Quantifying Knowledge Complexity
(Preliminary Version)

by

O. Goldreich and E. Petrank

Technical Report #683
July 1991
Quantifying Knowledge Complexity∗

(Preliminary Version)

Oded Goldreich †
Erez Petrank ‡
Department of Computer Science
Technion
Haifa, Israel.

ABSTRACT

One of the many contributions of the seminal paper of Goldwasser, Micali and Rackoff [GMR] is the introduction of the notion of knowledge complexity. Knowledge complexity zero (also known as zero-knowledge) seems to have received most of the attention of the authors and all the attention of their followers. Unfortunately, the formulation of knowledge complexity (greater than zero) as appearing in that pioneering paper seems to be inadequate.

In this paper, we present several alternative definitions of knowledge complexity and investigate the relations between them.

1. Introduction

One of the many contributions of the seminal paper of Goldwasser, Micali and Rackoff [GMR] is the introduction of the notion of knowledge complexity. Knowledge complexity is intended to measure the computational advantage gained by interaction. Hence, something that can be obtained without interaction is not considered knowledge. The latter formalization is somewhat qualitative and therefore quite sound. Quantifying the amount of knowledge gained by interaction, in case it is not zero, is more problematic.

Goldwasser, Micali and Rackoff have suggested to characterize languages according to the knowledge complexity of their interactive proof systems [GMR]. The lowest class consists of languages having knowledge complexity zero. This class, also known as zero-knowledge, has received much attention in recent years. It should be stressed that the class zero-knowledge does not depend on the formulation of “the amount of knowledge gained” as this class consists of the case in which no knowledge is gained.

∗ This research was supported by grant No. 89-00312 from the United States - Israel Binational Science Foundation (BSF), Jerusalem, Israel.
† Bitnet: oded@cs.technion.ac.il
‡ Bitnet: erez@technix.technion.ac.il
An attempt to formalize the "amount of knowledge" (in case it is not zero) has appeared in [GMR] but was omitted from the later version of this work [GMR2] since the authors themselves found it inadequate (Micali, private communication). In particular, by the [GMR] formulation, a prover that with probability $\frac{1}{2}$ reveals a Hamiltonian path in the input graph and otherwise reveals nothing is considered to leak (only) one bit of knowledge. Furthermore, by that formalization, all languages in the class IP (i.e., all languages having interactive proof systems) have knowledge complexity $\log \left( \frac{1}{x} \right)$ bits. In fact, all interactive proofs leak only $\log \frac{1}{x}$ bits of knowledge (see Appendix). To the best of our knowledge, the notion of knowledge complexity (apart from zero-knowledge) has not been investigated any further, and that may be the reason that its inadequacy has not been pointed out.

We believe that the concept of knowledge complexity is an important one and may play a role in further investigations in Complexity Theory. We believe that our first task is to explore several alternative ways of defining knowledge complexity. In this paper we present several such alternatives and investigate the relations between them.

1.1. Background on zero-knowledge

Loosely speaking, an interactive proof system for a language $L$ is a two-party protocol, by which a powerful prover can "convince" a probabilistic polynomial time verifier of membership in $L$, but will fail (with high probability) when trying to fool the verifier into "accepting" non-members. An interactive proof is called zero-knowledge if the interaction of any probabilistic polynomial time machine with the predetermined prover on common input $x \in L$, can be "simulated" by a probabilistic polynomial time machine (called the simulator). The definition considers two types of probability distributions:

1. The distribution of the interaction of a probabilistic polynomial time machine with the prover on common input $x \in L$.

2. The output distribution of a probabilistic polynomial time machine (the simulator) on the same input $x$.

It is required that for every distribution of type (1), there is a distribution of type (2) such that these two distributions are similar. Similarity is interpreted in three possible ways yielding three different definitions of zero-knowledge.

(a) The most conservative interpretation is that the distributions are equal. The resulting definition is called perfect zero-knowledge. An example of a language having a perfect zero-knowledge interactive proof is Graph-Isomorphism [GMW].

(b) Slightly more liberal is the requirement that the distributions are statistically close, namely that their variation difference (Norm-1 difference) is negligible (i.e., smaller than any polynomial fraction). The resulting definition is called statistical (or almost perfect) zero-knowledge. The results of Fortnow [F] and Aiello and Hastad [AH] on the "complexity of zero-knowledge" refer to this definition.

(c) Most liberal is the requirement that the distributions are indistinguishable by all probabilistic polynomial time tests. The resulting definition is called computational zero-knowledge. The result of [GMW] stating that "all languages in NP have zero-knowledge proofs provided that one-way functions exist" refers to this definition.
1.2. Conceptual contributions of our work

Unless otherwise indicated, the following discussion refers to the definitions of knowledge complexity in which the simulated conversations are close to the real one in the statistical sense. Namely, we consider the hierarchy of knowledge complexity extending statistical zero-knowledge (as the zero level). Our first contribution is in defining several such hierarchies.

Among the definitions of knowledge complexity presented in this paper, we prefer the following. A prover $P$ is said to yield at most $k(lx I)$ bits of knowledge if whatever can be efficiently computed through an interaction with $P$ on input $x$ (in the language), can also be efficiently computed on input $x$ through an interaction with a "machine" which sends at most $k(lx I)$ bits. Hence, the computational advantage gained by the interaction with the prover who may send much more than $k(lx I)$ bits can be simulated by an interaction with a machine which only sends $k(lx I)$ bits. In some sense, this approach reduces knowledge complexity to communication complexity. Without loss of generality, the "knowledge-giving-machine" can be made memoryless and deterministic, by supplying it with all previous messages and with coin tosses. Hence, the "knowledge-giving-machine" is merely an oracle. We present three variants of the above definition. These variants are analogous of the common variants of probabilistic Communication Complexity. The most conservative (or "strict") approach is to consider the worst-case number of bits communicated on inputs $x$ of length $n$. The most liberal approach is to consider the average case (the average is taken only on the coin-tosses of the simulator, the input $x$ is fixed). In between and (in light of our results) much closer to the worst-case variant is a relaxation of the worst-case variant in which it is required that an output (i.e., a simulation of the interaction with the prover) is produced only with probability one half. This last variant is hereafter referred to as the oracle definition of knowledge complexity.

A different approach to knowledge complexity is to provide the simulator (of the interaction between the prover and the verifier) no help but rather relax the requirement concerning its output distribution. Instead of requiring, as in the zero-knowledge case, that the output distribution is similar to the distribution of the "real interaction", we only require that it contains a subspace which is similar to the distribution of the "real interaction". The knowledge complexity is defined as (minus) the logarithm of the density of this good subspace in the entire probability space (i.e., the output distribution). Interestingly, we show that knowledge complexity as defined equals (up to a constant) the definition of knowledge complexity in the oracle approach. It should also be stated that the definition of knowledge complexity as a "logarithm of the good fraction" agrees with the informal discussion in the [GMR] paper (although the formal definition presented there seems inadequate - for more details see Appendix). In fact, Micali (private communication) has independently discovered this definition.

