# Chicago Journal of Theoretical Computer Science

*The MIT Press*

Volume 1998, Article 1
*10 March 1998*

ISSN 1073–0486. MIT Press Journals, Five Cambridge Center, Cambridge, MA 02142-1493 USA; (617)253-2889; *journals-orders@mit.edu, journals-info@mit.edu*. Published one article at a time in LaTeX source form on the Internet. Pagination varies from copy to copy. For more information and other articles see:

- *http://mitpress.mit.edu/CJTCS/*
- *http://www.cs.uchicago.edu/publications/cjtcs/*
- *ftp://mitpress.mit.edu/pub/CJTCS*
- *ftp://cs.uchicago.edu/pub/publications/cjtcs*

The *Chicago Journal of Theoretical Computer Science* is abstracted or indexed in *Research Alert,®SciSearch,®Current Contents®/Engineering Computing & Technology*, and *CompuMath Citation Index.®*

©1998 The Massachusetts Institute of Technology. Subscribers are licensed to use journal articles in a variety of ways, limited only as required to insure fair attribution to authors and the journal, and to prohibit use in a competing commercial product. See the journal's World Wide Web site for further details. Address inquiries to the Subsidiary Rights Manager, MIT Press Journals; (617)253-2864; *journals-rights@mit.edu*.

The *Chicago Journal of Theoretical Computer Science* is a peer-reviewed scholarly journal in theoretical computer science. The journal is committed to providing a forum for significant results on theoretical aspects of all topics in computer science.

*Editor in chief:* Janos Simon

*Consulting editors:* Joseph Halpern, Stuart A. Kurtz, Raimund Seidel

*Editors:*

| | | |
|---|---|---|
| Martin Abadi | Greg Frederickson | John Mitchell |
| Pankaj Agarwal | Andrew Goldberg | Ketan Mulmuley |
| Eric Allender | Georg Gottlob | Gil Neiger |
| Tetsuo Asano | Vassos Hadzilacos | David Peleg |
| Laszló Babai | Juris Hartmanis | Andrew Pitts |
| Eric Bach | Maurice Herlihy | James Royer |
| Stephen Brookes | Ted Herman | Alan Selman |
| Jin-Yi Cai | Stephen Homer | Nir Shavit |
| Anne Condon | Neil Immerman | Eva Tardos |
| Cynthia Dwork | Howard Karloff | Sam Toueg |
| David Eppstein | Philip Klein | Moshe Vardi |
| Ronald Fagin | Phokion Kolaitis | Jennifer Welch |
| Lance Fortnow | Stephen Mahaney | Pierre Wolper |
| Steven Fortune | Michael Merritt | |

*Managing editor:* Michael J. O'Donnell

*Electronic mail: chicago-journal@cs.uchicago.edu*

# The Isomorphism Problem for Read-Once Branching Programs and Arithmetic Circuits

Thomas Thierauf

10 March, 1998

**Abstract**

*Abstract-1* We investigate the computational complexity of the isomorphism problem for read-once branching programs (1-BPI): upon input of two read-once branching programs $B_0$ and $B_1$, decide whether there exists a permutation of the variables of $B_1$ such that it becomes equivalent to $B_0$.

*Abstract-2* Our main result is that 1-BPI cannot be NP-hard unless the polynomial hierarchy collapses. The result is extended to the isomorphism problem for arithmetic circuits over large enough fields.

*Abstract-3* We use the known arithmetization of read-once branching programs and arithmetic circuits into multivariate polynomials over the rationals. Hence, another way of stating our result is: the isomorphism problem for multivariate polynomials over large enough fields is not NP-hard unless the polynomial hierarchy collapses.

*Abstract-4* We derive this result by providing a two-round interactive proof for the nonisomorphism problem for multivariate polynomials. The protocol is based on the Schwartz-Zippel theorem for probabilistically checking polynomial identities.

*Abstract-5* Finally, we show that there is a perfect zero-knowledge interactive proof for the isomorphism problem for multivariate polynomials.

## 1 Introduction

*1-1* An interesting computational issue is to decide the *equivalence* of two given programs with respect to some computational model such as Boolean circuits, branching programs, or Boolean formulas. A more general problem is to

decide the *isomorphism* of two given circuits (for example), that is, whether the circuits become equivalent after permuting the input variables of one of the circuits. Isomorphism is one way of formalizing the idea that two circuits are "almost" equivalent.

