Quantifying Knowledge Complexity*

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.

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 ½ 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 $\frac{log(|x|)}{o(1)}$. In fact, all interactive proofs leak only $\frac{\log |x|}{o(1)}$ 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

* 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@techunix.technion.ac.il

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 $k(|x|)$ 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 $k(|x|)$ bits. Hence, the computational advantage gained by the interaction with the prover who may send much more than $k(|x|)$ bits can be simulated by an interaction with a machine which only sends $k(|x|)$ 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 so 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 our other definitions of knowledge complexity).(2)

In order to summarize our results concerning the relations between the various definitions, we present the following notations. Let $kc^{Strict}_{Oracle}(\Pi)$, $\overline{kc}_{Oracle}(\Pi)$ and $kc^{1/2}_{Oracle}(\Pi)$ denote the knowledge complexity of an interactive proof $\Pi=(P\leftarrow\rightarrow V)$ according to the strict, average and "intermediate" (i.e., output with probability $1/2$) variants of the oracle approach. Likewise, $kc_{Fraction}(\Pi)$ and $kc_{Hint}(\Pi)$ denote the knowledge complexity of Π 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 Π, $\overline{kc}_{Oracle}(\Pi)-2 \le kc^{1/2}_{Oracle}(\Pi) \le kc^{Strict}_{Oracle}(\Pi) \le kc_{Hint}(\Pi)$

(2) For every interactive proof Π, $\left| kc^{1/2}_{Oracle}(\Pi) - kc_{Fraction}(\Pi) \right| = O(1)$.(3)

(3) For every interactive proof Π, $kc^{Strict}_{Oracle}(\Pi) \le kc^{1/2}_{Oracle}(\Pi)+log(log(|x|))+\frac{1}{o(1)}$; whereas there exists an interactive proof Π such that $kc^{1/2}_{Oracle}(\Pi)=1$, and $kc^{Strict}_{Oracle}(\Pi) \ge kc^{1/2}_{Oracle}(\Pi)+log(log(|x|))+\Omega(1)$.

(4) For every polynomial $p(|x|)$, there exists an interactive proof Π such that $\overline{kc}_{Oracle}(\Pi) \le \frac{1}{p(|x|)}$, and $kc^{1/2}_{Oracle}(\Pi) \ge p(|x|)$.

(5) There exists an interactive proof Π such that $kc^{Strict}_{Oracle}(\Pi) = 1$, and $kc_{Hint}(\Pi) > p(|x|)$ for any polynomial p.

(6) For every polynomial p and every polynomial-time computable function $k(|x|)$, there exists an interactive proof Π such that $kc(\Pi) \le k(|x|)$ by all measures, yet $kc(\Pi) > k(|x|)-3$ by all measures.

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 [GMW]) consists of many *sequential* executions of a basic protocol. In contrast to some common belief, Goldreich and Krawczyk [GK] showed that the parallel executions of these protocols cannot be shown zero-knowledge using a black-box argument(4) (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 $\varepsilon(|x|)$ for a language L cannot be shown (using a black-box argument) to have knowledge complexity $k(|x|) \le \frac{1}{20}\cdot\left(\log\frac{1}{\varepsilon(|x|)}-|x|\right)$, unless L can be recognized in probabilistic time $|x|^{O(1)}\cdot 2^{20\cdot k(|x|)}$. (This generalizes [GK]'s result in which $k(|x|)=0$).

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) This measure of knowledge was originally suggested in [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 shown 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.

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,IY]. 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 $(P \longleftrightarrow V)(x)$ 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, $k: \mathrm{N} \to \mathrm{N}$, is polynomial time computable if there exists a probabilistic polynomial time algorithm that on input 1^n outputs $k(n)$ with probability $\frac{2}{3}$ (equivalently $1-2^{-n}$).

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 $k: \mathrm{N} \to \mathrm{N}$ be a function over the integers. We say that an interactive proof system $\Pi=(P \longleftrightarrow V)$ 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 $(P \longleftrightarrow V')(x)$. The *knowledge complexity in the hint sense* is an operator, denoted kc_{Hint}, which assigns each interactive proof system Π a function k_Π so that for any $x \in L$, Π can be simulated using a hint of length $k_\Pi(|x|)$. The class $KC_{Hint}(k(|x|))$ 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" $\notin \{0,1\}$) 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 Π 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 $x \in L$, $M_{V'}^{A}(x)$ is statistically close to $(P \longleftrightarrow V')(x)$.

