Appears in SIAM J. on Computing, Vol. 23, No. 1, February 1994.

The Complexity of Decision versus Search

Mihir Bellare* Shafi Goldwasser†

September 1992

Abstract

A basic question about NP is whether or not search reduces in polynomial time to decision. We indicate that the answer is negative: under a complexity assumption (that deterministic and non-deterministic double-exponential time are unequal) we construct a language in NP for which search does not reduce to decision.

These ideas extend in a natural way to interactive proofs and program checking. Under similar assumptions we present languages in NP for which it is harder to prove membership interactively than it is to decide this membership, and languages in NP which are not checkable.

Keywords: NP-completeness, self-reducibility, interactive proofs, program checking, sparse sets, quadratic residuosity.

* Department of Computer Science & Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093. E-mail: mihir@cs.ucsd.edu. Part of this work was done while the author was at MIT, partially supported by NSF grant No. CCR-8719689 and DARPA grant No. N00014-89-J-1988.

† MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA. E-mail: shafi@theory.lcs.mit.edu. Partially supported by NSF grant 25801, DARPA 71949, the Princeton computer science department, and grant No. 86-00301 from the United States - Israel Binational Science Foundation (BSF), Jerusalem, Israel.

1 Introduction

The work on interactive proofs brought to light a basic question: how powerful does a prover need to be to convince a verifier of membership in a language L?

Clearly, the prover needs at least the power to decide the language for himself. The question we focus on is whether this is enough.

There are interactive proofs known for complete problems in NP, $\mathrm{P}^{\#\mathrm{P}}$ and PSPACE where it is sufficient for the prover to be able to decide membership in the language. Such power is also sufficient for almost all of the languages in IP that have been closely examined (specifically, the languages of graph isomorphism, graph non-isomorphism [GMW], and quadratic non-residuosity [GMR]). On the other hand all known interactive proofs for complete languages for coNP require the prover to do more than decide membership in the language. Similarly, all known interactive proofs for the language of quadratic residuosity require the prover to do more than decide quadratic residuosity.

As we will see, this is essentially a generalization of the old question of whether search problems reduce to their decision counterpart for NP. Namely, is computing a witness for membership in $L \in \mathrm{NP}$ any harder than establishing the existence of such a witness? For NP-complete problems it is well known that the answer is no: given an oracle for membership a witness can be computed in polynomial time. But for general $L \in \mathrm{NP}$ the problem remains open.

In this paper, we use natural complexity assumptions to indicate that proving membership may be harder than deciding it. As a first example we look at decision versus search in NP. We then turn to interactive proofs, and finally apply the same ideas to derive results on the difficulty of program checking.

Let us proceed to describe our results in detail.

1.1 Decision Versus Search in NP

Before we can present our results, we need to say what we mean by "search," "decision," and the "reduction" of the former to the latter. We will keep the discussion here informal; for formal definitions we refer the reader to Section 2.1. We start with some terminology.

Suppose $\rho(\cdot,\cdot)$ is a polynomial time computable binary relation. We let $\rho(x) = \{w : \rho(x,w) = 1\}$ be the set of all ρ-witnesses for x. We say that ρ is an NP-relation if there exists a constant c such that for all $x \in \{0,1\}^*$ it is the case that $\rho(x) \subseteq \{0,1\}^{|x|^c}$. We let $L_\rho = \{\, x \in \{0,1\}^* : \rho(x) \neq \emptyset \,\}$.

Now let $L \subseteq \{0,1\}^*$ be a language. We say that ρ defines L if $L = L_\rho$. Clearly, $L \in \mathrm{NP}$ iff there exists an NP-relation which defines L. It is important to note, however, that for any particular NP language L, there are many (different) NP-relations which define it.

Associated to any NP language L is a (single) decision problem and a class of search problems. The decision problem, of course, is just the problem of deciding membership in L. As for the search problems, there is one for each NP-relation which defines L, and the search problem corresponding to a particular NP-relation ρ which defines L is the following: given $x \in L$, find a ρ-witness for x. For example, if L is SAT then the decision problem is to decide whether or not a given formula is satisfiable. One of the associated search problems is to determine a truth assignment of a given satisfiable formula (but there are other associated search problems as well).

We are interested in defining what it means for search to reduce to decision for L. As a means to obtaining the definition and understanding the issues involved, we begin by discussing a less general notion: that of reducing search to decision for an NP-relation ρ (defining L).

Fix a particular NP-relation ρ which defines L. We say that "search reduces to decision for ρ" if the search problem for ρ is solvable in polynomial time given an oracle for the decision of $L = L_\rho$.

More precisely, search reduces to decision for ρ if there exists a polynomial time oracle machine W such that for all $x \in L$ it is the case that $W^L(x)$ (the output of W with oracle L and input x) is a ρ-witness for x. Intuitively, the search problem for ρ is no harder than the decision problem for the corresponding language.

We are now ready to state what it means for search to reduce to decision for a *language* $L \in$ NP. We recall that there are many different NP-relations defining L. In general, search might reduce to decision for some of these and not for others. Our definition is to say that search reduces to decision for L has long as there is *some* NP-relation ρ (defining L) such that search reduces to decision for ρ. In other words, we say that search reduces to decision for L as long as at least one of the (many different) search problems associated to the decision problem for L is no harder than this decision problem.

The motivation for this definition, which stems from the question of whether proving membership can be harder than deciding it, will become clearer as we go on. For the moment, it is more important to stress the generality of our definition and the strength of negative conclusions that are based on it. In particular, to say that search does *not* reduce to decision for a particular language L (as in the conclusion of the theorem that follows) is to make a strong statement indeed, because it means that for *all* ρ defining L it is the case that search does not reduce to decision for ρ. That is, *all* the search problems corresponding to the decision problem of L are harder than this decision problem. In particular, the existence of a language for which search does not reduce to decision certainly implies the existence of an NP-relation for which search does not reduce to decision.

To state the theorem we first need the following definitions:

$$\mathrm{EE} \;=\; \bigcup_{c\geq 0}\mathrm{DTIME}(2^{c2^n}) \quad \text{and} \quad \mathrm{NEE} = \bigcup_{c\geq 0}\mathrm{NTIME}(2^{c2^n})\,.$$

Theorem 1.1 *Suppose* EE $\neq$ NEE. *Then there is a language in* NP *for which search does not reduce to decision.*

Note that the conclusion (of the above theorem) implies P $\neq$ NP. Whether the assumption could be reduced to P $\neq$ NP (or even E $\neq$ NE) remains an open question.

We note that if L is NP-complete, then for *any* NP-relation ρ that defines L, it is the case that search reduces to decision for ρ (a consequence of the "self-reducibility" and NP-completeness of SAT as well as certain features of the proof of Cook's theorem [Co], this fact is one of the most basic and well-known ones in the theory of computation). In particular, by our definition, search certainly reduces to decision for any NP-complete language. So the "hard" problems (from the point of view of search versus decision) will necessarily be non-NP-complete. In particular, the language of the conclusion of the above theorem is not NP-complete.

The decision versus search question has attracted the attention of researchers ever since NP was introduced (we survey some of the work on this subject in Section 1.5). However, we note that previous work has focused on the question of whether search reduces to decision for NP-relations (not NP languages) and the conclusions have been weaker than ours.

1.2 Competitive Proof Systems: the Natural Extension

NP represents the simplest kind of proof system. An NP proof system for L is defined by a polynomial time "verifier" V. This verifier talks to a "prover" who, on an input x common to both parties is allowed to send the verifier a single message of length polynomial in $n = |x|$. As a function of this message and the common input, the verifier decides whether or not to accept (this "decision" of the verifier is a polynomial time binary predicate ρ evaluated on the common input and the prover's message). In the case that $x \in L$, there must exist some *deterministic* "prover"

P who can convince the verifier to accept (this is the "completeness" condition). In the case that $x \notin L$, no "prover" should be able to convince the verifier to accept (this is the "soundness" condition). We usually specify an NP proof system by a pair (P, V) where P is a prover satisfying the completeness condition. Clearly, $L \in$ NP if and only if it possesses an NP proof system.

How powerful need the prover P be in an NP proof system for L? It is clear he must have at least the ability to decide L for himself. Let us call a NP proof system (P, V) "competitive" if this "minimal" ability is also sufficient; more precisely, (P, V) is *competitive* if P runs in polynomial time given an oracle for L. It now becomes clear that the question of whether or not search reduces to decision for $L \in$ NP captures the computational difficulty of the prover's task under this "competitive" measure of complexity. More precisely, we observe that L has a competitive NP proof system if and only if search reduces to decision for L. Thus, Theorem 1.1 indicates that there is a language $L \in$ NP to give an "NP proof" of which any prover must use power over and above that necessary to decide L.

NP-proof systems, however, are very restrictive. It becomes natural to ask: would the prover's task be alleviated if the parties were allowed interaction and the proof was now only required to be correct with high probability? In other words, we now consider interactive proofs (cf. [GMR]). We recall that in an interactive proof both parties are allowed to be probabilistic and the parties are allowed to exchange messages, for a polynomial number of rounds, before the verifier decides whether or not to accept. Completeness and soundness are required to hold only with high probability (see Section 4.1 for precise definitions). Let us call an interactive proof system *competitive* if the prover[†] runs in probabilistic polynomial time given access to L as an oracle. Then does every language in NP (and more generally in IP) have a competitive interactive proof? In other words, does the extra leeway provided by interaction and randomness reduce the burden on the prover, or does the discrepancy between proving and deciding remain even if coins and interaction are allowed?

