow \{ r \mid \exists u \in T : u = \text{SUCC}(T, \text{current node}, r) \}.
\]

\[
r_i \leftarrow \text{Merlin Choose} (S_i, m-1).
\]

\[
\text{current node} \leftarrow \text{SUCC}(T, \text{current node}, r_i).
\]

\[
y_i \leftarrow M(x, r_{i-1}, r_i) \quad (y_i \text{ is the label of current node})
\]

Send $y_i$

In case Merlin enters the simulation stage in the error mode he answers all Arthur messages by an error message (i.e. $y_i = \text{error}$ for all $i$).

**Arthur's program:**

Arthur protocol is identical to the Arthur original protocol, with the difference that instead of choosing the next string locally (at random), it is chosen through the Choose protocol. Formally, for each iteration $i (1 \leq i \leq \lg n)$ Arthur performs:

\[
r_i \leftarrow \text{Arthur Choose}(m-1).
\]

Receive $y_i$

**The value of a conversation**
The value of a conversation is determined by the following polynomial-time predicate
\[ \hat{p}(r_1, y_1, \ldots, r_q, x_1, \ldots, x_q) \land \forall i (r_i \neq \text{error} \land y_i \neq \text{error}) \]

4.2 The Perfect-Completeness of the protocol

We show that if the input \( x \) is in \( L \), then an honest Merlin always convinces Arthur.

Lemma 7: If \( x \in L \) then an honest Merlin does not enter the error mode in the preprocessing phase.

Proof: Merlin enters the error mode in the preprocessing phase if and only if the pruned computation tree is empty (i.e. all the nodes were deleted from the tree). In the original \( \hat{A}M \) protocol, for inputs \( x \) such that \( x \in L \), \( M \) convinces \( A \) with probability \( 1 - \epsilon \). This implies that the number of leaves in \( T_{AM} \) (the computation tree of \( \hat{A}M \)) that cause Arthur to reject is at most \( \epsilon 2^m \). By backward induction on \( i \), we show that the number of nodes pruned at level \( i \) is at most \( 2^i \epsilon 2^m \).

The induction base \((i=q)\) holds since the number of rejecting leaves is \( \leq \epsilon 2^m \). For the induction step, assume that at level \( i \) at most \( 2^i \epsilon 2^m \) were pruned. At the next level (i.e. level \( i-1 \)) we prune all the nodes with half or more of their sons missing. Every node that is pruned has at least \( 2^{m-1} \) pruned sons. Thus, the numbers of nodes that are pruned at level \( i-1 \) is bounded by
\[ \frac{2^{i-1} \epsilon 2^m}{2^{m-1}} = 2^{i-1} \epsilon 2^{m-1}. \]

The number of nodes pruned at level \( i \) is at most \( 2^{i-1} \epsilon 2^m \). Recalling that \( \epsilon < 2^{-d} \) the Lemma follows.

Lemma 8: If \( x \in L \) then Arthur always accepts.

Proof: By Lemma 7, Merlin can create from the computation tree of the original Arthur Merlin game a subtree in which all internal nodes are of degree \( 2^{m-1} \) and all the paths represents accepting conversations. It follows that each of the sets \( S_i \) in the simulation stage has cardinality \( 2^{m-1} \). By Lemma 1, the conversations of \( \text{Choose} \left( S_i, m-1 \right) \) always have value in \( S_i \) (i.e. they never evaluate to error). The Lemma follows.

4.3 The Probabilistic Soundness of the protocol
We now show that for every input $x$ not in $L$, no matter what Merlin does, the probability that he convinces Arthur is less than $1/4$.

**Lemma 9:** If $x \notin L$ then $\Pr(\text{Arthur accepts } x) \leq \frac{1}{4}$.

**Proof:** Consider the execution of the original and simulated games on $x \notin L$. In the original game, the number of accepting conversations is bounded by $\varepsilon 2^m$ no matter how Merlin plays. For every Merlin playing in the simulating game there exists a Merlin for the original game such that each accepting conversation in the simulated game corresponds to an accepting conversation in the original game. This correspondence maps the simulated accepting conversation to the subsequence of $r_j$'s and $y_j$'s appearing in it, ignoring the internal messages of the executions of the *Choose* protocol. We stress that in an accepting conversation of the simulated game neither of the $r_j$'s or $y_j$'s is error. It follows that the number $\varepsilon$ accepting conversations in the simulated game is $\leq \varepsilon 2^m$.