The *knowledge complexity in the strict oracle sense,* denoted kc_{Oracle}^{Strict} (also kc_{Oracle}^{1}), and the class KC_{Oracle}^{Strict} (also KC_{Oracle}^{1}) 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 Π, $kc_{Oracle}^{Strict}(\Pi) \leq kc_{Hint}(\Pi)$.

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 $M_{V'}^{A}$ produces an output with probability $>1/2$, and given that $M_{V'}^{A}$ has produced an output, $M_{V'}^{A}(x)$ is statistically close to $(P \longleftrightarrow V')(x)$.

The *knowledge complexity in the oracle sense,* denoted $kc^{1/2}_{Oracle}$ (also kc_{Oracle}) and the class $KC^{1/2}_{Oracle}$ (also KC_{Oracle}) are defined similarly to the previous definitions.

Clearly,

Proposition 2.5: For every interactive proof Π, $kc^{1/2}_{Oracle}(\Pi) \leq kc^{Strict}_{Oracle}(\Pi)$.

The constant ½ 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 $kc^{\delta}_{Oracle}(|x|)$. The relation of this definition to the original one is given in the following two proposition:

Proposition 2.6: For every interactive proof Π, and every $0<\varepsilon<\frac{1}{2}$, $kc^{1/2}_{Oracle}(\Pi) \leq kc^{\varepsilon}_{Oracle}(\Pi)+\left\lceil \log\left(\frac{1}{\varepsilon}\right)\right\rceil$

Proof: We build a new simulator that picks $2^{\left\lceil \log\left(\frac{1}{\varepsilon}\right)\right\rceil}-1$ 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 $(1-\varepsilon)^{2^{\left\lceil \log\left(\frac{1}{\varepsilon}\right)\right\rceil}-1} < \frac{1}{2}$. □

Proposition 2.7: For every interactive proof Π, and every $0<\varepsilon<\frac{1}{2}$, $kc^{1-\varepsilon}_{Oracle}(\Pi) \leq kc^{1/2}_{Oracle}(\Pi)+\lceil \log\lceil log\left(\frac{1}{\varepsilon}\right)\rceil\rceil$.

Proof: Similar. This time the list is of length $\lceil log\left(\frac{1}{\varepsilon}\right)\rceil$, the failure probability is $\left(\frac{1}{2}\right)^{\lceil log\left(\frac{1}{\varepsilon}\right)\rceil} \leq \varepsilon$, and a pointer to an element in this list requires $\lceil \log\lceil \log\left(\frac{1}{\varepsilon}\right)\rceil\rceil$ 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 $M_{V'}$ makes to oracle A is at most $k(|x|)$.

The *average knowledge complexity in the oracle sense,* denoted kc_{Oracle} and the class KC_{Oracle} are defined analogously to definition 2.1.

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

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

(1) $PROB(SUCC_x) \geq \rho(|x|)$.

(2) For every $x \in L$, when the coin-tosses for machine $M_{V'}$ are picked uniformly from $SUCC_x$, $M_{V'}(x)$ is statistically close to $(P \leftrightarrow V')(x)$.

The *knowledge complexity in the fraction sense,* denoted $kc_{Fraction}$, assigns the interactive proof Π a function k_Π so that Π can be simulated at density 2^{-k_Π}. The class $KC_{Fraction}(k(|x|))$ is defined analogously to the previous definitions.

3. Inclusion results

For every interactive proof Π, $kc^{1/2}_{Oracle}(\Pi) \leq kc^{Strict}_{Oracle}(\Pi) \leq kc_{Hint}(\Pi)$ (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 Π, $kc_{Fraction}(\Pi) \leq kc^{1/2}_{Oracle}(\Pi)+1$

Proof sketch: Let $k=kc^{1/2}_{Oracle}(\Pi)$. we build a simulator that guesses a random string for the original simulator and then, guesses the oracle-answers. With probability at least $\frac{1}{2^{-(k+1)}}$, 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 Π with polynomial time computable knowledge complexity in the fraction sense, $kc^{1/2}_{Oracle}(\Pi) \leq kc_{Fraction}(\Pi)+11$. Furthermore, if $kc_{Fraction}(\Pi)=O(log\ |x|)$, then $kc^{1/2}_{Oracle}(\Pi) \leq kc_{Fraction}(\Pi)+1$.