Quadratic residuosity provides a telling example. Let $QR = \{\, (x, N) : \exists y \in Z_N^* \text{ s.t. } x \equiv y^2 \pmod N \,\}$, and $QNR = \{\, (x, N) : \neg\exists y \in Z_N^* \text{ s.t. } x \equiv y^2 \pmod N \,\}$. Search is not known to reduce to decision for QNR; in all known NP proof systems for QNR, the prover requires the ability to factor N, and factoring is not known to be reducible to quadratic residuosity. Yet, we do know of *interactive* proofs for $\in QNR$ where it suffices for the prover to be able to tell membership in QR (i.e., QNR does have competitive interactive proofs) [GMR]. On the other hand, there is no known interactive proof for QR where it suffices for the prover to be able to decide membership in QR (i.e QR is *not* known to have a competitive interactive proof).

Our next result indicates that in general, interaction and randomness will not make the prover's task easier. More precisely, we indicate that not all languages in NP have competitive interactive proofs. Letting BPEE denote the class of languages recognized with bounded error by a probabilistic TM running in time 2^{c2^n} for some constant $c \geq 0$, we have the following

Theorem 1.2 *If* NEE $\not\subseteq$ BPEE *then there is a language in* NP *that does not have a competitive interactive proof.*

The complexity of a prover in an interactive proof system is a basic question which is attracting a fair amount of attention (cf. Section 1.5). The notion of competitive interactive proofs that we introduce provides a new angle from which to understand this question; whereas past work has focused on providing upper bounds on the complexity of provers, we are instead trying to understand the "comparative" complexity of proving versus deciding.

[†] The prover here refers, of course, to the "honest" prover of the completeness condition; the soundness condition of the proof system is as usual required to hold with respect to any (computationally unbounded) prover.

1.3 Program Checking

We briefly mention our results on program checking that are in the same vein as the above.

Blum and Kannan [BK] introduced the notion of program checkers (see Section 5 for full definitions). Negative results in this domain begin with Yao [Ya] who presented a language in deterministic space $2^{n^{\log\log n}}$ that does not have a checker. Beigel and Feigenbaum [BF] and Krawczyk [Kr] improved this to deterministic space $n^{\log^* n}$. The question of whether there are languages of reasonable complexity that are not checkable was answered by [BF] under an assumption: they showed there was one such in NP provided non-deterministic *triple* exponential time is not contained in bounded probabilistic *triple* exponential time. We improve the assumption to *double* exponential time. Namely, we have the following

Theorem 1.3 *If* NEE $\not\subseteq$ BPEE *then there is a language in* NP *that does not have a checker.*

1.4 A Natural Candidate?

Clearly, it would be most interesting to exhibit an example of a *natural* problem in NP for which there are no competitive interactive proofs. A candidate example — as we indicated in Section 1.2 — is the quadratic residuosity problem. Let us consider the interactive proofs known for membership both in QR and QNR in more detail.

First for membership of (x, N) in QNR, the following protocol is repeated k times (cf. [GMR]). The verifier tosses a coin $c \in \{0, 1\}$. If $c = 1$ then the verifier sends the prover $z = xr^2 \bmod N$ for random $r \in Z_N^*$, else the verifier sends the prover $z = r^2 \bmod N$ for random $r \in Z_N^*$. The prover is then asked to guess the value of c. If the prover guesses correctly in each repetition of the protocol, then the verifier accepts. Clearly, if $(x, N) \in QNR$ then $c = 1$ if and only if $(z, N) \notin QR$. Thus, it is sufficient for the prover to be able to tell membership in QR in order to guess c and the prover is competitive. On the other hand, if $(x, N) \notin QNR$ then the probability that the verifier accepts is no greater than 2^{-k}.

How about proving that $(x, N) \in QR$? A simple proof would be the factorization of N or a $y \in Z_N^*$ such that $x \equiv y^2 \pmod{N}$. In fact, all known interactive proofs of this fact require the ability to factor N. As we do not know whether factoring reduces to deciding quadratic residuosity, it remains an intriguing open problem whether membership in QR can be interactively proved by a probabilistic polynomial time prover with access to a QR oracle. In particular, if the answer to this question were negative, we would get the following interesting number theoretic implication: integer factorization is not polynomial time reducible to deciding quadratic residuosity.

We note that there are special classes of integers N for which $(x, N) \in QR$ can be interactively proved by a probabilistic polynomial time prover with access to a QR oracle. For example, this can be done when N is the product of a constant number of primes (cf. Section 6).

1.5 Related Work

We discuss related work on decision versus search and the complexity or provers.

Decision versus Search for NP. The "decision versus search" question has attracted the attention of researchers ever since NP was introduced. It has been studied in many different contexts and from many angles, and, in particular, many results have indicated that for NP, search is likely to be harder than decision. We stress that all these results are about search versus decision for NP-relations, not NP-languages. So, in that sense, the conclusions are weaker than ours.

Let us now describe some of this work. In what follows we let ρ denote a polynomial time computable binary predicate.

Valiant [Va] appears to have been the first to indicate that there are NP-relations ρ for which search is unlikely to reduce to decision; specifically, assuming $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, he presents a particular NP-relation ρ with the property that search does not reduce to decision for ρ. However, the underlying language L_ρ in Valiant's result is easy; in fact, it equals $\{0,1\}^*$. Borodin and Demers [BD] strengthen Valiant's result in this regard by showing that under the same assumption, there is a NP-relation ρ for which $L_\rho \in \mathrm{NP} - \mathrm{P}$ but search still does not reduce to decision for ρ. Hartmanis and Hemachandra [HH] present results similar to Valiant's but assuming $\mathrm{P} \neq \mathrm{UP} \cap \mathrm{coUP}$.

Impagliazzo and Naor [IN] indicate that, at least in relativized worlds, the assumption $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ is not necessary for the conclusion of Valiant's result. More precisely, they present a relativized world in which $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$ but there exists an NP-relation ρ such that $L_\rho = \{0,1\}^*$ and search does not reduce to decision for ρ.

The assumption $\mathrm{P} \neq \mathrm{NP}$ suffices to indicate that the "usual" method of self-reduction (where one constructs a witness bit-by-bit, given an oracle for the language) may not always work: Selman [Se] shows that under this assumption there is a NP-relation ρ for which $L_\rho \in \mathrm{NP} - \mathrm{P}$ but, given a pair of strings (x, u) and an oracle for L_ρ, it is impossible to decide in polynomial time whether or not there is an extension of u which is a ρ-witness for x.

We stress again that none of the above work addresses the problem we consider. They focus on NP-relations, asking whether there exist (specific) NP-relations ρ for which it is impossible to reduce the search problem for ρ to the corresponding decision problem for L_ρ. We focus on NP languages, asking whether there exists a language L, for which, for *any* associated search problem ρ, it is impossible to reduce the search problem for ρ to its corresponding decision.

Decision versus Search in other settings. The usual reduction of search to decision has a strong sequential flavor, and Karp, Upfal and Wigderson [KUW] investigated the degree to which this is necessary. Ben-David, Chor, Goldreich and Luby [BCGL] investigate the "decision versus search" question in the context of average-case complexity. Impagliazzo and Tardos [IT] consider the decision versus search question in the exponential case, and present an oracle relative to which $\mathrm{E} = \mathrm{NE}$ but there is an exponential time binary predicate whose search problem is not solvable in exponential time.

The Complexity of Provers. Several recent works present results on the complexity of provers in interactive proofs. Let us describe some of them.

Shamir's result [Sh] implies that polynomial space provers suffice to prove PSPACE languages. The best upper bound on the complexity of a prover of a coNP language, due to Lund, Fortnow, Karloff and Nisan [LFKN], is probabilistic, polynomial time with a #P oracle.

Bellare and Petrank [BP] investigate the complexity of zero-knowledge (ZK) provers, and indicate that such provers can be reasonably efficient; specifically, they show that any language possessing a statistical ZK interactive proof possesses one with a prover who is a probabilistic, polynomial time machine with access to an NP oracle.

In the case of multi-prover proofs, Babai, Fortnow and Lund [BFL] show that exponential time provers suffice for exponential time languages.

We stress that all these works are concerned with upper bounding the complexity of provers in an "absolute" sense. The model of competitive interactive proofs that we introduce here is for the purpose of studying the complexity of provers in a different way; namely, in terms of the "comparative" complexity of proving versus deciding.

Recent Work. Independently of this work, Impagliazzo and Sudan [IS] show that if $\mathrm{NE} \neq \mathrm{coNE}$ then there is a language in NP for which search does not reduce to decision. Here the conclusion is the same as in Theorem 1.1, but the assumption is different (and not known to be either weaker or stronger). They also show that if $\mathrm{E} \neq \mathrm{NE}$ then there is a NP-relation ρ for which search does not reduce to decision; this is the same conclusion as in the above-mentioned result of Borodin and Demers [BD], but under an assumption which is different from that of [BD] (but, again, not known to be either weaker or stronger). Finally, Spielman [Sp] has constructed an uncheckable set in Σ_2^{P} under the assumption that $\Sigma_2^{\mathrm{E}} \neq \Pi_2^{\mathrm{E}}$.

Publication Notes. These results appeared in a preliminary form in [BG]. Later, merged with [BF], they appeared in [BBFG].

1.6 Relations to other Notions

We focus in this paper on (competitive) interactive proofs and checking. Related notions are function-restricted interactive proofs [BK], multi-prover interactive proofs [BGKW] and coherence [Ya]. Here we discuss how these notions relate to ours and also how our results impinge on them. First, let us list the (complexity classes corresponding to) the notions in this area.