The last interpretation of knowledge complexity, presented in this paper, is as the length of the shortest hint which allows an efficient machine to simulate the real interaction. Unlike in the oracle approach to knowledge complexity this hint is a predetermined function of the input and in particular, does not depend on the coin-tosses of the simulator (or even the verifier). Hence, the amount of knowledge in the hint sense leaked by two executions of the same protocol on the same input, equals the amount of knowledge leaked by a single execution. Another indication to the limitations of this approach is that only languages in $AM(2) \subseteq PH$ (which is believed to be a proper subset of $IP = PSPACE$) have polynomial knowledge complexity in the hint sense. We would expect that the knowledge complexity of a protocol is bounded by its Communication Complexity and hence, that all languages in $IP$ have polynomial knowledge complexity (as is indeed the case with
In order to summarize our results concerning the relations between the various definitions, we present the following notations. Let $\kappa_{Oracle}(\Pi)$, $\kappa_{Oracle}(\Pi)$, and $\kappa_{Oracle}(\Pi)$ denote the knowledge complexity of an interactive proof $\Pi=(P \leftarrow \rightarrow \mathbb{V})$ according to the strict, average and "intermediate" (i.e., output with probability $\frac{1}{2}$) variants of the oracle approach. Likewise, $\kappa_{fraction}(\Pi)$ and $\kappa_{hint}(\Pi)$ denote the knowledge complexity of $\Pi$ according to the "fraction" and "hint" approaches. In the following lines we summarize our results concerning the relations between the various definitions:

(1) For every interactive proof $\Pi$, $\kappa_{frac}(\Pi)-2 \leq \kappa_{Oracle}(\Pi) \leq \kappa_{Oracle}(\Pi) \leq \kappa_{frac}(\Pi)$.

(2) For every interactive proof $\Pi$, $|\kappa_{frac}(\Pi) - \kappa_{frac}(\Pi)| = O(1)$.\(^{(3)}\)

(3) For any interactive proof $\Pi$, $\kappa_{frac}(\Pi) \leq \kappa_{frac}(\Pi) + \log \log(\log(1x)) + O(1)$.

(4) For every polynomial $p$, there exists an interactive proof $\Pi$ such that $\kappa_{frac}(\Pi) = p(1x)$, and $\kappa_{frac}(\Pi) \leq p(1x)$.

(5) There exists an interactive proof $\Pi$ such that $\kappa_{frac}(\Pi) = 1$, and $\kappa_{frac}(\Pi) \geq p(1x)$.\(^{(4)}\)

(6) For every polynomial $p$ and every polynomial-time computable function $k(1x)$, there exists an interactive proof $\Pi$ such that $\kappa(\Pi) \leq p(1x)$, and $\kappa(\Pi) \geq p(1x)$.

Another contribution of our paper concerns the issue of parallel versus sequential execution of zero-knowledge proofs. In particular, many known zero-knowledge proofs (e.g., the one of Graph Isomorphism or the one for 3-Colorability which assumes the existence of one-way functions) consists of many sequential executions of a basic protocol. In contrast to some common belief, Goldreich and Krawczyk \cite{GK} showed that the parallel executions of these protocols cannot be shown zero-knowledge using a black-box argument (which is the only argument used so far in demonstrating that a protocol is zero-knowledge). We extend their results by showing that these parallel executions cannot be shown to have knowledge complexity smaller than the logarithm of the computational complexity of the language. In fact, we show that constant round AM proof systems having error probability $k(1x)$ for a language $L$ cannot be shown (using a black-box argument) to have knowledge complexity $k(1x) \leq \frac{1}{20}(\log(\frac{1}{1-e(1x)}))$, unless $L$ can be recognized in probabilistic time $\text{ix} O(1)\cdot 2^{(1x)}$.\(^{(5)}\) (This generalizes \cite{GK}'s result in which $k(1x)=0$.)

\(^{(2)}\) This measure of knowledge was originally suggested in \cite{BCK}, and seems adequate in the information theoretic model discussed there.

\(^{(3)}\) This is true only under the (reasonable) assumption that the knowledge complexity bound in the fraction sense is a polynomial time computable function.

\(^{(4)}\) Loosely speaking, we say that an interactive proof for a language $L$ can be shown zero-knowledge (or of knowledge complexity $k$) using a black-box argument, if there exists a universal simulator which can simulate the interaction, of the prover with any verifier, when only given black-box access to that verifier. It should be stressed that all known zero-knowledge were show zero-knowledge using such a "black-box simulator" and that it is hard to conceive an alternative way of proving that an interactive proof is zero-knowledge.
1.3. Technical Contribution

Among the results regarding the relations between our various definitions of knowledge complexity, the most interesting one is the proof that knowledge complexity in the fraction sense is greater, up to a constant, than the knowledge complexity in the oracle sense (Proposition 3.2). Our proof makes use of a new lemma concerning hash functions. We believe that this simple lemma (Lemma 3.6) might be useful in many other settings.

As stated in subsection 1.2, we prove that languages having polynomial knowledge complexity in the hint sense are in AM[2]. This proof is an extension of the result by Aiello and Hastad [AH] that languages having zero-knowledge interactive proofs are in AM[2]. We also prove that languages having logarithmic knowledge complexity in the hint sense are in co-AM[2], thus extending Fortnow’s proof which regards zero-knowledge [F].

Our result regarding the knowledge complexity of constant round Arthur Merlin proof systems extends the work of Goldreich and Krawczyk [GK] This result refers only to simulators which use the verifier as a black box.

2. Definitions of knowledge complexity and some basic results

In this section we present the various definition of knowledge complexity. In each of the various definitions, knowledge complexity zero coincides with zero-knowledge. For sake of simplicity we present only the definitions of knowledge complexity that extend statistical zero-knowledge. Analogue definitions can be derived to extend perfect zero knowledge and computational zero-knowledge. All the analogous results hold except for the “perfect knowledge” analogue of Proposition 3.8. We believe that the hierarchy of statistical knowledge complexity is the most important one. Assuming the existence of secure bit commitment scheme (i.e., the existence of one-way functions), all languages in IP have interactive proofs of knowledge complexity zero [BGGHKMR,Y]. Also, for simplicity, we consider in all our definitions only machines (verifiers and simulators) which run in strict polynomial time. Analogue definitions and results for machines which run for expected polynomial time, can be derived (in some cases the proofs are slightly more involved).

Notation: Let $(P \leftrightarrow V)(x)$ be a random variable representing the interaction of $P$ and $V$ on input $x$. The probability space is that of all possible outcome of the internal coin-tosses of $V$ and $P$.

In the following, we shall use functions over the integers, to describe the knowledge complexity of a protocol. Some of our result require that this function is polynomial time computable in the following (liberal) sense:

Definition 2.0: We say that the knowledge complexity bound, $k: N \rightarrow N$, is polynomial time computable if there exists a probabilistic polynomial time algorithm that on input $n$ outputs $k(n)$ with probability $1 - 2^{-k(n)}$ (equivalently $1 - 2^{-2k(n)}$).