Proof: Let $k=kc_{Fraction}(\Pi)$. Our purpose is to pick a random $r \in SUCC_x$ and run the original simulator on r. The problem is how to sample $SUCC_x$. The naive solution of asking the oracle to pick an $r \in_R SUCC_x$, 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 $SUCC_x$. 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 $k=O(\log |x|)$. The probability of failing (not having any good string in the list) is $(1-\frac{1}{2^k})^{2^k} < \frac{1}{2}$. Hence in this case, $(k=O(\log |x|))$, we have $kc^{1/2}_{Oracle}(\Pi) \le kc_{Fraction}(\Pi)+1$.

Dealing with $k=\frac{\log |x|}{o(1)}$ 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$_2$ 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 $\{0,1\}^m \to \{0,1\}^l$, 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 $h_{A,b}(r) = A \cdot r + b$ (where all operations are done over GF[2]). We use two properties of this family. The universal$_2$ 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 $\alpha \in \{0,1\}^l$. The list that is implicitly chosen is $h^{-1}(\alpha)$, i.e., the affine space of vectors r that satisfies the equation $A \cdot r + b = \alpha$. 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 $k+9$ 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 α to an m-bit vector, α', and get the equation $A' \cdot r + b' = \alpha'$ 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 α-good" (to be defined below), that, loosely speaking, means that h maps to $\alpha \in \{0,1\}^l$ approximately $2^{-l} \cdot |S_x|$ of the elements of $S_x \triangleq SUCC_x$.

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

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

(2) If rank$(A) \neq l$, 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', α) to the oracle.

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

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

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

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

Definition: h is called α-*good* if $\frac{1}{2} \cdot \frac{|S_x|}{2^l} \le |h^{-1}(\alpha) \cap S_x| \le \frac{3}{2} \cdot \frac{|S_x|}{2^l}$.

In our technical report [GP] we show that if we set $P_{x,r}$ appropriately then the probability that an $r \in S_x$ is chosen in the above process is exactly $\frac{1}{4} \cdot \frac{1}{|S_x|}$. The analysis uses Lemma 3.3 which asserts that for all $r \in S_x$ the probability (under a uniformly chosen $h \in H_{m,l}$) that $h(r)$ is $h(r)$-good, is at least $\frac{1}{2} + \frac{1}{100}$. Namely,

Lemma 3.3: Let $H_{m,l}: \{0,1\}^m \to \{0,1\}^l$ be a universal$_2$ family of hash functions, $n \ge l+9$, and $S \subseteq \{0,1\}^m$ a

set of cardinality 2^n. Then for every $x \in S$,

$$\text{Prob}_h\left(h \text{ is } h(x)\text{-}good\right) \geq \frac{1}{2}+\frac{1}{100}$$

The above statement should not be confused with the standard lemma which asserts that, for every $\alpha \in \{0,1\}^l$, most functions $h \in H_{m,l}$ are α-*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 [S]), and an elementary technical fact (Fact 3.5).

Lemma 3.4 (hashing a set): Let $H_{n,b}$ be a universal$_2$ family of hash functions from $\{0,1\}^n$ to $\{0,1\}^b$ and let S be a subset of $\{0,1\}^n$, then, for all $\alpha \in \{0,1\}^b$,

$$\text{Prob}_h\left(\left| \left|\{x: x \in S \wedge h(x)=\alpha\}\right| - \frac{|S|}{2^b}\right| > \frac{|S|}{2^{b+d}}\right) < \frac{2^{2d+b}}{|S|}\cdot\left(1-\frac{1}{2^b}\right)$$