The Complexity Classes in this area. The following are the (main) complexity classes that the ensuing discussion will focus on.

$$\begin{aligned}
\mathrm{compNP} &= \{\, L : L \text{ has a competitive NP-proof system} \,\} \\
&= \{\, L \in \mathrm{NP} : \text{search reduces to decision for } L \,\} \\
\mathrm{IP} &= \{\, L : L \text{ has an interactive proof system} \,\} \\
\mathrm{compIP} &= \{\, L : L \text{ has a competitive interactive proof system} \,\} \\
\mathrm{frIP} &= \{\, L : L \text{ has a function-restricted interactive proof system} \,\} \\
\mathrm{MIP} &= \{\, L : L \text{ has a multi-prover interactive proof system} \,\} \\
\mathrm{Check} &= \{\, L : L \text{ is checkable} \,\} \\
\mathrm{Coh} &= \{\, L : L \text{ is coherent} \,\}
\end{aligned}$$

Function-restricted interactive proofs. Function-restricted interactive proof systems are a variant of interactive proof systems introduced by Blum and Kannan [BK]. Like competitive interactive proofs, they make the restriction that the honest prover be a probabilistic, polynomial time machine with access to an oracle for the language in question. But, in contrast to competitive interactive proofs, they also restrict the dishonest prover. Specifically, they ask that a dishonest prover be a function from verifier messages to strings. In particular, the response of the dishonest prover to a verifier message is not allowed to depend on previous questions of the verifier (that is, its messages are independent of the history).

As we will see (cf. Lemma 4.3) it is the case that $\mathrm{compIP} \subseteq \mathrm{frIP}$. Blum and Kannan also established that $\mathrm{Check} \subseteq \mathrm{frIP}$. On the other hand, based on the techniques of [FRS], one can show that $\mathrm{frIP} \subseteq \mathrm{MIP}$.

Function-restricted interactive proofs were introduced in order to relate program checking to interactive proofs. We introduce competitive interactive proofs to address the question of how much power is necessary for the honest prover to prove membership interactively, whence it is imperative to not weaken the definition of interactive proofs by making assumptions on the power of the dishonest prover. However, we use the notion of function-restricted proof systems to provide a unified treatment of our results. We establish the main technical lemmas needed for our proofs in terms of the equivalent notion of "deciders" (cf. Section 3), and then use these lemmas to derive our results on competitive interactive proofs and checking in a simple way. In particular, (improving [BF]; see below) we do show that if NEE $\not\subseteq$ BPEE then there is a language in NP $-$ frIP.

Competitive multi-prover proofs. One could define competitive multi-prover proofs. However, using techniques of [FRS], one can show that the corresponding class of languages is identical to frIP.

Coherence. The notion of coherence was introduced by Yao [Ya]. Informally, a language L is *coherent* if the membership of x in L can be decided in probabilistic polynomial time and bounded error by a machine (called the *examiner*) which has access to L as an oracle but is allowed to query this oracle only on points different from x. If L is not coherent we say it is incoherent. Beigel and Feigenbaum [BF] prove the existence of incoherent languages in NP under the assumption that non-deterministic *triple* exponential time is not contained in bounded probabilistic *triple* exponential time. Since checkable languages and languages in frIP are coherent (cf. [Ya, BF]), they thereby establish the existence of uncheckable languages in NP, and languages in NP $-$ frIP, under the same assumption.

One can show that if search reduces to decision for L, or L has a competitive interactive proof then also L is coherent. So the construction of incoherent sets yields negative results about these notions as well. However, the fact that our stronger results on all these notions (namely decision versus search, competitive interactive proofs, checking and function restricted proofs) are obtained more directly (i.e. avoiding incoherence) indicates that coherence may not be the best approach to negative results in this area.

We note that, intuitively, coherence has a flavor different from that of the other notions we have considered; while the common underpinning of these others is the notion of a "proof" (interactive or non-interactive), coherence is not a form of "proof." Indeed, the examiner gives no "proof" that $x \in L$ — there is no guarantee as to what would happen if the examiner is run with an oracle different from L. It is by exploiting this "proof" like quality of the notions we consider that we were able to derive results that are stronger than those derived by the coherence approach.

In this context we note also that one can separate the classes of checkable and coherent languages inside NP, assuming NEE $\not\subseteq$ BPEE (cf. Theorem 5.4).

Summary. We summarize relationships amongst the various complexity classes we have discussed. First, some notation. We define the triple exponential time class

$$\mathrm{NEEE}^* \;=\; \bigcup_{c\geq 0} \mathrm{NTIME}(2^{2^{2^{n^c}}}) \,.$$

Similarly, we let BPEEE* denote the class of languages recognized with bounded error by a probabilistic TM running in time $2^{2^{2^{n^c}}}$ for some constant $c \geq 0$.

The following inclusions are known, or easily derived from known techniques:

(1) compNP $\subseteq$ NP $\cap$ compIP.

(2) NP $\cup$ compIP $\subseteq$ IP $\subseteq$ MIP.

(3) compIP $\cup$ Check $\subseteq$ frIP $\subseteq$ MIP $\cap$ Coh.

Under the assumption NEE $\not\subseteq$ BPEE we establish the following:

(4) NP is not contained in any of the following: compNP, compIP, Check, frIP.

(5) NP $\cap$ Coh is not contained in any of the following: compNP, compIP, Check, frIP.

For the results of line (4), see Theorems 2.9, 4.4, 5.3 and 3.6. For those of line (5), see the (theorem and) discussion at the end of Section 5. Finally, under the assumption NEEE* $\not\subseteq$ BPEEE*, [BF] establish the following:

(6) NP $\not\subseteq$ Coh.

2 Decision versus Search in NP

In this section we present a simple construction of a language in NP for which search does not reduce to decision, assuming that EE $\neq$ NEE. In later sections we will extend the argument to interactive proofs and program checking. Let us begin with the definitions.

2.1 Definitions

The goal of this section is to make precise what we mean by "search reduces to decision for an NP language L." Since the issues were discussed at length in Section 1.1, we will here be brief, stating the (formal) definitions and limiting the discussion to essentials.

It is convenient to proceed in steps. We begin by defining NP-relations and saying what it means for search to reduce to decision for them. We then use this this to say what it means for search to reduce to decision for a NP language.

Definition 2.1 *Let $\rho(\cdot,\cdot)$ be a polynomial time computable binary relation, and let $x \in \{0,1\}^*$. We let $\rho(x) = \{\, w \in \{0,1\}^* : \rho(x,w) = 1 \,\}$, and call the members of this set* ρ-witnesses *for x. We say that ρ is an* NP-relation *if there exists a constant $c \in \mathsf{N}$ such that for all $x \in \{0,1\}^*$ it is the case that $\rho(x) \subseteq \{0,1\}^{|x|^c}$. The* language defined by ρ *is $\{\, x \in \{0,1\}^* : \rho(x) \neq \emptyset \,\}$ and is denoted L_ρ.*

Note that if ρ is an NP-relation then $L_\rho \in$ NP.

Notation: If $W(\cdot)$ is an oracle machine, then $W^L(x)$ denotes the output of W with oracle $L \subseteq \{0,1\}^*$ and input $x \in \{0,1\}^*$.

We now say what it means for search to reduce to decision for an NP-relation. An equivalent formulation of the definition that follows appears in [BD].

Definition 2.2 *Suppose ρ is an NP-relation and $W(\cdot)$ is a polynomial time oracle machine. Let $L = L_\rho$. We say that W is a* ρ-witness finder *if for each $x \in L$ it is the case that $W^L(x) \in \rho(x)$. We say that* search reduces to decision for ρ *if there exists a ρ-witness finder.*

Note that the witness finder is not restricted to any particular method (for example, it is not required that the length of queries be decreasing with time). Rather, any polynomial time computation is allowed. This strengthens negative results.

We now wish to say what it means for search to reduce to decision for an NP *language* (as opposed to an NP-relation). Begin with the following terminology.

Definition 2.3 *Suppose ρ is an NP-relation and $L \subseteq \{0,1\}^*$ is a language. We say that ρ* defines *L if $L = L_\rho$.*

Clearly, $L \in$ NP iff there exists an NP-relation which defines L. However, for any particular language $L \in$ NP, there may be many different NP-relations which define L. If L is NP-complete, then search reduces to decision for *any* of these NP-relations. However if L is not NP-complete, then search might reduce to decision for some of them but not for others. In defining what it means for search to reduce to decision for L we have chosen to be liberal: we ask only that there be *some* NP-relation ρ defining L for which search reduces to decision.

Definition 2.4 *Suppose $L \subseteq \{0,1\}^*$. We say that* search reduces to decision for *L if there exists an NP-relation ρ such that ρ defines L and search reduces to decision for ρ.*

As we have indicated in Section 1, our definition is motivated by interactive proofs and the question of whether proving membership is harder than deciding it. Proving membership in L is easy (in the sense that L has a competitive NP-proof system) as long as search reduces to decision for *some* NP-relation defining L, so we are led to Definition 2.4. We note, in this context, that there are languages (such as $\{0,1\}^*$) which are easy but have an associated search problem which is hard [Va], and we certainly don't wish to think of search as being harder than decision for these languages. Appropriately, search does reduce to decision for these languages according to our definition.

Finally, we note that the existence of a language for which search does not reduce to decision does, of course, imply the existence of an NP-relation for which search does not reduce to decision, so negative results under our definition are stronger than those which simply conclude the existence of NP-relations for which search is harder than decision.

Whenever ρ is understood we will say "witness" or "witness finder" rather than "ρ-witness" or "ρ-witness finder."

2.2 Uniformly Log-Sparse Languages

Our proof will use languages which combine logarithmic sparseness with the property that it be possible to efficiently identify a logarithmic sized superset of the strings below any given length. Let us proceed to the formal definitions.

Definition 2.5 *The* census function *$\mu_L\colon \mathsf{N} \to \mathsf{N}$ of $L \subseteq \{0,1\}^*$ of a language L is defined by $\mu_L(n) = \sum_{i=0}^{n} |L \cap \{0,1\}^i|$. We say that L is* log-sparse *if $\mu_L(n) = O(\log n)$.*

That is, $\mu_L(n)$ is the number of strings in L which have length $\leq n$, and a language is log-sparse if it contains at most $O(\log n)$ strings of length at most n. Log-sparse languages were used in [HSI] where they were called "super-sparse."

The next definition formalizes the idea of being able to efficiently identify some super-set of a language, and then specifies the notion of "uniform log-sparseness" in which we are interested.