This definition is more liberal than the standard one, in which, the input $(n)$ is given to the machine in binary representation, thus allowing the machine to run only for $poly(log(n))$ number of steps.

In the first definition, knowledge is interpreted as a function of the input $x$. 

Definition 2.1: (knowledge complexity - Hint version) Let $k: \mathbb{N} \rightarrow \mathbb{N}$ be a function over the integers. We say that an interactive proof system $\Pi = (P \leftrightarrow V)$ for a language $L$ can be simulated using a hint of length $k$, if for every probabilistic polynomial time verifier $V'$ there exists a probabilistic polynomial (in $|x|$) time machine $M_V$ (simulator) such that for every $x \in L$ there exists a string $h(x)$ of length $k(|x|)$ such that $M_V(x, h(x))$ is statistically close to $(P \leftrightarrow V')(x)$. The knowledge complexity in the hint sense is an operator, denoted $k_{CHi}^\Pi$, which assigns each interactive proof system $\Pi$ a function $k_\Pi$ so that for any $x \in L$, $\Pi$ can be simulated using a hint of length $k_\Pi(|x|)$. The class $K_{CHi}^\Pi(k(|x|))$ is the set of languages having interactive proofs with knowledge complexity (in the hint sense) bounded above by $k(|x|)$.

A subtle point regarding the above definition, concerns the convention of feeding the hint to the simulator. We adopt the convention by which, the hint (a binary string) is placed on the left of a special "hint tape", padded to its right by infinitely many zeros. The alternative convention, by which the padding is by a special symbol ("blank" Φ {0,1}) provides implicit knowledge and is not compatible with the other definitions (presented below). Nevertheless, in case the knowledge complexity bound (i.e., the function $k(|x|)$) is polynomial-time computable, the difference between the two conventions yields at most a difference of one bit in the knowledge complexity. In any case, for knowledge complexity zero there is no difference.

The second definition widens the interpretation of knowledge to a stochastic one. The formalization is by use of oracle machines. There are three variants of oracle knowledge complexity.

Definition 2.2: (knowledge complexity - strict Oracle version) We say that an interactive proof $\Pi$ for a language $L$ can be strictly simulated using $k$ oracle queries, if for every probabilistic polynomial time verifier $V'$ there exists a probabilistic polynomial time oracle machine $M_V$ and an oracle $A$ such that:
1. On input $x \in L$, machine $M_V$ queries the oracle $A$ at most $k(|x|)$ times.
2. For each $x \in L$, $M_V(x)$ is statistically close to $(P \leftrightarrow V')(x)$.

The knowledge complexity in the strict oracle sense, denoted $k_{CO}^\Pi$ (also $k_{CO}^\Pi$), and the class $K_{CO}^\Pi$ (also $K_{CO}^\Pi$) are defined analogously to definition 2.1.

Using the bits of the oracle as a hint we get:

Proposition 2.3: For every interactive proof $\Pi$, $k_{CO}^\Pi(\Pi) \leq k_{CHi}^\Pi(\Pi)$.

The second variant of oracle knowledge complexity allows the simulator to announce failure in half of its runs.

Definition 2.4: (knowledge complexity - Oracle version) This definition is the same as the previous one except that condition (2) is substituted by:

2'. For each $x \in L$, machine $M_V$ produces an output with probability $\geq \frac{1}{2}$, and given that $M_V$ has produced an output, $M_V(x)$ is statistically close to $(P \leftrightarrow V')(x)$.

The knowledge complexity in the oracle sense, denoted $k_{CO}^\Pi$ (also $k_{CO}^\Pi$) and the class $K_{CO}^\Pi$ (also $K_{CO}^\Pi$) are defined similarly to the previous definitions.

Clearly,

Proposition 2.5: For every interactive proof $\Pi$, $k_{CO}^\Pi(\Pi) \leq k_{CO}^\Pi(\Pi)$.

The constant $\frac{1}{2}$ used as a lower bound on the probability that the simulator produces an output, can be substituted by any constant $0 \leq \delta \leq 1$ to get $k_{CO}^\Pi(\Pi)$. The relation of this definition to the original one is given in the following two proposition:
Proposition 2.6: For every interactive proof $\Pi$, and every $0 < \varepsilon < \frac{1}{2}$, $\kappa_{\text{Oracle}}(\Pi) \leq k_{\text{Oracle}}(\Pi) + [\log(\frac{1}{\varepsilon})]$.  

Proof: We build a new simulator that picks $2^\log(\frac{1}{\varepsilon})$ random tapes for the original simulator and sends them to the oracle. The oracle specifies the first random tape on which the original simulator produces a conversation, or zero if no such tape exists. In the first case, the new simulator runs the original simulator on the selected tape, and refers its queries to the oracle. The probability that no output is produced is $(1 - \varepsilon)^{\log(\frac{1}{\varepsilon})} < \frac{1}{2}$.  

Proposition 2.7: For every interactive proof $\Pi$, and every $0 < \varepsilon < \frac{1}{2}$, $\kappa_{\text{Oracle}}(\Pi) \leq k_{\text{Oracle}}(\Pi) + \{\log \log \left(\frac{1}{\varepsilon}\right)\}$.  

Proof: Similar. This time the list is of length $\log \left(\frac{1}{\varepsilon}\right)$, the failure probability is $(\frac{1}{2})^{\log \log \left(\frac{1}{\varepsilon}\right)} \leq \varepsilon$, and a pointer to an element in this list requires $\log \log \left(\frac{1}{\varepsilon}\right)$ bits.  

The third variant of the oracle knowledge complexity, allows no failure, but instead, allows the simulator more flexibility in the number of queries, and measures the expected number of queries.

Definition 2.8: (knowledge complexity - Average Oracle version) This definition is the same as definition 2.2, except that condition (1) is replaced by:

1'. On input $x$, the average number of queries that machine $M_\nu$ makes to oracle $A$ is at most $k(|x|)$.  

The average knowledge complexity in the oracle sense, denoted $\kappa_{\text{Oracle}}$, and the class $\kappa C_{\text{Oracle}}$ are defined analogously to definition 2.1.

The last definition gives the simulator no explicit help. Instead, only a $\frac{1}{2\varepsilon}$ fraction of its output distribution is being considered.

Definition 2.9: (knowledge complexity - Fraction version) Let $\rho$ be a function from the integers to the reals in $[0,1]$. We say that an interactive proof system $\Pi$ for a language $L$ can be simulated at density $\rho(|x|)$ if for every probabilistic polynomial time verifier $V_\nu$ there exists a probabilistic polynomial time machine $M_\nu$ such that for each $x \in L$ there exists a subset $\text{SUCC}_{x}$ of the set of all possible coin tosses of the simulator with the following properties:

1) $\text{Prob}(\text{SUCC}_x) \geq \rho(|x|)$.