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

(1) $\prod_{i=a}^{m}(1-b^{-i}) \geq e^{-\frac{b^{-a}}{1-b^{-1}}-\frac{b^{-2a}}{1-b^{-2}}}$

(2) $\prod_{i=a}^{m}(1+b^{-i}) \leq e^{\frac{b^{-a}}{1-b^{-1}}}$

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 α-good then for all $\alpha \in \{0,1\}$ h is α-good, and in particular, we get that for every r, h is $h(r)$-good. Denote by $[h(y)]_i$ the projection of $h(y)$ on the i-th position (i.e., if $h(y)=\sigma_1\sigma_2....\sigma_l$ 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 $H_{m,l}$, the random variables $[h(y)]_1, [h(y)]_2, ...,[h(y)]_i$ and $[h(z)]_1, [h(z)]_2, ...,[h(z)]_i$ are totally independent and uniformly distributed in $\{0,1\}$. Fix $x \in S$, and consider, for each $h \in H_{m,l}$, the following process of determining a subset $S_l \subseteq S$, in l steps. $S_0 \triangleq S$, and $S_i \triangleq \{y \in S_{i-1} : [h(y)]_i=[h(x)]_i\}$ for $i=1,2,...l$. 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 $\frac{2^{2d_i}}{|S_{i-1}|}$ fraction of the $h \in H_{m,l}$ we have $\forall i \ 1 \leq i \leq l$ $\frac{1}{2}|S_{i-1}|\left(1-\frac{1}{2^{d_i}}\right) \leq |S_i| \leq \frac{1}{2}|S_{i-1}|\left(1+\frac{1}{2^{d_i}}\right)$. Setting $d_i = \frac{n-i}{3}$ and using Fact 3.5, we are done. □

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 $\frac{1}{4}$ (and outputs nothing otherwise). Hence, we only proved $kc^{1/4}_{Oracle}(\Pi) \leq kc_{Fraction}(\Pi)+9$ but using Proposition 2.6 the current proposition follows.

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

Lemma 3.6: Let $H_{m,l}$ be a universal$_2$ family of hash functions from $\{0,1\}^m$ to $\{0,1\}^l$, let S be a subset of $\{0,1\}^m$ and let $n \triangleq \lfloor log(|S|)\rfloor$, then, for all $x \in \{0,1\}^m$ and all $0 \leq d \leq n-l-9$

$$\text{Prob}_h\left(\left| \left|\{y: y \in S \wedge h(y)=h(x)\}\right| - \frac{|S|}{2^l}\right| > 2^{\frac{n-l-d-10}{3}}\cdot\frac{|S|}{2^l}\right) < 2^{-\frac{n-l+2d-10}{3}}$$

Proof: similar to the proof of Lemma 3.3.

3.2. Oracle vs. Average Oracle

Proposition 3.7: For any interactive proof Π, $\overline{kc}_{Oracle}(\Pi) \leq kc^{1/2}_{Oracle}(\Pi)+2$

Proof: Suppose we have a simulator of the oracle type that queries the oracle $k(|x|)$ 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 then nothing). If the oracle says "yes", the original simulator is run on these coin-tosses, and its $k(|x|)$ queries are answered by the oracle. Otherwise the new simulator tries again. The new simulator allows itself up to $p(|x|)$ tries, where p is a polynomial bounding the length of the conversations in Π. If all the 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 $kc^{1/2}_{Oracle}(\Pi) \leq \overline{kc}_{Oracle}(\Pi)+p(|x|)$ for some polynomial p and all Π's).

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

Clearly, as stated in Proposition 2.5, $kc^{Strict}_{Oracle}(\Pi) \geq kc^{1/2}_{Oracle}(\Pi)$. Proposition 3.8 gives us a complementary relation.

Proposition 3.8: For any interactive proof Π and any unbounded $g: N \to N$, $kc^{Strict}_{Oracle}(\Pi) \leq kc^{1/2}_{Oracle}(\Pi)+log(log(|x|))+g(|x|)$.

Proof: Use Proposition 2.7 with $\varepsilon=\frac{1}{|x|^{f(|x|)}}$, where $f(|x|)=\min\{2^{g(|x|)}, |x|\}$. Note that Proposition 2.7 holds as long as $\varepsilon>2^{-poly(|x|)}$. Modify the resulting simulator that it always produces an output. □

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 BPP and PSPACE).