Definition 2.6 *We say that $C \subseteq \{0,1\}^*$ is a* candidate selector *for L if C is polynomial time decidable and $L \subseteq C$. We say that a language L is* uniformly log-sparse *if it has a log-sparse candidate selector.*

As we will see in Section 2.3, the interest of uniformly log-sparse languages is that they form a class for which the problem of reducing search to decision is particularly hard. Let us end this section by stating a lemma which we will use later. This lemma generalizes work of Hartmanis, Sewelson and Immerman [HSI], who showed that there is a log-sparse language in NP − P if EE $\neq$ NEE. † Log-sparseness is weaker than uniform log-sparseness in that no candidate selector is required, but it is easy to see that a uniformly log-sparse language in NP − P nonetheless exists under the same assumption. For completeness we provide a sketch of the (entirely standard) proof.

Lemma 2.7 *If* EE $\neq$ NEE *then there is a uniformly log-sparse language in* NP − P*.*

Proof: We use a standard "downward separation" argument. Assume EE $\neq$ NEE and suppose $L' \in$ NEE − EE. Define $L = \{y.0^{g(|y|)-|y|} : y \in L'\}$, where $g(k) = 2^{2^k}$. We claim that L is uniformly log-sparse and $L \in$ NP − P.

† [HSI] claimed the converse as well, but Allender [Al] points out that their proof is flawed and the theorem cannot be proved using techniques that relativize.

Define A to be the algorithm which on input $x \in \{0,1\}^n$ behaves as follows. If n is not in the range of g then A outputs 0. Else it computes $k = g^{-1}(n)$ and outputs 1 iff the last $n - k$ bits of x are zero. Then $C = \{ x : A(x) = 1 \}$ is a candidate selector for L and $\mu_C(n) \leq 2^{1+g^{-1}(n)} = O(\log n)$, so L is uniformly log-sparse.

The fact that $L \in \mathrm{NP} - \mathrm{P}$ follows directly from the fact that $L' \in \mathrm{NEE} - \mathrm{EE}$. ■

2.3 A Language for which Search Does not Reduce to Decision

The following, which is the main lemma of this section, shows that the reduction of search to decision for a uniformly log-sparse language is only possible in the trivial case where the language is already in P.

Lemma 2.8 *Suppose L is a uniformly log-sparse language for which search reduces to decision. Then $L \in \mathrm{P}$.*

Proof: By assumption, there exists an NP-relation ρ and a polynomial time oracle machine W such that ρ defines L and W is a ρ-witness finder. We will construct a polynomial time machine M which decides L. We begin by describing the idea informally.

The idea is to use W as a subroutine to find a witness. The difficulty is, of course, that W makes oracle queries about L itself. Not having access to an oracle for L, our machine M certainly cannot correctly answer these queries. To see how it can nonetheless exploit W, suppose for a moment that W is guaranteed to make only one oracle query in its entire computation on input x. Then M can try both possible answers. That is, it branches into a pair of parallel computations. In the first it answers the query by 0 and in the second it answers it by 1, and in both cases it then runs W until W halts. Clearly $x \in L$ if and only if at least one of these runs outputs a witness, and the strategy is polynomial time.

This idea extends to W making $O(\log n)$ queries. In reality, however, W could make polynomially many queries so that this strategy is not efficient. This is the point where we invoke the uniform log-sparseness of L (which we have not used so far). This implies that there are really only $O(\log n)$ "effective" queries that W can make: since W can only write down queries of polynomial length, we can use the log-sparse candidate selector of L to identify a set of at most $O(\log n)$ strings which include all strings in L which W could possibly query, and we need branch only on these.

With this overall strategy in mind let us now specify the operation of M more precisely. Since ρ is an NP-relation, there exists a constant $c \in \mathsf{N}$ such that for all $x \in \{0,1\}^*$ it is the case that $\rho(x) \subseteq \{0,1\}^{|x|^c}$. We can assume that there is a constant $d > 0$ such that on any input $x \in \{0,1\}^*$ the machine W will halt in $\leq d|x|^d$ steps and output a string of length $|x|^c$, regardless of how the oracle queries of W are answered. We also assume (wlog) that all queries made by W are distinct. Let C be a log-sparse candidate selector for L. Now, on input $x \in \{0,1\}^n$ the machine M behaves as follows:

(1) M runs W on input x. Each time W makes an oracle query q, the machine M provides a response as follows:

  (1.1) If $q \notin C$ then M responds with 0.

  (1.2) Else it continues by trying in parallel both possible answers 0 and 1. That is, M branches into two "parallel" computations. In the first it lets the response to q be 0 and in the second it lets the response to q be 1. It then continues to run W in each computation.

  In this manner M generates a number of parallel computations. After dn^d steps all of these computations have halted and each has yielded an n^c bit output (the output of W).

(2) M now examines the set of outputs from the previous step. It accepts if at least one of these outputs w satisfies $\rho(x, w) = 1$, and rejects otherwise.

This completes the description of the machine M. The fact that it works should be clear, but for completeness let us spell it out.

First, to see that M accepts x if and only if $x \in L$, it suffices to check that on at least one of the parallel computations all oracle queries are correctly (that is, according to L) answered. But Step (1.1) is obviously correct by definition of the candidate selector and in step (1.2) everything is being tried, so one of the runs will certainly end up having all the right query answers.

The next thing to check is that M runs in polynomial time. It suffices to show that the total number of parallel computations is $n^{O(1)}$. For this it suffices to show that the number of branches on any path is $O(\log n)$. We now argue the latter. First note that any query q has length at most dn^d (the running time of W). But branching only occurs when $q \in C$, and we have assumed that all W's queries are distinct. So the number of times branching occurs is at most the size of $\{ q \in C : |q| \leq dn^d \}$, which is at most $\mu_C(dn^d) = O(\log n)$. This completes the proof. ■

We can now put the pieces together to obtain the result:

Theorem 2.9 *If* EE $\neq$ NEE *then there exists a language in* NP *for which search does not reduce to decision.*

Proof: By Lemma 2.7 there exists a uniformly log-sparse language $L \in$ NP $-$ P. By Lemma 2.8, search cannot reduce to decision for L. ■

Note that the fact that search does not reduce to decision for L implies that L is not NP-complete. The existence of a non NP-complete language in NP $-$ P can however be established assuming only P $\neq$ NP (cf. [La]).

3 Deciders and their Properties

Before extending the ideas of the previous section to interactive proofs and checking, we pause to develop some technical materiel. This materiel will be useful in proving the results of later sections. In particular we introduce the notion of a "decider" which will enable us to give a unified and more concise treatment of the rest of the results of this paper. We begin with the definition.

Definition 3.1 *Let D be a probabilistic, polynomial time oracle machine. We say that D is a* decider *for language L if for each $x \in \{0,1\}^*$ the following is true:*

(1) *if $x \in L$ then $D^L(x)$ accepts with probability $\geq 2/3$*

(2) *if $x \notin L$ then the probability that $D^A(x)$ accepts is $\leq 1/3$ for all oracles A.*

We note that L has a decider if and only if it is in function-restricted IP (cf. [BK]). So deciders are just a way of characterizing languages in frIP. They can also be viewed as "checkers for YES instances." They are weaker than multi-prover interactive proofs: the results of [FRS] imply that if L has a decider then it has a multi-prover interactive proof. For us the motivation of Definition 3.1 is to "generalize" the notion of a witness finder in the light of our proof of Lemma 2.8. The property of the witness finder that was important in that proof was that it was correct for $x \in L$ as long as oracle queries were answered correctly (i.e. according to L), and it was "correct" for $x \notin L$ no matter how oracle queries were answered. Like a witness finder, correctness of the decider on inputs $x \in L$ is guaranteed (except here only with high probability) as long as oracle queries are correctly answered. On the other hand if $x \notin L$ then again correctness is guaranteed with high probability, *no matter how oracle queries are answered.* As we will see, these properties will suffice for us to appropriately extend Lemma 2.8 to Lemma 4.3.

The error probability of 1/3 in the above definition is not always sufficient. It is convenient to also define the following.

Definition 3.2 *Let D be a probabilistic, polynomial time oracle machine. We say that D is a* strong decider *for L if for each $x \in \{0,1\}^*$ the following is true:*
(1) *if $x \in L$ then $D^L(x)$ accepts with probability $\geq 1 - 2^{-|x|}$*
(2) *if $x \notin L$ then the probability that $D^A(x)$ accepts is $\leq 2^{-|x|}$ for all oracles A.*

Standard error-reduction, of course, says that strong deciders exist whenever deciders exist. For completeness let us state this as a proposition and provide a sketch of the proof.

Proposition 3.3 *If L has a decider then it has a strong decider.*

Proof: Let D be a decider for L. We define machine D' as follows. On input x, machine D' runs D on input x a total of $m = O(n)$ times, each time with independent coin tosses. The oracle queries made by D' are answered by D' by way of his own oracle (that is, if D makes oracle query q then D' makes oracle query q, and provides the answer he receives to D). D' outputs the majority vote of the outputs of D in the m trials. To see that D' is a strong decider for L, let $X_1^A, \ldots, X_m^A$ denote the sequence of random variables representing the outcomes of D with oracle A in these successive trials. These are independent, and for each $i = 1, \ldots, m$ satisfy $\Pr[X_i^L = 1] \geq 2/3$ in the case that $x \in L$, and $\Pr[X_i^A = 1] \leq 1/3$ for all oracles A in the case that $x \notin L$. An application of standard Chernoff bounds yields the desired conclusions. ∎

We saw in Section 2 that reducing search to decision for uniformly log-sparse languages is hard. Here we show that these same languages also do not have deciders unless they are in BPP.