By Lemma 3, for every $i$, every prefix $r_{j-1}y_{j-1}$, and every $\alpha \in \{0,1\}^m$, the probability that $r_i = \alpha$ is either $2^{-m+1}$ or 0. Therefore, the probability of every accepting conversation in the simulated game is $(2^{-m+1})^{\varepsilon}$. Hence the probability of Arthur accepting is bounded by $\varepsilon 2^m \cdot 2^{-m+1}$, which equals $\varepsilon 2^{m-1}$. Recalling that $\varepsilon \leq \frac{2}{3}$, the Theorem follows. $\square$

### 4.4 Main Result

Using the equivalence of interactive proofs and Arthur Merlin proofs [GS], and combining Lemmas 8 and 9 we get

**Main Theorem (Theorem 10):** If a language $L$ has an interactive proof system then $L$ has an (Arthur Merlin) interactive proof system with perfect completeness. $\square$

### 5. INTERACTIVE PROOF SYSTEMS WITH PERFECT SOUNDNESS

In the previous section, we showed that interactive proofs can be modified so that the verifier always accepts valid statements. What happens if we require that the verifier never accepts false statements? In this case we show that the set of languages recognized equals NP.

The reader should note that the transformation of Goldwasser and Sipser [GS] does not preserve perfect completeness. Thus it is not clear that proving the above statement with respect to Arthur Merlin
games yields the same result with respect to interactive proofs. The difficulty can be resolved by modifying the transformation of [GSJ], using our approximate lower bound protocol. We prefer to give a direct proof considering interactive proofs themselves.

The difference between interactive proofs and Arthur Merlin games is that in interactive proofs the verifier’s $i$-th message $\alpha_i$ is a function of the input $x$, his random coin tosses $r$, and the previous messages of the prover (i.e. $\alpha_i = V(x,r,y_1...y_{i-1})$). After the last (say $q$-th) iteration, the verifier decides whether to accept or reject by evaluating the polynomial-time predicate $p(x,r,y_1...y_q) \in \{\text{accept, reject}\}$.

Theorem 11: If a language $L$ has an Interactive Proof with perfect soundness then $L \in \text{NP}$

Proof: Assume that for a language $L$, there exists an interactive proof with perfect soundness. Since the verifier is limited to probabilistic polynomial time, then for any input $x \in L$ there is a conversation that convinces him, and is of polynomial length. The NP machine guesses this conversation, checks that it is indeed a legitimate one and that it leads the verifier to accept. Namely, the machine guesses a random tape $r$ and a conversation $\alpha_0y_1...\alpha_qy_q$, and checks that $\alpha_i = V(x,r,y_1...y_{i-1})$ (for every $i$) and that $p(x,r,y_1...y_q) = \text{accept}$.

If $x \in L$ then, by the probabilistic completeness condition, there exist (many) accepting conversations. If $x \notin L$ then, by the perfect-soundness condition, there is no such conversation, and any guess of the machine will fail. $\square$

6. CONCLUDING REMARKS

We have showed how to transform an arbitrary Arthur Merlin game into an Arthur Merlin game with perfect-completeness. The transformation has the disadvantage of increasing the number of iteration substantially. The resulting protocol has $mq$ iterations, where $q$ is the number of iterations in the original Arthur Merlin game and $m$ is the size of a messages in that game. The following modification results in the reduction of the total number of iterations. The Merlin Choose protocol is modified such that Merlin sends as his replay a balanced subtree of size $m$, whose root is the current node of Merlin. The Arthur Choose protocol is modified such that Arthur chooses a random path in the $\log m$-depth subtree, record the last node and sends it to Merlin. The number of iterations, of each Choose protocol, is reduced by a factor of $\log m$, reducing the total number of iterations to $\frac{mq}{\log m}$ iterations (instead of $mq$). A goal for
future research is to find a transformation that uses substantially less iterations, as a function of \( m \) and \( q \).

In particular, is it possible to implement random selection in a fixed number of rounds?

Babai [B] showed that any Arthur-Merlin game with a fixed number of interactions can be simulated by a game with two interactions. A similar proof applies to the hierarchy of interactive proofs with perfect completeness.

Assuming the existence of secure encryption functions (in the sense of [GM]) and using the results of [GMW], one can easily demonstrate the existence of zero-knowledge interactive proofs with perfect completeness for every language in \( IP \). However, it is not clear whether every language having a perfect (resp. almost perfect) zero-knowledge interactive proof (see [F] for definition) has a perfect (resp. almost perfect) zero-knowledge interactive proof with perfect completeness.
REFERENCES