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

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

(1) $(P \longleftrightarrow V)$ has knowledge complexity $k(|x|)$ in the Hint sense (and all other senses).

(2) $(P \longleftrightarrow V)$ has knowledge complexity greater than $k(|x|)-3$ under each of the definitions considered in this paper.

Proof sketch: Consider the interactive proof in which, on input x, the prover sends to the verifier $K(x)$. Clearly, the protocol has knowledge complexity $\leq k(|x|)$ in the Hint sense for any function $K: \{0,1\}^* \to \{0,1\}^*$ satisfying $|K(x)| \leq k(|x|)$. We now show that for an appropriate choice of a function K, the above interactive proof has knowledge complexity greater than $k(|x|)-1$ 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 $M(x)$. Set $K(x)=y$, where y is a $k(|x|)$-bit string satisfying $\text{Prob}(M(x)=y) \leq 2^{-k}$ (such a string necessarily exists). Hence, M cannot simulate the protocol on input x with density $2^{-(k-1)}$.

Similarly, we can foil all oracle machines which make at most $k(|x|)-3$ queries in the worst case, and even on the average. The proof for the oracle and the average oracle versions are more involved. □

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

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

(1) $(P \longleftrightarrow V)$ has knowledge complexity $\leq p(|x|)\cdot k(|x|)$ in the *average* Oracle sense.

(2) $(P \longleftrightarrow V)$ has knowledge complexity greater than $k(|x|)-2-\log\frac{1}{p(|x|)}$ in the Oracle sense.

Proof sketch: We consider an interactive proof in which, on input x, with probability $p(|x|)$ the prover sends $K(x)$ to the verifier (and otherwise sends nothing). Clearly, for $|K(x)| \leq k(|x|)$, this interactive proof has *average* knowledge complexity bounded by $p(|x|)\cdot k(|x|)$ in the Oracle sense. We now show that for an appropriate choice of K, the above interactive proof has knowledge complexity greater than $t(|x|) \stackrel{\Delta}{=} k(|x|)-2-\log\frac{1}{p(|x|)}$ in the Oracle sense. We set $K(x)=y$, where y is a $k(|x|)$-bit string satisfying $\text{Prob}(M^{\alpha}(x)=y) \leq 2^{-k(|x|)}$ (for a random oracle α). It follows that for every oracle set α, we have $\text{Prob}(M^{\alpha}(x)=y) \leq 2^{t(|x|)}\cdot 2^{-k(|x|)} = \frac{p(|x|)}{4}$ 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 $k(|x|)+1-log(\frac{1}{p(|x|)})$ even in the strict oracle measure.

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

Proposition 4.3: There exists an interactive proof Π that has knowledge complexity 1 in the strict Oracle sense, yet for any polynomial p, $kc_{Hint}(\Pi) > p(|x|)$.

Sketch of proof: The protocol is the following: V sends a random string in $\{0,1\}^{|x|}$ to P, which responds with a single bit $K(x,r)$. Let $n=|x|$. We build a function $K(x,r)$ such that any probabilistic polynomial time machine that uses a hint of length less than $t(n) \stackrel{\Delta}{=} \frac{1}{10}\cdot 2^n$ fails to simulate Π. 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 Π. Let us consider the distributions $M(x,h)$ for all possible h's ($h \in \{0,1\}^{t(n)}$). For each $h \in \{0,1\}^{t(n)}$, define a function $f^{(h)}: \{0,1\}^n \to \{0,1\}$ so that $f^{(h)}(r)=\sigma$ iff $\text{Prob}(M(x,h)=(r,\sigma)) \geq \text{Prob}(M(x,h)=(r,\bar{\sigma}))$. we select a function $K(x,r)$ to foil the simulation of M on x, such that $K(x,\cdot)$ is not close to $f^{(h)}$ for any $h \in \{0,1\}^{t(|x|)}$. In fact, counting arguments show that we can select $K(x,\cdot)$ that disagrees with each $f^{(h)}$ in at least $\frac{1}{6}$ of the possible r's. □

Proposition 3.8 states that for any interactive proof Π, and any unbounded $g: N \to N$, $kc^{Strict}_{Oracle}(\Pi) \le kc^{1/2}_{Oracle}(\Pi) + log(log(|x|)) + g(|x|)$. The following proposition shows that we can not do better in general.