Lemma 3.4 *Suppose L is uniformly log-sparse and has a decider. Then $L \in$ BPP.*

Proof: By Proposition 3.3, L has a strong decider D. We show how to use D to construct a BPP machine M to decide L. The idea is very much the same as that in the proof of Lemma 2.8, with the decider here playing the role that the witness finder played in that proof. That is, on input x the machine M will run D on input x and answer its oracle queries according to the same rules as those used in the proof of Lemma 2.8. M accepts if and only if the decider accepts on at least one of the parallel computations. The main difference (with respect to Lemma 2.8) lies in the fact that there is no way to tell whether a particular output of the decider is correct (in Lemma 2.8 one can always check whether or not the output of the witness finder is really a witness). Instead, the correctness of the procedure follows from the fact that the error probability of D is very small (2^{-n}). Details follow.

Let d be a constant such that D always halts in $\leq dn^d$ steps on inputs of length n. Let C be the log-sparse candidate selector for L. We assume (wlog) that all oracle queries made by D are distinct. On input $x \in \{0,1\}^n$ the machine M behaves as follows:

(1) M runs D on input x. Each time D makes an oracle query q, the machine M provides a response as follows:
  (1.1) If $q \notin C$ then D responds with 0.
  (1.2) Else it continues by trying in parallel both possible answers 0 and 1. That is, M branches into two "parallel" computations. In the first it lets the response to q be 0 and in the second it lets the response to q be 1. It then continues to run W in each computation.

In this manner M generates a number of parallel computations. After dn^d steps all of these computations have halted and each has yielded a output of 1 or 0 (the output of D).

(2) M now examines the set of outputs from the previous step. It accepts if at least one of these outputs is 1.

Since the answers in one of these parallel computations correspond to L, machine M accepts with probability $\geq 1 - 2^{-n}$ if $x \in L$. Now suppose $x \notin L$. Let $Q = \{ q \in C : |q| \leq dn^d \}$, and let $\mathcal{A}$ denote the set of all subsets of Q. Each parallel computation of M corresponds to running D with some oracle $A \in \mathcal{A}$. It follows that the probability that M accepts is at most $\sum_{A \in \mathcal{A}} \Pr[D^A \text{ accepts}]$. By assumption D is a strong decider for L, so we can bound this by $|\mathcal{A}| \cdot 2^{-n}$. But we claim that $|\mathcal{A}| \leq n^{O(1)}$, and so the probability that M accepts is $o(1)$, completing the proof. To justify the claim, note that $|\mathcal{A}| \leq 2^{|Q|}$ and $|Q| \leq \mu_C(dn^d) = O(\log n)$. ■

Recall that BPEE denote the class of languages accepted in time 2^{c2^n} for some constant $c \geq 0$ by a probabilistic machine with bounded error. By an argument analogous to that used in the proof of Lemma 2.7 we can show the following.

Lemma 3.5 *If* NEE $\not\subseteq$ BPEE *then there exists a uniformly log-sparse language in* NP $-$ BPP.

Combining this with Lemma 3.4 we obtain the following theorem which we will use in the next two sections.

Theorem 3.6 *If* NEE $\not\subseteq$ BPEE *then there exists a language in* NP *which does not have a decider.*

We remarked earlier that L has a decider iff $L \in$ frIP, so Lemma 3.6 says that NEE $\not\subseteq$ BPEE implies NP $\not\subseteq$ frIP. In later sections we will use Theorem 3.6 to show that (NEE $\not\subseteq$ BPEE implies) NP $\not\subseteq$ compIP, Check by showing that languages in compIP and Check have deciders (cf. Lemmas 4.3 and 5.2).

4 Competitive Interactive Proofs

We begin by recalling the notion of an interactive proof. We then define competitive interactive proofs and present our results.

4.1 Interactive Proofs

Interactive proofs are extensions of NP ones, so let us begin by recalling the latter. An NP proof system for a language $L \in$ NP is defined by a polynomial time *verifier* V. We imagine this verifier talking to a "prover." The parties receive a common input x, and the prover's goal is to convince the verifier to accept. To this end he is allowed to send the verifier a (single) message of length $n^{O(1)}$. The verifier's decision as to whether or not to accept is made as a function of the common input and this message (since the verifier is deterministic, this "decision" is a polynomial time binary predicate ρ evaluated on the common input and the prover's message). In the case that $x \in L$, we ask there exist some *deterministic* "prover" P who can convince the verifier to accept (this is the "completeness" condition). In the case that $x \notin L$, no "prover" should be able to convince the verifier to accept (this is the "soundness" condition). We usually specify an NP proof system as a pair (P, V) where P is a prover satisfying the completeness condition. Clearly, $L \in$ NP if and only if it possesses an NP proof system.

Interactive proofs, which were introduced by Goldwasser, Micali and Rackoff [GMR], are a natural extension of such NP proof systems. Both parties are now allowed to be probabilistic.

Moreover, they are allowed to interact (that is, they exchange messages for a polynomial number of rounds, and it is only at the end of this exchange that V decides whether or not to accept). We say that (P,V) is an interactive proof for a language L if (1) on common input a string in L, it is possible for P to induce V to accept with high probability, and (2) on common input a string not in L, there is no prover who can prevent V from rejecting with high probability.

Let us now proceed more formally. A party A in an interactive proof may be viewed as a (probabilistic) function of the common input and the conversation so far. The outcome of this function on input x (the common input) and c (transcript of conversation so far), which we denote by $A(x,c)$, is the next message computed by A (and sent to the other party).[†] We assume that the transcript of the conversation at any point may be uniquely parsed into its constituent messages. We may discuss the complexity of such parties in the usual way, viewing them as being computed by (probabilistic) Turing machines. For example, the verifier is a party computable by a probabilistic, polynomial time TM. Complexity is measured as a function of the length of the common input (which we usually denote by n). The total number of moves (a move consists of a party computing and sending a message) as well as the length of all messages are assumed to be bounded by a polynomial in n. At the end of the interaction, the verifier accepts or rejects by applying a (deterministic) binary predicate to the common input and transcript of conversation. Suppose a prover A interacts with a verifier B on common input x. The "probability that B accepts in its interaction with A on common input x" is the probability that B accepts given the common input x and transcript $\alpha_1\beta_1\ldots\alpha_{g-1}\beta_{g-1}\alpha_g$ chosen according to the following experiment: $\alpha_1 = A(x,\lambda), \beta_1 = B(x,\alpha_1), \alpha_2 = A(x,\alpha_1\beta_1), \beta_2 = B(x,\alpha_1\beta_1\alpha_2), \ldots, \alpha_g = A(x,\alpha_1\beta_1\ldots\alpha_{g-1}\beta_{g-1})$. (Here $g = g(n)$ is the total number of moves, λ is the empty string, and we are assuming for simplicity that A speaks first and last. The probabilities are over the random choices of both parties in this conversation).

Definition 4.1 (Interactive Proofs [GMR]) *Let (P,V) be a pair of (probabilistic) functions. We say that (P,V) is an interactive proof system for language L if V is probabilistic, polynomial time, and*

(1) *For every $x \in L$ the probability that V accepts in its interaction with P on common input x is $\geq 2/3$*

(2) *For every $x \notin L$ and every function $\widehat{P}$, the probability that V accepts in its interaction with $\widehat{P}$ on common input x is $\leq 1/3$.*

The first condition is the completeness condition and the second is the soundness condition.

We note the strength of the soundness condition: the quantification is over all functions $\widehat{P}$ (we call them "cheating" or "dishonest" provers), even non-computable ones.

We note that an NP proof system is a special kind of interactive proof system. Specifically, an NP proof system is an interactive proof system (P,V) in which both P and V are deterministic, the interaction is restricted to a single message from the prover to the verifier, and the probabilities in the completeness and soundness conditions are 1 and 0 (rather than 2/3 and 1/3), respectively. The addition of interaction and randomness, however, seems to add significantly to the language recognition power of the system. It was established by Lund, Fortnow, Karloff and Nisan [LFKN] that IP (the class of languages possessing interactive proofs of membership) contains the polynomial time hierarchy, and Shamir [Sh] extended this to show that IP equals PSPACE.

[†] When we say that this function is probabilistic we mean that to any x,c party A actually associates a distribution on strings, and $A(x,c)$ is a random element of this distribution.

4.2 Competitive Interactive Proofs

A basic complexity theoretic question is to determine how efficient the prover P can be in an interactive proof (P, V) of a language L. Certainly he would need at least the ability to decide the language himself. We define a competitive interactive proof system as one where the prover is allowed no more than this. Specifically, he must run in probabilistic polynomial time given access to L as an oracle. As we will see, competitive interactive proofs represent the natural generalization of the problem of decision vs. search.

Definition 4.2 *Let P be a probabilistic polynomial time oracle machine and V a probabilistic polynomial time machine. We say that (P, V) is a* competitive interactive proof system *for a language L if*

(1) *For every $x \in L$ the probability that V accepts in its interaction with P^L on common input x is $\geq 2/3$*

(2) *For every $x \notin L$ and every interactive TM $\widehat{P}$, the probability that V accepts in its interaction with $\widehat{P}$ on common input x is $\leq 1/3$.*

The first condition is the completeness condition and the second is the soundness condition. We call P a competitive prover.

We note that the soundness condition remains the same as in the definition of interactive proofs. In particular, we do not restrict the computational power of the "cheating" prover $\widehat{P}$ in the case $x \notin L$. Our goal is to understand the difficulty of providing a correct proof, and unrestricted soundness would appear to be an inherent property of "proofs."

Competitive NP proof systems are defined in the natural way. That is, a *competitive NP proof system* is a competitive interactive proof system in which both parties are deterministic, the interaction is restricted to a single message from the prover to the verifier, and the probabilities in the completeness and soundness conditions are 1 and 0 (rather than 2/3 and 1/3), respectively. Equivalently, it is an NP proof system in which the prover is restricted to polynomial time plus an oracle for L. We now note that search reduces to decision for L if and only if L has a competitive NP proof system (the prover in the competitive NP proof system corresponds to the witness finder). It is in this sense that competitive interactive proofs are the natural extension of the problem of decision versus search.