2) For every $x \in L$, when the coin tosses for machine $M_\nu$ are picked uniformly from $\text{SUCC}_x$, $M_\nu(x)$ is statistically close to $(P \leftarrow \nu(x))$.

The knowledge complexity in the fraction sense, denoted $\kappa_{\text{Fraction}}$, assigns the interactive proof $\Pi$ a function $k_\varepsilon$ so that $\Pi$ can be simulated at density $2^{-k_\varepsilon}$. The class $\kappa C_{\text{Fraction}}(k(|x|))$ is defined analogously to the previous definitions.

3. Inclusion results

For every interactive proof $\Pi$, $\kappa_{\text{Oracle}}(\Pi) \leq \kappa_{\text{Oracle}}(\Pi) \leq \kappa_{\text{Stab}}(\Pi)$ (see Propositions 2.3 and 2.5). In the following sections we prove further relations between the definitions. We believe that the most interesting result (and proof) in this section is that of Proposition 3.2.
3.1. The oracle and the fraction versions are equal up to a constant

In this subsection, we prove equivalence up to a constant of the oracle and the fraction versions.

Proposition 3.1: For any interactive proof $\Pi$, $k_{\text{frac}}(\Pi) \leq k_{\text{oracle}}(\Pi) + 1$

Proof sketch: Let $k = k_{\text{frac}}(\Pi)$. We build a simulator that guesses a random string for the original simulator and then, guesses the oracle-answers. With probability at least $\frac{i}{2^k}$, the original simulator gets the correct oracle answers and yields an output. In this case, the new simulator produces exactly the same output distribution as the original simulator.

Proposition 3.2: For any interactive proof $\Pi$ with polynomial time computable knowledge complexity in the fraction sense, $k_{\text{frac}}(\Pi) \leq k_{\text{frac}}(\Pi) + 1$. Furthermore, if $k_{\text{frac}}(\Pi) = O(\log |x|)$, then $k_{\text{frac}}(\Pi) \leq k_{\text{frac}}(\Pi) + 1$.

Proof: Let $k = k_{\text{frac}}(\Pi)$. Our purpose is to pick a random $r \in \text{SUCC}_x$ and run the original simulator on $r$. The problem is how to sample $\text{SUCC}_x$. The naive solution of asking the oracle to pick an $r \in \text{SUCC}_x$ uniformly, is not good, since $r$ might be much longer than $k (|x|)$. A better suggestion, that works for $k = O(\log |x|)$ is the following. Pick a list of $2^k$ random strings for the original machine, and ask the oracle to specify one of them that belongs to $\text{SUCC}_x$. The oracle will use one bit to indicate whether there is any good string in the list and another $k (|x|)$ bits to point to the first good place in the list if such exists. The length of the list is polynomial in $|x|$ because $k = O(\log |x|)$.

Dealing with $k = \log |x|/o(1)$ is somewhat more complicated. We would like to do the same as before, but the problem is that the list of length $2^k$ is not of polynomial (in $|x|$) length, and thus cannot be given explicitly. Instead, we treat the problem of presenting the list and specifying an element in it using Universal Hash Functions (presented by Carter and Wegman [CW], and used in complexity theory in many works following Sipser [S]). We use the family of random affine transformation from $\{0,1\}^m \rightarrow \{0,1\}^l$, where $m$ is the length of the random string for the original simulator, and $l = m - k - 9$. Let $H_{m,l} \triangleq (A,b)$: $A$ is a binary $l \times m$ matrix, and $b \in \{0,1\}^l$. Applying an $h_{a,b} \in H_{m,l}$ on $r \in \{0,1\}^m$ is done by $h_{a,b}(r) = A'r + b$ (where all operations are done over GF(2)). We use two properties of this family. The universal property is used to give a short description of "somewhat random" sets, while its linearity enables us to interpret the oracle replies by solving a system of linear equations.

A list is implicitly picked by a random selection of $h_{a,b} \in H$ and $a \in \{0,1\}^l$. The list that is implicitly chosen is $h^{-1}(a)$, i.e., the affine space of vectors $r$ that satisfies the equation $A'r + b = a$. The oracle has to specify a point in this affine space, and this is done using additional linear equations. If the matrix $A$ is of rank $l$ (which is almost always the case), then the affine space is of dimension $m - l (=k+9)$. This affine space can be identified with $h$ to $l$ linear independent equations that are linearly independent of $A$. Therefore, completing $A$ to a non-singular $m \times m$ matrix $A'$, and padding $b$ with zeros to get $b'$, we need only $k+9$ bits for completing $a$ to an $m$-bit vector, $a'$, and get the equation $A' r + b' = a'$ that has a single solution $r$.

Let us now present the details of the new simulator and its corresponding oracle. In this presentation, we use the phase "$h$ is $\alpha$-good" (to be defined below), that, loosely speaking, means that $h$ maps to $a \in \{0,1\}^l$ approximately $2^{1/2} n$ of the elements of $\text{SUCC}_x$. 

The new simulator (for the oracle definition of knowledge complexity):

1. The simulator picks uniformly at random \( h_{A,b} \in H_{m,n} \) and \( \alpha \in \{0,1\}^t \). (\( t=m-k-9 \)).

2. If \( \text{rank}(A)=1 \), the simulator stops with no output.

3. The simulator completes \( A \) to a non-singular \( m \times m \) matrix \( A' \), and pads \( b \) with zeros to get \( b' \) of length \( m \).

4. The simulator sends \( (x,A',b',\alpha) \) to the oracle.

5. The oracle answers as follows:
   (5.1) If \( h \) is not \( \alpha \)-good, it returns "failure".
   (5.2) Picks at random \( r \in h^{-1}(\alpha) \cap S_2 \).
   (5.3) With probability \( P_{\alpha,r} \) (to be specified below), the oracle returns the last \( k+9 \) bits of \( A'r+b' \).
   (5.4) With probability \( 1-P_{\alpha,r} \) the oracle returns "failure".

6. If the oracle returns "failure", then the simulator stops with no output.

7. Else the simulator concatenates the bits given by the oracle to \( \alpha' \) resulting in \( \alpha' \) of length \( m \).