Proposition 4.4: For every polynomial time computable function k: N→N, there exists an interactive proof Π such that: $kc^{1/2}_{Oracle}(\Pi) = k(|x|)+1$ and $kc^{Strict}_{Oracle}(\Pi) \ge k(|x|) + log(log(|x|)) + O(1)$.

Proof sketch: Suppose, first, that $k(|x|)=0$. Let $S_x \subseteq \{0,1\}^{|x|}$ be a set of cardinality $\frac{1}{2}\cdot 2^{|x|}$. Consider a protocol, Π, in which the prover P sends V a uniformly chosen $y \in S_x$. A simulator of the oracle type can choose a string uniformly in $\{0,1\}^{|x|}$, and ask the oracle whether $y \in S_x$. Thus, for any S_x, the simulator outputs a conversation with probability $\frac{1}{2}$. We use again the diagonalization technique, and show that for each strict oracle simulator M, there exists an S_x such that M can not simulate $\Pi(x)$ properly. To extend the proof for any polynomial computable function $k(|x|)$, we compose (sequentially) the above protocol with the protocol used in the proof of Proposition 4.1. □

Propositions 2.6 and 2.7 state the relations between the oracle version, and the measures we get by changing the constant ½ (the lower bound on the probability that the simulator produces an output) to any constant $0 < \delta < 1$. The following propositions show that we cannot get tighter relations in general.

Proposition 4.5: For any polynomial computable function k: N→N, and for each constant $0 < \delta < \frac{1}{2}$, there exists a protocol Π_δ such that $kc^{1/2}_{Oracle}(\Pi_\delta) = k(|x|)+1$ and $kc^{1-\delta}_{Oracle}(\Pi_\delta) \ge k(|x|) + log(log(\frac{1}{\delta})) - O(1)$.

Proof: The same construction as in the previous proof.

Proposition 4.6: For every interactive proof Π, and every $0 < \varepsilon < \frac{1}{2}$,

$kc^{\varepsilon}_{Oracle}(\Pi) \le 1 + \max\{kc^{1/2}_{Oracle}(\Pi) - \lfloor \log(\frac{1}{\varepsilon}) \rfloor, 0\}$.

Proof sketch: Guess a random string for the original simulator, and guess the first $\lfloor \log(\frac{1}{\varepsilon}) \rfloor$ answers of the oracle. Query the oracle whether this prefix of the oracle answers is correct for the chosen random string. If so, let the original simulator run (spearing the first $\lfloor \log(\frac{1}{\varepsilon}) \rfloor$ queries). Otherwise, stop with no output. □

5. Properties of knowledge complexity of languages in the Hint sense

In this section we investigate the "hint-knowledge complexity" hierarchy of languages and establish two results - $KC_{Hint}(poly(|x|)) \subseteq AM$ [2], and $KC_{Hint}(O(log(|x|)) \subseteq co\text{-}AM$ [2] . These results are obtained by extending the results proven for zero-knowledge by Aiello and Hastad [AH] and Fortnow [F], respectively. For more details see our technical report [GP].

Theorem 5.2: Let L be a language that has an interactive proof with knowledge complexity $k = poly(|x|)$ in the Hint sense, then $L \in AM$ [2].

Proof: omitted.

Theorem 5.3: Let L be a language that has an interactive proof with knowledge complexity $k = O(log(n))$ in the Hint sense, then the complement of L is in AM [2].

Proof: omitted.

6. The knowledge complexity of constant round AM proofs

The following theorem is an extension of a theorem appearing in [GK]. The model that Goldreich and Krawczyk considered, and that we use here too, is of black box simulation (see also [O,GO]).

Theorem 6.1: For all polynomial-computable function $k(|x|)$, only languages that can be recognized in probabilistic time $|x|^c \cdot 2^{20 \cdot k(|x|)}$ (for a constant $c>0$) have an Arthur Merlin protocol Π with the following properties:

(1) Π has a constant number of rounds.