4.3 A NP Language not Possessing a Competitive Interactive Proof

In Section 2 we presented a language $L \in \text{NP}$ for which search is unlikely to reduce to decision. In other words, L is unlikely to have a competitive NP proof system. The truth, however, is that proving membership in NP languages remains hard even when interaction and randomness are allowed: we will show here that there is probably an NP language which does not even have a competitive interactive proof system. Given the results of Section 3 we need only the following lemma which shows that any language possessing a competitive interactive proof also has a decider (or, equivalently, that compIP $\subseteq$ frIP).

Lemma 4.3 *Suppose L has a competitive interactive proof system. Then it has a decider.*

Proof: Let (P, V) be a competitive interactive proof for L. We note that in probabilistic polynomial time we can run both P and V. So given an oracle for A, the machine D, on input x, can sample the space of conversations between P^A and V on input x, and accept if and only if the conversation obtained is accepting. Details follow.

Let $r(n)$ denote (a polynomial bound on) the number of coin tosses used by P and V on any input of length n. D picks, uniformly at random, a $r(n)$ bit string R_P and a $r(n)$ bit string R_V. He now runs P and V on common input x, using R_P as the coins for P and R_V as the coins for V. That is, assuming for example that P sends the first message, D would run P (with coins R_P) to get P's first message. He would then run V (with coins R_V) to get the response, and so on. Oracle queries made by P in this process are answered by D by way of its own oracle (that is, if P makes oracle query q then D makes oracle query q, and provides the answer he receives to P). Eventually D obtains the output of V (1 if V accepts and 0 otherwise), and outputs this value. Given any particular oracle A, it is the case that $D^A(x)$ is a 0/1 random variable, and clearly the probability that it is 1 equals the probability that V accepts in its interaction with P^A on common input x. By the assumption that (P, V) was a competitive interactive proof for L it follows that $\Pr[D^L(x) = 1] \geq 2/3$ in the case that $x \in L$, and $\Pr[D^A(x) = 1] \leq 1/3$ for all oracles A in the case that $x \notin L$. ■

Combining Lemma 4.3 and Theorem 3.6 yields the theorem.

Theorem 4.4 *If* NEE $\not\subseteq$ BPEE *then there exists a language in* NP *that does not have a competitive interactive proof.*

We note that we have done more than simply show that interactive proofs may be more powerful than competitive ones, because the language L of Theorem 4.4 is in NP, a subclass of IP which possesses particularly simple interactive proofs. To show only that interactive proofs are more powerful than competitive ones, it would suffice to present a language in IP (but not necessarily in NP) which does not have a competitive interactive proof. This can be done under weaker assumptions, by an extension of the same argument we used above. For example, let

$$\mathrm{EESPACE} = \bigcup_{c \geq 0} \mathrm{SPACE}(2^{c2^n}) .$$

Then we can show that if EESPACE $\not\subseteq$ BPEE then there exists a language in IP which does not possess a competitive interactive proof.

In general, to construct a language L which lies in some particular complexity class C but does not possess a competitive interactive proof, it suffices to assume that the "double-exponential counterpart" of C is not contained in BPEE. We put the phrase "double-exponential counterpart" in quotes because it, of course, does not always make sense (many classes have no such counterpart). But there are many natural classes (such as those used here) for which this paradigm does make sense.

4.4 Zero-Knowledge Aspects

The "competitive" aspects of zero-knowledge proofs may also be worth investigating. To initiate such an investigation, let us try to discuss briefly what one can easily infer from known work and what are the open questions.

Do NP languages have competitive zero-knowledge interactive proofs? In general, of course, they probably do not since by Theorem 4.4 they probably do not even have competitive interactive proofs (let alone ZK ones). An appropriate question then is whether NP languages which possess competitive interactive proofs also possess competitive zero-knowledge interactive proofs. The answer depends on the kind of zero-knowledge one considers and on the kind of cryptographic assumptions one is willing to make.

Let us first consider computational ZK. The result of Goldreich, Micali and Wigderson [GMW] implies that NP-complete languages have competitive ZK interactive proofs, given the existence of one-way functions (more generally, it implies that if search reduces to decision for L then L has a competitive ZK interactive proof, given the existence of one-way functions). We do not know whether the assumption that there exist one-way functions suffices to show that *any* language which possesses a competitive interactive proof also possesses a competitive ZK interactive proof. But we do know that the latter conclusion may be established with stronger assumptions such as the existence of "ideal" secure circuit evaluation or the existence of "oblivious transfer." This follows from the result of Kilian [Ki] (and we refer the reader to that paper for details on what exactly are these assumptions).

All statistical ZK languages known to possess competitive interactive proofs are also known to possess statistical ZK competitive interactive proofs (these languages are graph isomorphism [GMW], graph non-isomorphism [GMW], and quadratic non-residuosity [GMR]). We do not, of course, know whether or not quadratic-residuosity has a competitive statistical ZK interactive proof given that we do not know whether or not it has a competitive interactive proof at all.

5 Program Checking

Blum and Kannan [BK] introduced the notion of program checkers. Informally, a checker for a function f is a probabilistic, polynomial time oracle machine which receives as an oracle a program P which purports to compute f. The checker also receives an input x. If the program is entirely correct (that is, $P(y) = f(y)$ for all y) then the checker is supposed to accept with high probability. However if the program disagrees with f on the particular input x provided to the checker then the checker should reject with high probability. The definition follows. We note that by a "program" we mean a deterministic machine that halts on all inputs. We also recall that the characteristic function of a language L is the function $\chi_L\colon \{0,1\}^* \to \{0,1\}$ defined by $\chi_L(x) = 1$ if $x \in L$ and 0 otherwise.

Definition 5.1 [BK] *Let C be a probabilistic polynomial time oracle TM. C is a* checker *for $f\colon \{0,1\}^* \to \{0,1\}$ if for all programs P and all $x \in \{0,1\}^*$ it is the case that*

(1) *If $P(y) = f(y)$ for all $y \in \{0,1\}^*$ then $C^P(x)$ accepts with probability $\geq 2/3$*

(2) *If $P(x) \neq f(x)$ then the probability that $C^P(x)$ accepts is $\leq 1/3$.*

We say C is a checker for a language L if it is a checker for the characteristic function of L. L is checkable *if it has a checker.*

The definition is close in spirit to that of (competitive) interactive proofs, but there are two important differences. First, unlike interactive proofs, checking is a "symmetric" notion in which the checker for language L must be able to determine that P is correct on x not only when $x \in L$ but also when $x \notin L$. Second, programs are history independent objects, while (cheating) provers are not. Thus, if L and $\overline{L}$ both have competitive interactive proofs then L has a checker, while we do not know whether or not every checkable language has a competitive interactive proof.

Checkers are also related to multi-prover interactive proofs [BGKW]. In particular, results of [FRS] imply that the class of languages which possess checkers in contained in MIP $\cap$ coMIP (where MIP is the class of languages possessing multi-prover interactive proofs of membership). We note that MIP = NEXP, by the result of [BFL].

Blum and Kannan [BK] showed that Check = frIP $\cap$ cofrIP. It follows that checkable languages have deciders. For completeness, however, let us see this directly.

Lemma 5.2 *Suppose L has a checker. Then it has a decider.*

Proof: Let C be a checker for L. Let D be the probabilistic polynomial time oracle machine which, on input x, works as follows. D begins by querying its oracle with the string x. If the oracle returns 0 then D rejects. Else, it runs C on input x, using its own oracle (denoted A) to answer C's oracle queries, and accepts iff C accepts. We claim that D is a decider for L. To see this we need to check that for each $x \in \{0,1\}^*$ the two conditions of Definition 3.1 hold. The first condition is clear. To see that the second is true, suppose $x \notin L$, and suppose first that $A(x) = 0$. In this case, D rejects with probability 1. Now suppose $A(x) = 1$. Then the probability that D accepts is at most 1/3 because C is a checker. ∎

Combining Lemma 5.2 and Theorem 3.6 yields the theorem.

Theorem 5.3 *If* NEE $\not\subseteq$ BPEE *then there exists a language in* NP *that is not checkable.*

Similarly if EESPACE $\not\subseteq$ BPEE then there exists a language in PSPACE which is not checkable.

We recall that a language L is *coherent* if the membership of x in L can be decided in probabilistic polynomial time and bounded error by a machine (called the *examiner*) which has access to L as an oracle but is allowed to query this oracle only on points different from x. If L is not coherent we say it is incoherent. Previous negative results on checking were established by first exhibiting incoherent sets and then exploiting Yao's observation that any incoherent set is uncheckable (cf. [Ya, BF]). We note that our (stronger) results are obtained more directly. Moreover, our techniques indicate that even within NP the class of coherent sets could be much "larger" than the class of checkable ones. Let us sketch why this is so.

The "disjoint union" of languages A and B, denoted $A \oplus B$, is $\{0x : x \in A\} \cup \{1x : x \in B\}$; this construct is widely used in complexity theory (eg. [BD, HH]). It is easy to see (cf. [BF]) that $L \oplus L$ is coherent for any language L. It is also easy to see that the transformation $L \mapsto L \oplus L$ preserves many complexity characteristics of L; for example, membership in NP, compNP, compIP, Check, frIP. In particular, combining this observation with Theorem 5.3 yields the claimed separation:

Theorem 5.4 *If* NEE $\not\subseteq$ BPEE *then there exists a language in* NP *that is coherent but not checkable.*

6 Towards Competitive Proofs for Quadratic Residuosity

In this section we return to the unresolved question of whether the language of quadratic residuosity has a competitive interactive proof system, and present a special case of the problem where competitive proofs are possible.

6.1 Definitions

We will be looking at "promise" problems rather than problems of language membership. The difference is that in the former we begin with a "promise" that the input already belongs to some set, and we have only to "decide" whether or not it falls in a given subset of this set. Such problems have been considered in many works; eg. [ESY]. The formalization we use is different from (but equivalent to) the ones used in these works, and is as follows. The problem is specified by a pair of disjoint sets (A, B). Intuitively, the input is promised to be in $A \cup B$ and we have to decide whether it is in A or in B. Corresponding to promise problems are promise oracles which are guaranteed to be correct only when the "promise" is true.