8. The simulator solves \( A'r+b'=\alpha' \), to get \( r \).

9. The simulator runs the original simulator on \( r \).

Analysis: Recall that \( |S_2|=2^{m-k} \) and \( t=m-k-9 \). Let us first assume, for simplicity, that the cardinality of \( h^{-1}(\alpha) \cap S_2 \) is always exactly its expected value, i.e., \( |h^{-1}(\alpha) \cap S_2|=|S_2| \cdot 2^{-t}=2^k \). Also assume that all \( h \)'s are \( h(r)-good \), and set \( P_{\alpha,r}=1 \). In this case, \( r \) (used in step (9)) is uniformly distributed on \( S_2 \): the probability of each \( r \in S_2 \) to be picked by this process is \( \text{Prob}_{h,\alpha}(r=\alpha)=\text{Prob}(r \text{ is picked from } h^{-1}(\alpha) \cap S_2) = \frac{1}{2^t} \cdot \frac{1}{2^k} = \frac{1}{2^{2k}} \). However, this simple analysis does not suffice in general since the "uniform behavior" of the cardinality of \( h^{-1}(\alpha) \cap S_2 \) cannot be guaranteed in general. Let us define \( h \) to be \( \alpha \)-good if

\[ |h^{-1}(\alpha) \cap S_2| \leq \frac{3}{2} |S_2| \cdot 2^k \cdot 2^{-k} = \frac{3}{2} |S_2| \cdot 2^k \cdot 2^{-k} \]

Definition: \( h \) is called \( \alpha \)-good if

\[ \frac{1}{2} \leq \frac{|S_2|}{2^k} \leq \frac{3}{2} \]

Let us first define the following events associated with a string \( r \in S_2 \):

- Event \( C_1(r) \) holds if \( h(r)=\alpha \) holds for the values \( h_{A,b} \) and \( \alpha \) chosen in step (1).
- Event \( R \) holds if \( \text{rank}(A)=1 \) holds for the value \( h_{A,b} \) chosen in step (1).
- Event \( G(r) \) holds if \( h \) is \( h(r)-good \) holds for the value \( h_{A,b} \) chosen in step (1).
- Event \( C_2(r) \) holds if \( r \) is chosen by the oracle in step (5.2).

Clearly, \( C_2(r) \) implies all other events (namely, \( C_1(r) \), \( R \) and \( G(r) \)) and

\[ \text{Prob}(C_2(r) \mid R \land G(r) \land C_1(r)) = \frac{1}{|h^{-1}(\alpha) \cap S_2|} \]

Also since \( H_{m,n} \) is a Universal 2 Family of Hash Function, we get \( \text{Prob}(C_1(r)) = \frac{1}{2^t} \). Let us now evaluate the probability that the simulator uses string \( r \) in step (9).

\[ \text{Prob}(\text{The simulator uses } r \in S_2) = \text{Prob}(C_2(r)) \cdot P_{\alpha,r} \]

As \( C_2(r) \) implies \( R \land G(r) \land C_1(r) \) we can write:

```
```
Using the independence between $R \land G(r)$ and $C_1(r)$ we get:

$$= \frac{1}{\lambda_{i-1}(\omega) \cap S_x} \cdot q_{r,x} \cdot 2^x \cdot P_{s,r}$$

where $q_{r,x} = \Pr(R \land G(r))$. Denote $q_{r,x} = 2^x \Pr(C_2(r) \mid R \land G(r) \land C_1(r))$, and note that by the definition of "\(a\)-good" $\frac{1}{2} \leq q_{r,x} \leq \frac{3}{2}$. We shall show, in Lemma 3.3 below, that $\Pr(C_2(r)) > \frac{1}{2} + \frac{1}{100}$ and using the fact that $\Pr(k_i < \frac{1}{100}) < \frac{1}{100} (for 1 \leq n \leq \lambda_4(\log(m)))$, and sufficiently large $m$, we get that $\frac{1}{2} \leq q_{r,x} \leq 1$. Let us set $P_{s,r}$ (the probability that the oracle does not fail in step 5.3) to be $\frac{1}{4} q_{r,x}$. Note that $P_{s,r}$ is indeed bounded by 1, as $q_{r,x} \leq \frac{1}{2}$ and $q_{r,x} \leq \frac{1}{2}$. Now, for every $r \in S_x$,

$$\Pr(r \ is \ chosen) = \frac{q_{r,x}}{2^x} \cdot q_{r,x} \cdot \frac{1}{2^x} \cdot \left(\frac{1}{4} q_{r,x} \cdot q_{r,x}^2\right) = \frac{1}{4} \cdot \frac{1}{2^x} = \frac{1}{4} \cdot \frac{1}{100}$$

In the above description, we required the oracle to make random choices, an action which it cannot standardly do. To overcome this problem, we simply let the simulator supply the oracle with coin-tosses. Another minor difficulty is that the simulator described above outputs conversations with probability $\frac{1}{2}$ (and outputs nothing otherwise). Hence, we only proved $k_{e}^{Ale(\lambda_1)}(T) \leq k_{e}^{Ale(\lambda_1)}(T) + 9$ but using Proposition 2.6 the current proposition follows.

To complete the proof, we have to show that the fraction of $h$’s which are $h(r)$-good is at least $\frac{1}{2} + \frac{1}{100}$. Namely,

Lemma 3.2: Let $H_{m,j} : \{0,1\}^n \rightarrow \{0,1\}^j$ be a universal family of hash functions, $n \geq 1+9$, and $S \subseteq \{0,1\}^m$ a set of cardinality $2^l$. Then for every $x \in S$,

$$\Pr_h (h(x) is h(x)-good) \geq \frac{1}{2} + \frac{1}{100}$$

The above statement should not be confused with the standard lemma which asserts that, for every $ae \{0,1\}^j$, most functions $h \in H_{m,j}$ are $\omega$-good.

In the proof of Lemma 3.3 we use Lemma 3.4, a standard lemma regarding hash functions (used in many complexity theory works following Sipser [SI]), and an elementary technical fact (Fact 3.5).

Lemma 3.4 (hashing a set): Let $H_{m,j}$ be a universal family of hash functions from $\{0,1\}^n$ to $\{0,1\}^j$, and $S \subseteq \{0,1\}^n$ a set of cardinality $2^l$.

$$\Pr_h (\exists \exists \exists \in S \land h(\exists) = a) \leq \frac{1}{2^j} \left(\frac{1}{2^j} \right) \leq \frac{1}{2^{2l+1}}$$

Fact 3.5: Let $m \geq n > 5$ be two integers, and $b > 1.25$ be a real number, then:

- $m \\ n \\ b > 5$ is two integers, and $b > 1.25$ be a real number, then:
Now, let us prove Lemma 3.3. Intuitively, we reduce this problem to the case of Lemma 3.4 by considering hash functions from \( \{0,1\}^m \) to \( \{0,1\} \). Clearly, in this special case, if there exists an \( \alpha \in \{0,1\} \) such that \( h \) is \( \alpha \)-good then for all \( \alpha \in \{0,1\} \) \( h \) is \( \alpha \)-good, and in particular, we get that for every \( r \in \{0,1\}^m \), \( h \) is \( r \)-good. Denote by \( [h(y)]_i \) the projection of \( h(y) \) on the \( i \)-th position (i.e., if \( h(y) = \sigma_1 \sigma_2 \ldots \sigma_i \), then \( [h(y)]_i = \sigma_i \)). Clearly, for every \( i \) and every pair of strings \( y, z \in \{0,1\}^m \), when \( h \) is chosen uniformly in \( H_{m,1} \), the random variables \( [h(y)]_i, [h(y)]_2, \ldots, [h(y)]_n \) and \( [h(z)]_1, [h(z)]_2, \ldots, [h(z)]_n \) are totally independent and uniformly distributed in \( \{0,1\} \). Fix \( x \in S \), and consider, for each \( h \in H_{m,1} \), the following process of determining a subset \( S_i \subseteq S_i \), in \( l \) steps. \( S_0 \triangleq S \), and \( S_i \triangleq \{ y \in S_{i-1} : [h(y)]_i = [4(x)]_i \} \) for \( i = 1, 2, \ldots, l \). For every \( i \), we may apply Lemma 3.4 (use \( b = 1 \) and \( d_i \) instead of \( d \)) and conclude that for all but a \( \frac{2d_i}{|S_{i-1}|} \) fraction of the \( h \in H_{m,1} \) we have

\[
\forall i \quad 1 \leq i \leq l \quad \frac{1}{2} |s_{i-1}| \left(1 - \frac{1}{2^d} \right) \leq |s_i| \leq \frac{1}{2} |s_{i-1}| \left(1 + \frac{1}{2^d} \right).
\]