1-2 The isomorphism problem for Boolean circuits is in fact a very old one, dating back to the last century (see [BRS96] for background and early references on this problem and variants of it). More recently, the problem has been reconsidered with respect to its computational complexity (see, for example, [AT96, BR93, BRS96, CK91]).

1-3 The equivalence problems for Boolean circuits, branching programs, and Boolean formulas are known to be coNP-complete. Asking for isomorphism roughly amounts to putting an existential quantifier in front of the problem. Therefore, the corresponding isomorphism problems are in the second level of the polynomial hierarchy, $\Sigma_2^p$. An obvious question is whether these isomorphism problems are complete for $\Sigma_2^p$. This question was solved by Agrawal and Thierauf [AT96], who showed that none of these isomorphism problems is complete for $\Sigma_2^p$ unless the polynomial hierarchy collapses. Thus, loosely speaking, the existential quantifier we get by seeking an isomorphism does not seem to add full NP-power to the corresponding equivalence problem.

1-4 The most prominent example that supports the latter observation might be the *graph isomorphism problem*, GI. Here, the equivalence problem for graphs, which is in fact an equality problem, is trivially solvable in polynomial time; therefore, GI is in NP. However, GI is not NP-complete unless the polynomial hierarchy collapses [GMW91, BHZ87] (see also [Sch89]).

1-5 For a restricted class of branching programs, namely, the *read-once branching programs*, where on each path, each variable is tested at most once (see the next section for precise definitions), the equivalence problem is easier: it can be efficiently solved by a randomized Monte-Carlo type algorithm [BCW80] with one-sided error, and is in the complexity class coRP. Therefore, when an existential quantifier is put in front, the isomorphism problem for read-once branching programs, 1-BPI, is in NP·coRP. Motivated by the examples above, we ask whether 1-BPI is NP-hard.

1-6 In this paper, we show that 1-BPI is also in the Arthur-Merlin class, coAM = BP · coNP. As a consequence, it is not NP-hard unless the polynomial hierarchy collapses. The result is extended to *arithmetic circuits* (or *straight-line programs*) over large enough fields.

1-7 One crucial point required to prove these results is that both read-once branching programs and arithmetic circuits can be arithmetized, yielding

equivalent multivariate polynomials [BCW80, IM83]. In the case of read-once branching programs, these polynomials are multilinear. Therefore, our main results can be restated in purely algebraic terms: the isomorphism problem for multivariate polynomials over large enough fields is not NP-hard unless the polynomial hierarchy collapses.

1-8 Our proof is based on a two-round interactive proof for the nonisomorphism problem for multivariate polynomials. In principle, the interactive proof protocol follows the one for *graph nonisomorphism*, GNI. However, there is a crucial difference: in our protocol, the verifier needs to compute a normal form of a polynomial to hide the syntactic structure of the input polynomials that are manipulated. This problem does not occur when working with graphs; however, this requirement seems to exceed the computational power of the verifier. We get around this difficulty by computing a *randomized normal form* instead. The randomized normal form is based on the Schwartz-Zippel theorem for testing polynomial identities.

1-9 Combining these ideas with the ones from Goldreich, Micali, and Wigderson [GMW91], we obtain perfect zero-knowledge interactive proofs for these isomorphism problems.

1-10 An extension of an isomorphism is a *congruence*, where in addition to permuting variables, one can exchange a variable with its negation. Although we formulate our results only for isomorphism problems, it can easily be checked that all our theorems also hold for the corresponding congruence problems.

# 2 Preliminaries

## 2.1 Basic Definitions

2.1-1 We use fairly standard notions of complexity theory. We refer the reader to [BDG88, BDG91, HU79] for definitions of complexity classes such as P, NP, RP, or BPP. The $k^{\text{th}}$ level of the polynomial hierarchy is denoted by $\Sigma_k^p$. For any class $\mathcal{C}$, we denote the complement class by co $\mathcal{C}$.

2.1-2 Class BP $\cdot$ NP is the class of sets $L$ such that there exist a set $A \in$ NP and a polynomial $p$ such that for every $x$:

$$\begin{aligned} x \in L \quad &\Longrightarrow \quad \mathbf{Pr}[(x,y) \in A] \;=\; 1 \\ x \notin L \quad &\Longrightarrow \quad \mathbf{Pr}[(x,y) \in A] \;\leq\; \frac{1}{2} \end{aligned}$$

where $y$ is chosen uniformly at random from $\Sigma^{p(|x|)}$. The obvious definition of BP $\cdot$ NP would be with two-sided error, but this is equivalent with the definition given here.