Definition 6.1 *A* promise problem *is a pair of disjoint sets* (A, B)*. A* promise oracle *(for a promise problem* (A, B)*) is an oracle which given a query* q *returns* 1 *if* $q \in A$ *and* 0 *if* $q \in B$*.*

Note that while promise problems are, intuitively, "easier" than language recognition problems, promise oracles are correspondingly weaker than (normal) oracles. In particular, a promise oracle for (A, B) is weaker than an oracle for just A (or B) in that its response on queries outside $A \cup B$ is indeterminate.

A competitive interactive proof for a promise problem (A, B) is just an interactive proof that $x \in A$ given that $x \in A \cup B$ and having the property that the "competitive" prover gets only a promise oracle for (A, B). The more formal definition follows.

Definition 6.2 *Let P be a probabilistic polynomial time oracle machine and V a probabilistic polynomial time machine. We say that (P, V) is a competitive interactive proof for promise problem (A, B) if*

(1) *For every $x \in A$ and every promise oracle O for (A, B), the probability that V accepts in its interaction with P^O on common input x is $\geq 2/3$*

(2) *For every $x \in B$ and every interactive TM $\widehat{P}$, the probability that V accepts in its interaction with $\widehat{P}$ on common input x is $\leq 1/3$.*

6.2 Results

We recall that $x \in Z_N^*$ is a *quadratic residue* (or *square*) mod N if $x \equiv y^2 \pmod{N}$ for some $y \in Z_N^*$, and a *quadratic non-residue* (or *non-square*) mod N otherwise. Also, recall from Section 1.2 that

$$\begin{aligned} QR &= \{ (x, N) : x \text{ is a square mod } N \} \\ QNR &= \{ (x, N) : x \text{ is a non-square mod } N \} . \end{aligned}$$

The special case we are interested in is when N is the product of a constant number of distinct odd primes. To be more precise, first define

$$\begin{aligned} QR_s &= \{ (x, N) \in QR : N \text{ is a product of } s \text{ distinct odd primes} \} \\ QNR_s &= \{ (x, N) \in QNR : N \text{ is a product of } s \text{ distinct odd primes} \} . \end{aligned}$$

We will present a competitive interactive proof that $(x, N) \in QR_s$ given that it is already in $QR_s \cup QNR_s$. Note that QR_s and QNR_s are not complements of each other so that, formally, we are talking of a competitive interactive proof for the promise problem (QR_s, QNR_s) in the sense of Definition 6.2.

Theorem 6.3 *Let s be an integer ≥ 1. Then the promise problem (QR_s, QNR_s) possesses a competitive interactive proof.*

In related work, Kompella and Adleman [KA] present checkers for this same special case of quadratic residuosity when the modulus is the product of a constant number of primes (i.e. they present checkers for the promise problem (QR_s, QNR_s)). Their construction does not, however, extend to competitive interactive proofs, because the correctness of their checker uses the fact that a program (in contrast to a cheating prover) is history independent.

To prove Theorem 6.3, we begin by recalling some basic number theoretic facts. We refer the reader to [An, NZ] for number-theoretic background and justification of these facts.

For $x \in Z_N^*$ we let $Q_N(x) = 0$ if x is a quadratic residue mod N, and 1 otherwise. Suppose $N = p_1 \cdots p_s$ where $p_1, \ldots, p_s$ are distinct odd primes. Define the binary relation $\simeq$ on Z_N^* by

$$x \simeq y \quad \text{iff} \quad \forall i : 1 \leq i \leq s : Q_{p_i}(x \bmod p_i) = Q_{p_i}(y \bmod p_i) .$$

This is an equivalence relation. The equivalence class of x under this relation (namely $\{ y \in Z_N^* : x \simeq y \}$) is called its *residue class*. The product $xy \bmod N$ of two elements $x, y \in Z_N^*$ is a square mod N if x, y are from the same residue class and a non-square mod N otherwise. The total number of residue classes is 2^s, and they are all of the same size. We will denote them by $R_N^1, \ldots, R_N^{2^s}$, with the convention that the last, $R_N^{2^s}$, is the class of quadratic residues mod N.

We recall that there exists a competitive interactive proof for quadratic non-residuosity. We will exploit this fact by reducing the proof of $(x, N) \in QR_s$ to a polynomial number of proofs of non-residuosity in such a way that the prover need use only probabilistic, polynomial time and a promise oracle for (QR_s, QNR_s). The first step is the following definition.

Definition 6.4 *Let N be a product of s distinct odd primes and let $t = 2^s$. We call a vector $(y_1, \ldots, y_{t-1}) \in (Z_N^*)^{t-1}$* representative *of Z_N^* if the following conditions hold*
(1) *y_i is a non-square mod N for each $i = 1, \ldots, t-1$*
(2) *$y_i y_j \bmod N$ is a non-square mod N for each pair of indices i, j satisfying $1 \le i < j \le t-1$.*

This leads us to a way to reduce a residuosity test to to a collection of non-residuosity tests as long as we are in possession of a representative vector.

Proposition 6.5 *Let N be a product of s distinct odd primes and let $x \in Z_N^*$. Suppose $\vec{y} = (y_1, \ldots, y_{t-1})$ is representative of Z_N^* (where $t = 2^s$). Then $(x, N) \in QR_s$ if and only if $xy_i \bmod N$ is a non-square mod N for each $i = 1, \ldots, t-1$.*

Proof: $xy_i \bmod N$ will be a non-square mod N for each $i = 1, \ldots, t-1$ if and only if its residue class differs from the residue class of y_i for each $i = 1, \ldots, t-1$. But since $\vec{y}$ is representative, this happens if and only if x is a square mod N. ■

To use this, however, we have to be able to get representative vectors. It suffices to show how the prover can construct a representative vector and then convince the verifier that it is indeed representative, all using only probabilistic, polynomial time and a promise oracle for (QR_s, QNR_s).

Proposition 6.6 *There is a probabilistic, polynomial time oracle machine R which on input $(x, N) \in QR_s \cup QNR_s$ and access to the promise oracle for (QR_s, QNR_s) outputs either a representative vector for Z_N^* or the special symbol $\perp$, with the probability of the latter event being at most $1/4$.*

Proof: R picks at random $\vec{y}_1, \ldots, \vec{y}_m \in (Z_N^*)^{t-1}$, where $t = 2^s$ and m is a constant to be defined later. For each $i = 1, \ldots, m$ the machine R then uses the promise oracle to test whether or not the conditions of Definition 6.4 hold for $\vec{y}_i$. If some vector $\vec{y}_i$ passes the test then the first such vector is output If all vectors fail the test then R outputs $\perp$. The probability that a particular vector passes the test is $(t-1)!/t^{t-1}$, which is a positive constant. So it suffices to choose m to be a constant such that

$$\left[1 - \frac{(t-1)!}{t^{t-1}}\right]^m \le \frac{1}{4}.$$

(A (crude) calculation shows that $m = O((2e)^t)$ suffices). That R runs in probabilistic, polynomial time is clear. ■

We can now proceed to describe the protocols. We begin by recalling (following [GMR]) the basic (competitive) protocol to prove non-residuosity.

Protocol QNR

Input: (x, N) and 1^k

(V1) V picks at random $c_1, \ldots, c_k \in \{0, 1\}$ and $r_1, \ldots, r_k \in Z_N^*$, sets $z_i = x^{c_i} r_i^2 \bmod N$ (for $i = 1, \ldots, k$) and sends $z_1, \ldots, z_k$ to P.

(P1) P sets d_i to 0 if z_i is a quadratic residue mod N and 1 otherwise (for $i = 1, \ldots, k$), and sends $d_1, \ldots, d_k$ to V.

(V2) V accepts if and only if $c_i = d_i$ for all $i = 1, \ldots, k$.

Proposition 6.7 *Protocol QNR has the following properties:*

(1) *If $(x, N) \in QNR$ then the probability that V accepts in its interaction with P is* 1

(2) *If $(x, N) \notin QNR$ then for any $\widehat{P}$ the probability that V accepts in its interaction with $\widehat{P}$ is $\leq 2^{-k}$.*

(3) *P is competitive (that is, it runs in probabilistic, polynomial time given an oracle for QNR).*

Proof: The first two items follow from basic properties of modular residues, and we refer the reader to [GMR] for proofs. The last item is clear. ■

We now proceed to the competitive interactive proof for QR_s. We will use Protocol QNR as a subprotocol.

Protocol QR(s)

Input: $(x, N) \in QR_s \cup QNR_s$

Notation: We let $t = 2^s$.

(P1) P runs the algorithm of Proposition 6.6 and sends the output to V

(V1) If V receives $\perp$ from P then it rejects. If instead it receives a vector $\vec{y} = (y_1, \ldots, y_{t-1}) \in (Z_N^*)^{t-1}$ then the parties proceed to the next step.

`Sub-Protocol`: P uses Protocol QNR (with security parameter k set to 2) to prove to V that

(1) y_i is a non-square mod N for each $i = 1, \ldots, t-1$

(2) $y_i y_j$ mod N is a non-square mod N for each pair of indices i, j satisfying $1 \leq i < j \leq t-1$.

(3) xy_i mod N is a non-square mod N for each $i = 1, \ldots, t-1$

(a total of $(t-1)(t+2)/2$ invocations of the QNR protocol).

(V2) V accepts iff each of the above sub-proofs was accepting.

The correctness of the protocol follows from the results established above. Details follow.

Suppose $(x, N) \in QR_s$. Proposition 6.6 implies that the parties get a representative vector and proceed to the sub-protocol with probability $\geq 3/4$. Definition 6.4 and Proposition 6.5 imply that the inputs to the non-residuosity sub-proofs are all indeed non-squares mod N, and thus Proposition 6.7 implies that these sub-proofs all succeed with probability 1. So V accepts with probability $\geq 3/4$.