Setting \( d_i = \frac{n-i}{3} \) and using Fact 3.5, we are done.

The following lemma is a generalization of Lemma 3.3, which might be useful in other settings.

**Lemma 3.6**: Let \( H_{m,1} \) be a universal family of hash functions from \( \{0,1\}^m \) to \( \{0,1\} \), let \( S \) be a subset of \( \{0,1\}^m \) and let \( n \geq \lceil \log (|S|) \rceil \), then, for all \( x \in \{0,1\}^m \) and all \( 0 \leq d \leq n - l - 9 \)

\[
\text{Prob}_{h \in H_{m,1}} \left( \left| \{ y \in S : h(y) = h(x) \} \right| > \frac{1}{2} |S| \right) < 2^{-\frac{2(n-d-9)}{3}}.
\]

Proof: similar to the proof of Lemma 3.3.

### 3.2. Oracle vs. Average Oracle

**Proposition 3.7**: For any interactive proof \( \Pi \), \( k_{\text{Oracle}}(\Pi) \leq k_{\overline{\text{Oracle}}}^{\overline{\Pi}}(\Pi) + 2 \)

Proof: Suppose we have a simulator of the oracle type that queries the oracle \( k_{\text{Oracle}}(\Pi) \) times, and outputs a conversation with probability \( \geq \frac{1}{2} \). Our new simulator will choose random coin-tosses for the original simulator and ask the oracle whether these coins are "good", i.e., whether on this random string the original simulator (querying the original oracle) outputs a conversation (rather than nothing). If the oracle says "yes", the original simulator is run on these coin-tosses, and its \( k_{\text{Oracle}}(\Pi) \) queries are answered by the oracle. Otherwise, the new simulator tries again. The new simulator allows itself up to \( p(1x) \) tries, where \( p \) is a polynomial bounding the length of the conversations in \( \Pi \). If all tries fail, it asks the oracle for a conversation. The expected number of additional queries made by the simulator is at most 2.

In light of the results in Section 4, it is not possible to bound the opposite direction (i.e., it is not even possible to prove that \( k_{\overline{\text{Oracle}}}^{\overline{\Pi}}(\Pi) \geq k_{\text{Oracle}}(\Pi) + p(1x) \) for some polynomial \( p \) and all \( \Pi \)'s).

---

3.3. The oracle and the strict oracle versions are close.

Clearly, as stated in Proposition 2.5, \( k_{compl}^{oracle}(\Pi) \leq k_{compl}^{strict}(\Pi) \). Proposition 3.8 gives us a complementary relation.

Proposition 3.8: For any interactive proof \( \Pi \) and any unbounded \( g: \mathbb{N} \to \mathbb{N} \), \( k_{compl}^{oracle}(\Pi) \leq k_{compl}^{strict}(\Pi) + \log(\log(1 \cdot \xi)) + g(1 \cdot \xi) \).

Proof: Use Proposition 2.7 with \( e = \frac{1}{1 \cdot \xi f(1 \cdot \xi)} \), where \( f(1 \cdot \xi) = \min\{2^k(1 \cdot \xi), 1 \cdot \xi\} \). Note that Proposition 2.7 holds as long as \( e > 2^{-poly(1 \cdot \xi)} \). Modify the resulting simulator so that it always produces an output. \( \square \)

In light of the results in Section 4, this result cannot be improved in general.

4. Separation results

In this section we provide separation results for the knowledge complexity of specific interactive proof systems. We stress that these results do not necessarily translate to a separation of languages according to their knowledge complexity (which would have implied separation between \( \text{BPP} \) and \( \text{PSPACE} \)).

We first show a separation in each of the knowledge complexity hierarchies.

Proposition 4.1: For every polynomial time computable \( k : \mathbb{N} \to \mathbb{N} \), there exists an interactive proof system \((\text{P} \leftrightarrow \text{V})\) (for \((0,1)^*\)) satisfying:

1. \((\text{P} \leftrightarrow \text{V})\) has knowledge complexity \( k(1 \cdot \xi) \) in the hint sense (and all other senses considered in Section 2).
2. \((\text{P} \leftrightarrow \text{V})\) has knowledge complexity at least \( k(1 \cdot \xi) \) in the hint sense.
3. \((\text{P} \leftrightarrow \text{V})\) has knowledge complexity at least \( k(1 \cdot \xi) \) in the strict oracle sense.
4. \((\text{P} \leftrightarrow \text{V})\) has knowledge complexity at least \( k(1 \cdot \xi) \) in the oracle sense.
5. For any polynomial \( p \), the interactive proof system \((\text{P} \leftrightarrow \text{V})\) has knowledge complexity greater than \( k(1 \cdot \xi) \cdot \frac{1}{p(1 \cdot \xi)} \) in the fraction sense.
6. For any polynomial \( p \), the interactive proof system \((\text{P} \leftrightarrow \text{V})\) has knowledge complexity greater than \( k(1 \cdot \xi) \cdot 2 \cdot \frac{1}{p(1 \cdot \xi)} \) in the average oracle sense.

Proof: Consider the interactive proof in which, on input \( x \), the prover sends to the verifier a string \( K(x) \). Clearly, the protocol has knowledge complexity \( k(1 \cdot \xi) \) in the hint sense (and all other senses considered in Section 2).

We first prove part (5) of the theorem, next, part (4) (and similarly part (3) & (2)), and conclude by proving part (6).

We start by showing that for an appropriate choice of a function \( K \), the above interactive proof has knowledge complexity greater than \( k(1 \cdot \xi) \cdot \frac{1}{p(1 \cdot \xi)} \) in the fraction sense. The function \( K \) is constructed using the diagonalization technique. Consider an enumeration of all probabilistic polynomial time machines. For each machine \( M \), select a different input \( x \), and consider the output distribution \( M(x) \). If machine \( M \) simulates the interactive proof properly, then \( M \), on input \( x \), must output \( K(x) \) with probability \( \geq \frac{1}{2^k(1 \cdot \xi) (1 - \epsilon)} \), where \( \epsilon \) is a negligible fraction. Note that \( \frac{1}{2^k(1 \cdot \xi) (1 - \epsilon)} > \frac{1}{2^k(1 \cdot \xi)} \) since \( \epsilon \) is smaller than any polynomial fraction. By \( a \)
pigeon hole argument, there is a string \( y \) of length \( k(lxI) \), that appears in \( M(x) \) with probability \( \leq \frac{1}{2^{k(lxI)}} \), and setting \( K(x)=y \) foils the simulation of \( M \) on \( x \).