2.1-3 Class NP$\cdot$coRP is the class of sets $L$ such that there exist a set $A \in$ coRP and a polynomial $p$ such that for every $x$, we have:

$$x \in L \iff \exists y \in \Sigma^{p(|x|)} : (x, y) \in A$$

2.1-4 Interactive proofs were defined in [GMR89]. An interactive proof system for a set $L$ consists of a prover $P$ and a verifier $V$. The verifier is a randomized polynomial-time algorithm that can communicate with the prover. The prover can make arbitrary computations. After following some communication protocol, the verifier finally has to accept or reject a given input, such that:

$$\begin{aligned} x \in L &\implies \exists \text{ prover } P : \mathbf{Pr}[(V,P)(x) \text{ accepts}] = 1 \\ x \notin L &\implies \forall \text{ prover } P : \mathbf{Pr}[(V,P)(x) \text{ accepts}] \leq \frac{1}{2} \end{aligned}$$

where the probability is taken over the random choices of the verifier.

2.1-5 Class IP is the class of sets that have an interactive proof system, and IP$[k]$ is the subclass of IP where the verifier and the prover exchange at most $k$ messages.

2.1-6 Arthur-Merlin games were introduced in [Bab85]. They are defined similarly to interactive proofs, with Arthur corresponding to the verifier and Merlin to the prover. The only difference is that Arthur has to make his random bits available to Merlin, whereas in the interactive proof model, the prover does not know the random bits of the verifier. The notation AM$[k]$ denotes when $k$ messages can be sent, and AM = AM[2].

2.1-7 In this paper, we are interested in constant-round interactive proof systems. It is known that for both interactive proof systems and Arthur-Merlin games, constantly many rounds are the same as one round: for any $k \geq 2$, IP$[k]$ = AM$[k]$ = AM [Bab85, GS89]. Moreover, it is known that AM = BP $\cdot$ NP.

2.1-8 An IP protocol for a set $L$ is a *perfect zero-knowledge* protocol [GMR89] if it decides $L$ correctly in the usual way and, in addition, for any $x \in L$, the prover does not reveal any extra information to any verifier $V^*$ besides the fact that $x$ is in $L$: the messages exchanged with the prover appear random to $V^*$ (that is, these messages could have been produced by $V^*$ himself).

2.1-9

*Definition 1-1* **Definition 1** *A prover $P$ is perfectly zero knowledge on $L$ if, for any interactive machine $V^*$ running in expected polynomial time, there is a probabilistic machine $M_{V^*}$ running in expected polynomial time such that for any $x \in L$ and any string $H$, $|H| \leq |x|^c$ for some $c > 0$, and the communication between $P$ and $V^*$ on input $(x, H)$, seen as a random variable, is identically distributed to the output of $M_{V^*}$ on the same input.*

*Definition 1-2* *Together, $P$ and $V$ form a perfect zero-knowledge proof system for $L$ if the system is an interactive proof system for $L$, and if $P$ is perfectly zero knowledge on $L$.*

*2.1-10* The string $H$ in the definition is needed if we wish to combine two zero-knowledge protocols into one zero-knowledge protocol: the history $H$ from the first protocol, which is known to the verifier in the second protocol, does not help the verifier. For a more detailed discussion of this definition and the need for expected polynomial time, see [GMR89, GMW91].

## 2.2 Verifying Polynomial Identities

*2.2-1* Let $\mathbf{F}$ be some field, and $p = p(x_1, \ldots, x_n)$ be a multivariate polynomial over $\mathbf{F}$. The *degree* of $p$ is the maximum exponent of any variable when $p$ is written as a sum of monomials. Polynomials of degree 1 are called *multilinear*. Note the difference from the *total degree* of a polynomial, where one first adds the exponents of the variables in each monomial and then takes the maximum over these sums.

*2.2-2* Given some polynomial $p$ written as an arithmetic expression, we want to find out whether $p$ is in fact the zero polynomial. The obvious algorithm, namely, to transform the arithmetic expression in a sum of monomials and check whether all coefficients are zero, can have up to exponential running time (in the size of the input). Efficient *probabilistic* zero tests were developed by Schwartz [Sch80] and Zippel [Zip79]. The version below is a variant shown by Ibarra and Moran [IM83]. They extended the corresponding theorem for multilinear polynomials shown by Blum, Chandra, and Wegman [BCW80] to arbitrary degrees. We give a proof for completeness.