(2) Π has knowledge complexity $k(|x|)$ using a black box argument in any of the measures considered in this paper (and even in the computational sense).

(3) The error probability of the AM protocol is less than $2^{-20 \cdot k(|x|) - |x|}$.

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:

[AH] Aiello, W., and J. Hastad, "Perfect Zero-Knowledge Languages can be Recognized in Two Rounds", *Proc. 28th FOCS*, pp. 439-448, 1987. To appear in *JCSS*.

[B] Babai, L., "Trading group theory for randomness", *Proc. 17th STOC*, 1985, pp. 421-429.

[BCK] Bar-Yehuda, R., B. Chor, and E. Kushilevitz, "Privacy, Additional Information, and Communication", *5th IEEE Structure in Complexity Theory*, July 1990, pp. 55-65.

[BGGHKMR] Ben-or, M., O. Goldreich, S. Goldwasser, J. Hastad, J. Killian, S. Micali, and P. Rogaway, "Everything Provable is Provable in Zero-Knowledge", *Advances in Cryptology - Crypto88 (proceedings)*, S. Goldwasser (ed.), Springer-Verlag, Lecture Notes in Computer Science, Vol. 403, pp. 37-56, 1990.

[CW] Carter, J., and M. wegman, "Universal Classes of Hash Functions", *JCSS*, 1979, Vol. 18, pp. 143-154.

[F] Fortnow, L., "The Complexity of Perfect Zero-Knowledge", *Proc. of 19th STOC*, pp. 204-209, 1987. *Advances in Computing Research (ed. S. Micali)*, 1989, Vol. 5, pp. 327-343.

[GK] Goldreich, O. and H. Krawczyk, "On the Composition of Zero-Knowledge Proof Systems", *17th ICALP* proceedings, Lecture Notes in Computer Science, Vol. 443, Springer Verlag, 1990, pp. 268-282.

[GMW] Goldreich, O., S. Micali, and A. Wigderson, "Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design", *Proc. 27th FOCS*, 1986, pp. 174-187. To appear in *Jour. of ACM*.

[GO] Goldreich, O. and Y. Oren, "Definitions and Properties of Zero- Knowledge Proof Systems", Technical Report #610, Computer Science Dept., Technion, Haifa, Israel.

[GP] Goldreich, O. and E. Petrank, "Quantifying Knowledge Complexity", Technical Report #683, Computer Science Dept., Technion, Haifa, Israel.

[GMR] Goldwasser, S., S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proofs", *Proc. 17th STOC*, 1985, pp. 291-304.

[GMR2] Goldwasser, S., S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof systems", *SIAM Jour. on computing*, Vol. 18, 1989, pp. 186-208.

[GS] Goldwasser, S., and M. Sipser, "Private Coins vs. Public Coins in Interactive Proof Systems", *Advances in Computing Research (ed. S. Micali)*, 1989, Vol. 5, pp. 73-90.

[IY] Impagliazzo, R., and M. Yung, "Direct Minimum-Knowledge Computations", *Advances in Cryptology - Crypto87 (proceedings)*, C. Pomerance (ed.), Springer-Verlag, Lectures Notes in Computer Science, vol. 293, 1987, pp. 40-51.

[O] Oren, Y., "On the Cunning Power of Cheating Verifiers: Some Observations About Zero-Knowledge Proofs", *Proc. of the 28th IEEE Symp. on Foundation of Computer Science*, 1987, pp. 462-471.

[S] Sipser, M., "A Complexity Approach to Randomness", *15th STOC* 1983, pp. 330-335.

Appendix: Inadequacies in the GMR definition.

In [GMR] the knowledge complexity of an interactive proof $(P \leftrightarrow V)$ is said to be $k(|x|)$ 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 $1-2^{-k(|x|)+|x|^{-c}}$ for all constants $c>0$ and sufficiently long x's. Recall that the variation difference of the random variables Y and Z is $\sum_{\alpha} |\text{Prob}(Y=\alpha) - \text{Prob}(Z=\alpha)|$. 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 $k(|x|) = \frac{\log(|x|)}{o(1)}$.