Next, let us prove part (4). Namely, we show how to foil all oracle machines that never ask more than \( k(lxI) \) questions and must produce an output with probability \( \geq \frac{1}{2} \). We consider an enumeration of all probabilistic polynomial time oracle machines, and select a different input \( x \) for each machine \( M \). As a mental experiment, consider the output distribution of machine \( M \) on input \( x \) and access to a random oracle \( \alpha \).

Again, by a pigeon hole argument, there is a string \( y \) of length \( k(lxI) \), that appears in this distribution with probability less or equal to \( \frac{1}{2^{k(lxI)}} \). We show that for any specific oracle \( \alpha \), the probability that the string \( y \) appears in the output distribution of machine \( M \) on input \( x \) and access to the oracle \( \alpha \) is \( \frac{1}{4} \) at the most. Let \( k=k(lxI) \) and let \( m=m(lxI) \) be a bound on the running time of machine \( M \) on input \( x \) (and therefore, a bound on the number of its coin-tosses). Denote by \( r \in \{0,1\}^m \) the coin-tosses of machine \( M \), and by \( A(x,r,\alpha) \) the \( k-2 \) answers that oracle \( \alpha \) gives machine \( M \) on input \( x \) and random tape \( r \). Note that \( x,r \) and \( A(x,r,\alpha) \) completely determine the behavior of machine \( M \). Also note that the random oracle \( \alpha \) is chosen independently of \( r \). Let us denote the output of machine \( M \) on input \( x \), random string \( r \) and access to oracle \( \alpha \) by \( M_{r,\alpha}(x) \).

The probability that the string \( y \) appears in the output distribution of machine \( M \) with access to a specific oracle \( \alpha_0 \) is

\[
\Pr_{r,\alpha}(M_{r,\alpha}(x)=y) = \Pr_{r,\alpha}(M_{r,\alpha}(x)=y | A(x,r,\alpha)=A(x,r,\alpha_0)) \leq \frac{\Pr_{r,\alpha}(M_{r,\alpha}(x)=y)}{\Pr_{r,\alpha}(A(x,r,\alpha)=A(x,r,\alpha_0))}
\]

we show now that the denominator is equal to \( \frac{1}{2^{k-2}} \).

\[
\Pr_{r,\alpha}(A(x,r,\alpha)=A(x,r,\alpha_0)) = \frac{1}{2^m} \sum_{r \in \{0,1\}^m} \Pr_{r,\alpha}(A(x,r,\alpha)=A(x,r,\alpha_0))
\]

As \( r \) (and \( x \)) are constants inside the summation, a random choice of \( k-2 \) oracle answers agrees with the answers of the specific oracle \( \alpha_0 \) with probability \( \frac{1}{2^{k-2}} \) and we can write:

\[
\frac{1}{2^m} \sum_{r \in \{0,1\}^m} \frac{1}{2^{k-2}} = \frac{1}{2^{k-2}}
\]

Thus, for any specific oracle \( \alpha_0 \), the string \( y \) appears in the output distribution of \( M_{r,\alpha}(x) \) with probability \( \frac{1}{4} \) at the most. Selecting \( K(x)=y \), makes \( M \) fail with probability \( \frac{3}{4} \) and therefore an improper simulator for \( \Pi \) on \( x \).

The bound for the strict oracle version is proven similarly (recall that a strict oracle machine must always produce an output) and implies in turn the bound for the hint version.

Next, let us prove part (6). Namely, we show how to foil all oracle machines which make at most \( k(lxI) \) queries on the average, for any polynomial \( p \). Consider an enumeration of all probabilistic polynomial time oracle machines, and select a different input \( x \) for each machine \( M \). Let \( k=k(lxI) \) and \( p=p(lxI) \). Let \( \mu \) denote the random variable that represents the number of queries that \( M \) makes on \( x \). We
first show that for all 0 ≤ i < \log (k \cdot p), \Pr(μ < k - i) < 2^{-i}, and that for all \log (k \cdot p) ≤ i < k, \Pr(μ < k - i) < \frac{1}{k \cdot p}, and next use this fact to derive \( E(μ) > k - \frac{1}{p} \). Assume, by way of contradiction, that there exists an i such that 
\( \log (k \cdot p) ≤ i < k \) and \( \Pr(μ < k - i) > \frac{1}{k \cdot p} \). Consider the behavior of M after making \( k - i - 1 \) queries to the oracle.

Machine M may either query more, or halt outputing a string without making further queries. By hypothesis, the later happens with probability at least \( \frac{1}{k \cdot p} \). We use this polynomial fraction of the output distribution to fail the simulation; namely, we show that for an appropriately chosen \( K(x) \), half of this (polynomial) fraction of the output distribution is different from \( K(x) \) no matter what the oracle set is. Again, consider the output distribution of M on input \( x \) and access to a random oracle \( a \). By the pigeon hole principle, there is a string \( y \) of length \( k \), that appears in this distribution with probability less or equal to \( \frac{1}{2^k} \).

Again, a random choice of \( k - i - 1 \) answers agrees with the answers of any specific oracle set \( a \) with probability \( \frac{1}{2^k} \). Therefore, for any specific oracle \( a \), the string \( y \) appears in the output distribution of \( M_a(x) \) with probability \( \frac{1}{2^k} \) at the most. Selecting \( K(x) = y \), we get that for every oracle \( a \) the probability that \( M_a(x) \) does not output \( K(x) \) is at least \( \frac{1}{k \cdot p} \). This implies that \( M_a(x) \) fails to simulate \( \Pi \) on \( x \). Now, that we have not partitioned the possible i's into two cases, we would have only got a failure probability of \( 2^{-i} - 2^{-i+1} \) which is not necessarily a polynomial fraction of the output distribution. A similar argument for \( 0 ≤ i < \log (k \cdot p) \) shows that we can choose \( K(x) = y \) so that \( M \) fails to simulate on input \( x \) with probability at least \( \frac{1}{2^k} \).