**Theorem 1 ([IM83, Sch80, Zip79])** *Let $p(x_1, \ldots, x_n)$ be a multivariate polynomial of degree $d$ over field $\mathbf{F}$ that is not the zero polynomial. Let $T \subseteq \mathbf{F}$ with $|T| \geq d$. Then there are at least $(|T| - d)^n$ points $(a_1, \ldots, a_n) \in T^n$ such that $p(a_1, \ldots, a_n) \neq 0$.*

**Proof of Theorem 1** The proof is by induction on $n$. For $n = 1$ the theorem is true, because a degree-$d$ polynomial has at most $d$ roots in $\mathbf{F}$.

2.2-3 Let $n > 1$ and let $p(x_1, \ldots, x_n)$ be a nonzero polynomial of degree $d$. Furthermore, let $\boldsymbol{a} = (a_1, \ldots, a_n)$ be a point such that $p(\boldsymbol{a}) \neq 0$. We define two polynomials, both of which are subfunctions of $p$:

$$\begin{aligned} p_0(x_1, \ldots, x_{n-1}) &= p(x_1, \ldots, x_{n-1}, a_n) \\ p_1(x_n) &= p(a_1, \ldots, a_{n-1}, x_n) \end{aligned}$$

By construction, both polynomials are nonzero and have degree bounded by $d$: $p_0$ has $n-1$ variables, and therefore differs from 0 on at least $(|T|-d)^{n-1}$ points in $T^n$ (by the induction hypothesis). Similarly, $p_1$ has one variable, and therefore has at least $|T| - d$ nonzero points.

2.2-4 For each of the $|T|-d$ choices for $a_n$ where $p_1$ is nonzero, the corresponding polynomial $p_0$ has $(|T| - d)^{n-1}$ nonzero points. Therefore, the number of nonzero points of $p$ in $T^n$ is at least $(|T| - d)(|T| - d)^{n-1} = (|T| - d)^n$.

**Proof of Theorem 1** □

2.2-5 We mention two important consequences of this theorem. First of all, let $T$ be any subset of $\mathbf{F}$ that has at least $d + 1$ elements. Then any nonzero polynomial of degree $d$ has a nonzero point in $T^n$.

**Corollary 1** *Let $p(x_1, \ldots, x_n)$ be a polynomial of degree $d$ over $\mathbf{F}$, and $T \subseteq \mathbf{F}$ with $|T| > d$. Then $p \not\equiv 0 \Longleftrightarrow \exists \boldsymbol{a} \in T^n \;\; p(\boldsymbol{a}) \neq 0$.*

2.2-6 By enlarging $T$ even further, we can show that any nonzero polynomial $p$ does not vanish on most of the points of $T^n$. This provides the tool for the probabilistic zero test.

**Corollary 2** *Let $p(x_1, \ldots, x_n)$ be a polynomial of degree $d$ over $\mathbf{F}$, and $T \subseteq \mathbf{F}$ with $|T| \geq 2nd$. Let $\boldsymbol{r} = (r_1, \ldots, r_n)$ be a random element from $T^n$. Then $p(\boldsymbol{r}) \neq 0$, with probability at least $1/2$.*

**Proof of Corollary 2**

$$\mathbf{Pr}[p(\boldsymbol{r}) \neq 0] \geq \left(\frac{|T| - d}{|T|}\right)^n \geq \left(1 - \frac{1}{2n}\right)^n \geq \frac{1}{2}$$

**Proof of Corollary 2** □

## 2.3 Branching Programs

2.3-1 A *branching program* $B$ in $n$ Boolean variables $x_1, \ldots, x_n$ is a directed acyclic graph with the following types of nodes. There is a single node of indegree zero, the *initial node of* $B$. All nodes have outdegree two or zero. A node with outdegree two is an *internal node of* $B$. One of its edges is labeled with $x_i$, the other with $\overline{x}_i$, for some $i \in \{1, \ldots, n\}$. A node with outdegree zero is a *final node of* $B$. The final nodes are labeled by either *accept* or *reject*.