Suppose $(x, N) \in QNR_s$. If $\widehat{P}$ sends $\perp$ in its first step then V rejects, so suppose he sends a vector $\vec{y} = (y_1, \ldots, y_{t-1}) \in (Z_N^*)^{t-1}$. If $\vec{y}$ is not representative then by Definition 6.4 either there is an i such that y_i is a square mod N or there is a pair $i < j$ such that $y_i y_j$ mod N is a square mod N. In either case, the corresponding non-residuosity sub-proof fails with probability $\geq 3/4$, and V rejects. So suppose $\vec{y}$ is representative. But then Proposition 6.5 implies there is an i such that xy_i mod N is a square mod N. So the corresponding non-residuosity sub-proof fails with probability $\geq 3/4$ and V again rejects.

The competitiveness of P follows from Propositions 6.6 and 6.7.

The reason this does not extend to arbitrary N is, of course, that the number of residue classes could in general be exponential in the length of N, and in polynomial time we could not even write down a representative list. On the other hand, working through the proofs shows that the result does extend to the case where $s\colon \mathsf{N} \rightarrow \mathsf{N}$ is a polynomial time computable function of N which is bounded above by $\lg\lg\lg N = \lg\lg|N|$. For simplicity we have stuck to the case of constant s.

Clearly, the weakness of this result is in the "promise" that N is already a product of (exactly) s odd primes; this is what may be hard to prove competitively if one wants a competitive interactive proof of QR.

7 Open Questions

- *Quadratic-residuosity.* We think that the most interesting open question is whether or not the language of quadratic residuosity has a competitive interactive proof. Conditional results on the subject would also be interesting: for example, could one show that if quadratic residuosity has a competitive interactive proof then factoring is reducible (in probabilistic, polynomial time) to deciding quadratic residuosity? (Note that an affirmative answer to this last question would imply that if QR has a competitive interactive proof then it has an NP-proof via the simple "factorization witness.")
- *Reducing assumptions.* Another open question is whether one can reduce the assumptions required for our results. In particular can one show that there is a language for which search does not reduce to decision given $\mathrm{P} \neq \mathrm{NP}$, or even $\mathrm{E} \neq \mathrm{NE}$? Or could cryptographic assumptions such as the existence of one-way functions be used to establish the existence of languages in IP which don't have competitive interactive proofs?
- *Other settings.* What is the relationship of decision to search in the context of optimization problems and approximation algorithms, and does "search reduce to decision" in this setting? For example, consider the Traveling Salesman Problem (TSP). Let $\omega(G)$ denote the weight of an optimal tour on a (weighted) graph G, and suppose $\mu \geq 1$ is a constant. Suppose A is an oracle satisfying $\omega(G) \leq A(G) \leq \mu \cdot \omega(G)$ for all graphs G. Is there a polynomial time procedure which, with oracle access to A and input G, outputs a tour (in G) of weight at most $\mu \cdot \omega(G)$?
- *Perfect completeness.* An interactive proof (P, V) for L is said to have perfect completeness if the probability of acceptance in the completeness condition is 1. We know that any language L possessing an interactive proof also possesses one with perfect completeness [FGMSZ]. Does any language possessing a competitive interactive proof also possesses a competitive interactive proof with perfect completeness? (One of the motivations for this question is the fact that our competitive proof for the special case of quadratic residuosity in Section 6 does not possess perfect completeness).
- *Zero-knowledge.* We discussed the open questions in Section 4.4.

Acknowledgments

Many people pointed out to us the error in [HSI]: we thank Eric Allender, Juris Hartmanis, Jack Lutz, and Osamu Watanabe in this regard. We thank Muli Safra, Lance Fortnow and Jack Lutz for helpful discussions, and Silvio Micali for suggesting the term "competitive interactive proofs." We thank Oded Goldreich for many helpful comments on the paper. We thank Satish Thate for drawing our attention to the results of [La] mentioned in Section 2. Finally we thank a pair of (anonymous) referees for many valuable comments on the paper.

References

[An] D. ANGLUIN. Lecture Notes on the Complexity of Some Problems in Number Theory. *Technical Report* 243, Dept. of Computer Science, Yale University, August 1982.

[Al] E. ALLENDER. Limitations of the Upward Separation Technique. *Mathematical Systems Theory* **24**, 53–67, 1991.

[BFL] L. BABAI, L. FORTNOW AND C. LUND. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity* **1**, 3–40, 1991.

[BBFG] R. BEIGEL, M. BELLARE, J. FEIGENBAUM AND S. GOLDWASSER. Languages that are Easier than their Proofs. Proceedings *of the* 32nd *Symposium on Foundations of Computer Science*, IEEE, 1991.

[BF] R. BEIGEL AND J. FEIGENBAUM. Improved Bounds on Coherence and Checkability. YALEU/DCS/TR-819, September 11, 1990.

[BG] M. BELLARE AND S. GOLDWASSER. The Complexity of Decision versus Search. *MIT-LCS Technical Memo.* TM-444, April 1991.

[BP] M. BELLARE AND E. PETRANK. Making Zero-Knowledge Provers Efficient. Proceedings *of the* 24th *Annual Symposium on Theory of Computing*, ACM, 1992.

[BCGL] S. BEN-DAVID, B. CHOR, O. GOLDREICH AND M. LUBY. On the Theory of Average Case Complexity. *Journal of Computer and System Sciences* **44**(2), 193–219, 1992.

[BGKW] M. BEN-OR, S. GOLDWASSER, J. KILIAN AND A. WIGDERSON. Multiprover Interactive Proof Systems: How to Remove Intractability Assumptions. Proceedings *of the* 20th *Annual Symposium on Theory of Computing*, ACM, 1988.

[BK] M. BLUM AND S. KANNAN. Designing Programs that Check their Work. Proceedings *of the* 21st *Annual Symposium on Theory of Computing*, ACM, 1989.

[BD] A. BORODIN AND A. DEMERS. Some Comments on Functional Self-Reducibility and the NP Hierarchy. *Cornell University Technical Report* TR76-284, 1976.

[Co] S. COOK. The Complexity of Theorem Proving Procedures. Proceedings *of the* 3rd *Annual Symposium on Theory of Computing*, ACM, 1971.

[ESY] S. EVEN, A. SELMAN AND Y. YACOBI. The Complexity of Promise Problems with Applications to Public-Key Cryptography. *Information and Control* **2**, 159-173, 1984.

[FRS] L. FORTNOW, J. ROMPEL AND M. SIPSER. On the Power of Multiprover Interactive Protocols. Proceedings *of the* 3rd *Annual Conference on Structure in Complexity Theory*, IEEE, 1988.

[FGMSZ] M. FURER, O. GOLDREICH, Y. MANSOUR, M. SIPSER, AND S. ZACHOS. On Completeness and Soundness in Interactive Proof Systems. *Advances in Comput. Research* Vol. 5, S. Micali ed., JAI Press Inc.

[GMW] O. GOLDREICH, S. MICALI AND A. WIGDERSON. Proofs that Yield Nothing but their Validity. *JACM,* July 1991.

[GMR] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proofs. *SIAM J. Computing* **18**(1), 186–208, 1989.

[HH] J. Hartmanis, and L. Hemachandra. Complexity Classes without Machines. *Theoretical Computer Science* **58**, 129-142, 1988.

[HSI] J. Hartmanis, V. Sewelson and N. Immerman. Sparse Sets in NP-P: EXPTIME versus NEXPTIME. *Information and Control* **65**, 158-181, 1985.

[IN] R. Impagliazzo and M. Naor. Decision Trees and Downward Closures. Proceedings *of the* 3rd *Annual Conference on Structure in Complexity Theory*, IEEE, 1988.

[IS] R. Impagliazzo and M. Sudan. Private communication, May 1991.

[IT] R. Impagliazzo and G. Tardos. Decision Versus Search Problems in Super-Polynomial Time. Proceedings *of the* 30th *Symposium on Foundations of Computer Science*, IEEE, 1989.

[KUW] R. Karp, E. Upfal and A. Wigderson. The Complexity of Parallel Search. *J. Computer and System Sciences* **36**, 225–253, 1988.

[Ki] J. Kilian. Achieving Zero-Knowledge Robustly. *Advances in Cryptology – Crypto* 90 *Proceedings*, Lecture Notes in Computer Science Vol. 537, A. J. Menezes and S. Vanstone ed., Springer-Verlag, 1990.

[KA] K. Kompella and L. Adleman. Fast Checkers for Cryptography. *Advances in Cryptology – Crypto* 90 *Proceedings*, Lecture Notes in Computer Science Vol. 537, A. J. Menezes and S. Vanstone ed., Springer-Verlag, 1990.

[Kr] H. Krawczyk. Personal communication.

[La] R. Ladner. On the Structure of Polynomial Time Reducibility. *J. Assoc. Comput. Mach.* **22**, 155-171, 1975.

[LFKN] C. Lund, L. Fortnow, H. Karloff and N. Nisan. Algebraic Methods for Interactive Proof Systems. Proceedings *of the* 31st *Symposium on Foundations of Computer Science*, IEEE, 1990.

[NZ] I. Niven and H. Zuckerman. An Introduction to the theory of Numbers. John Wiley and Sons, NY 1960.

[Se] A. Selman. Natural Self-Reducible Sets. *SIAM J. Comput.* **17**(5), 989–996, October 1988.

[Sh] A. Shamir. IP=PSPACE. Proceedings *of the* 31st *Symposium on Foundations of Computer Science*, IEEE, 1990.

[Sp] D. Spielman. Private communication via Beigel and Feigenbaum, June 1991.

[Va] L. Valiant. On the Relative Complexity of Checking and Evaluating. *University of Leeds Technical Report* LS29JT, October 1974.

[Ya] A. Yao. Coherent Functions and Program Checkers. Proceedings *of the* 22nd *Annual Symposium on Theory of Computing*, ACM, 1990.