Now, if for all \( i, 0 ≤ i < \log (k \cdot p) \), i satisfies \( \Pr(μ < k - i) < 2^{-i} \) and for all \( i, \log (k \cdot p) ≤ i < k \), i satisfies \( \Pr(μ < k - i) < \frac{1}{k \cdot p} \) then:

\[
E(μ) ≥ k - \sum_{i=0}^{k-1} \Pr(μ ≤ i) ≥ k - \sum_{i=0}^{\log (k \cdot p) - 1} \frac{1}{k \cdot p} - \sum_{i=\log (k \cdot p)}^{k-1} 2^{-i+1} > k - \frac{2^i - 1}{p}.
\]

Next we show that the gap between the average oracle and the other oracle versions, cannot be bounded, even by a polynomial.

\[ \text{Proposition 4.2: For every polynomial time computable } k(1/x) \text{ and every non-negligible (and polynomial-time computable) fraction } p(1/x) \text{ (i.e., } \exists c > 0 \text{ such that } p(1/x) > \frac{1}{1+c} \text{), there exists an interactive proof system (for } 0,1^* \text{) satisfying:} \]

1. \( (P \leftrightarrow V) \) has knowledge complexity \( Sp(1/x) k(1/x) \) in the average Oracle sense.
2. \( (P \leftrightarrow V) \) has knowledge complexity greater than \( k(1/x) - 2^{\log \frac{1}{p(1/x)}} \) in the Oracle sense.
Proof sketch: We consider an interactive proof in which, on input $x$, with probability $p(IxI)$ the prover sends $K(x)$ to the verifier (and otherwise sends nothing). Clearly, for $|K(x)| \leq k(IxI)$, this interactive proof has average knowledge complexity bounded by $p(IxI)k(IxI)$ in the Oracle sense. We now show that for an appropriate choice of $K$, the above interactive proof has knowledge complexity greater than $t(IxI)2^{-k(IxI)}$ in the Oracle sense. Again, we make a mental experiment, considering the output of $M$ on input $x$ and access to a random oracle $a$. We set $K(x)=y$, where $y$ is a $k(IxI)$-bit string satisfying $\Pr(M^a(x)=y) \leq 2^{-k(IxI)}$ (for a random oracle $a$). As each oracle $a$ has probability $2^{-1}(IxI)$ to coincide with $t(IxI)$ random answers, it follows that for any specific oracle set $a$, we have $\Pr(M^a(x)=y) \leq 2^{-k(IxI)}2^{-k(IxI)} = 2^{(IxI)}/4$ and hence $M$ fails to simulate $P$ (no matter which oracle is used to answer $M$’s queries).

Interestingly, the protocol appearing in the above proof, has knowledge complexity $k(IxI)+1-\log(1/(IxI))$ even in the strict oracle measure. The simulator (for the strict oracle version) guesses the first $\log(1/(IxI))$ bits of $K(x)$ and sends them to the oracle, which replies whether this guess is correct. In case it is not, the simulator outputs nothing. Otherwise, it queries the oracle for the remaining $k(IxI)-\log(1/(IxI))$ bits of $K(x)$. Thus, the simulator makes at most $k(IxI)-\log(1/(IxI))+1$ queries, and output $K(x)$ with probability $1/p(IxI)$ as it should.

The following proposition shows a huge gap between the hint version and the strict oracle version of knowledge complexity.

Proposition 4.3: There exists an interactive proof $\Pi$ that has knowledge complexity 1 in the strict Oracle sense, yet for any polynomial $p$, satisfies $k_{\mbox{hint}}(\Pi) > p(IxI)$.

proof: The protocol is the following: $V$ sends a random string $r$ in $[0,1]^{\leq 1}$ to $P$, which responds with a single bit $K(x,r)$. Clearly, for any $K$, this protocol yields one bit of knowledge in the strict Oracle sense. Let $n=IxI$. We build a function $K(x,r)$ such that any probabilistic polynomial time machine (PPTM) that uses a hint of length less than $t(n)$ fails to simulate $\Pi$. We consider an enumeration of all probabilistic polynomial time "hint machines" and for each machine $M$ we select a different input $x$, on which $M$ fails to simulate $\Pi$. Let us consider the distributions $M(x,h)$ for all possible hints $h \in \{0,1\}^{\leq t(n)}$. For each $h \in \{0,1\}^{\leq t(n)}$, define a function $f^{(h)}: \{0,1\}^{\leq t(n)} \rightarrow \{0,1\}$ so that $f^{(h)}(r)=0$ iff $\Pr(M(x,h)=(r,0))<\Pr(M(x,h)=(r,1))$ (set $r=0$ if the probabilities are equal). Clearly, $M(x,h)$ does not simulate the protocol properly if $f^{(h)}(r)$ is not "close" to $K(x,r)$. We show that there is a function that is not close to any $f^{(h)}$ ($h \in \{0,1\}^{\leq t(n)}$), and thus, by selecting this function for $K(x)$, we foil the simulation of $M$ on $x$ (no matter which $h$ is used as a hint). We say that a function $f$ is close to a function $g$ if they disagree on less than $1/6$ of their possible inputs. We use a counting argument to show that there exists a function $K(x,r): \{0,1\}^{\leq t(n)} \rightarrow \{0,1\}$ which is not close to any of the possible $f^{(h)}$'s. The number of possible $f^{(h)}$'s is $2^{2t(n)}$ (this is the number of possible $h$'s), and the number of functions that agree with a specific $f^{(h)}$ on at least $5/6$ of the $r$'s is bounded by $l(n)2^{(n)} \leq 2^{(n)}$. Therefore, the total number of functions that are close to $f^{(h)}$ for some $h \in \{0,1\}^{\leq t(n)}$ is less than $l(n)2^{2t(n)} < 2^t$, and we can select a function $K(x,r)$ to foil the simulation of $M$ on $x$. \qed
Proof idea: It suffices to consider the oracle version. We apply the construction of [GK] to each of the possible oracle answers. This requires some changes in the original proof. Details are omitted here.

Corollary: Languages with constant round AM protocols that have a negligible error probability (i.e., smaller than $\frac{1}{|x|^c}$ for any $c > 0$ and sufficiently long $x$) and that leak $O(\log(|x|))$ bits of knowledge (in the black box model) are in BPP.

In a similar way, we can extend Theorem 2 in [GK] regarding IP[3].

7. Acknowledgement:
We are grateful to Benny Chor, Johan Hastad, Hugo Krawczyk, and Eyal Kushilevitz for helpful discussions.

8. References:

Appendix: Inadequacies in the GMR definition.

In [GMR] the knowledge complexity of an interactive proof $(P \leftrightarrow V)$ is said to be $k(1x1)$ if there exists a simulator which can generate a distribution $M(x)$ such that the variation distance of $M(x)$ and $(P \leftrightarrow V)(x)$ is bounded by $1-2^{-k(1x1)+1x1 <}$ for all constants $c > 0$ and sufficiently long $x$'s. Recall that the variation difference of the random variables $Y$ and $Z$ is $\sum_a |\text{Prob}(Y=a) - \text{Prob}(Z=a)|$. Hence, any prover which with probability $\frac{1}{2}$ sends nothing, leaks (by this definition) at most 1 bit of knowledge. Also, by this definition, all interactive proofs have knowledge complexity bounded by $k(1x1) = \frac{\log(1x1)}{o(1)}$. 