2.3-2 We call a branching program *read-once* if, on each path from the initial node to a final node, every variable or its complement occurs at most once as an edge label.

2.3-3 A read-once branching program is called *ordered* if the order of variable occurence on each path is consistent with some ordering on the set of variables.

2.3-4 Branching programs are also called *binary decision diagrams* (BDD) or *Boolean graphs*. Read-once branching programs and ordered branching programs are also called *free binary decision diagrams* (FBDD) or *free Boolean graphs* and *ordered binary decision diagrams* (OBDD), respectively.

2.3-5 A branching program $B$ defines an $n$-ary Boolean function from $\{0,1\}^n$ to $\{0,1\}$ as follows. For an assignment $\boldsymbol{a} = (a_1, \ldots, a_n) \in \{0,1\}^n$, we walk through $B$, starting at the initial node, always following the (unique) edge that evaluates to one under $\boldsymbol{a}$, until we reach a final node. If the final node is an accepting node, we define $B(\boldsymbol{a}) = 1$, and $B(\boldsymbol{a}) = 0$ otherwise.

2.3-6 Two branching programs $B$ and $B'$ in $n$ variables are equivalent, $B \equiv B'$ for short, if they define the same Boolean function. By BPE we denote the problem of deciding whether two given branching programs are equivalent. That is,

$$\text{BPE} = \{\, (B, B') \mid B \equiv B' \,\}$$

It is not hard to see that branching programs can compute CNF Boolean formulas. Therefore, the satisfiability problem for branching programs is NP-complete; hence BPE is coNP-complete.

2.3-7 Blum, Chandra, and Wegman [BCW80] showed that the equivalence problem for read-once branching programs, 1-BPE, is easier because one can transform these programs into equivalent multilinear polynomials over the rational numbers, $\mathbf{Q}$. To see this, note first that a branching program $B$ can be viewed as a compact way of denoting a DNF formula $F_B$: each path of $B$ can be written as a monomial, the conjunction of the literals occurring

along that path. Then, the function computed by $B$ is simply the disjunction of all monomials coming from accepting paths of $B$.

2.3-8 We convert $F_B$ into a polynomial $p_B$ over the rational numbers $\mathbf{Q}$ as follows. A variable $x_i$ is kept as $x_i$. A negated variable $\overline{x}_i$ is replaced by $1 - x_i$. A conjunction is replaced by multiplication, and a disjunction is replaced by addition. For each satisfying assignment $\boldsymbol{a} \in \{0,1\}^n$, exactly one path of $B$ evaluates to true under $\boldsymbol{a}$. Therefore, at most one product term in $p_B$ will be one on input $\boldsymbol{a}$; hence, $B$ and $p_B$ agree on $\{0,1\}^n$. That is,

$$B(\boldsymbol{a}) = p_B(\boldsymbol{a}) \quad \text{for all } \boldsymbol{a} \in \{0,1\}^n$$

It is easy to get an arithmetic expression for $p_B$ from $B$ that has about the same size as $B$. Note, however, that when written as a sum of monomials, $p_B$ may consist of exponentially many terms in the size of $B$. Therefore, in general, we cannot write down $p_B$ in this normal form in polynomial time in $|B|$. However, with the expression at hand, we can evaluate $p_B$ at a given point in $\mathbf{Q}^n$ in polynomial time, and this suffices for our purposes. To evaluate $p_B$ at a point $\boldsymbol{a} = (a_1, \ldots, a_n)$, we start by writing a 1 at the initial node. Suppose now that a node has value $v$, and its edges are labeled by variable $x_i$. Then, values $va_i$ and $v(1-a_i)$ are sent along the $x_i$- and $\overline{x}_i$-edges, respectively. When all incoming edges of a node $u$ have sent values, the value of $u$ is the sum of all these incoming values. Finally, the value of $p_B$ appears at the accepting node.

2.3-9 Since $B$ is read-once, $p_B$ is a multilinear polynomial. Next let $B'$ be another read-once branching program, and let $p_{B'}$ be the corresponding polynomial. If $B$ and $B'$ are equivalent, then $p_B$ and $p_{B'}$ agree on $\{0,1\}^n$, a two-element set. By Corollary 1 (applied to $p = p_B - p_{B'}$), it follows that $p_B$ and $p_{B'}$ agree on $\mathbf{Q}$. Choosing $T$ in Corollary 2 as $T = \{1, \ldots, 2n\}$, we detect an inequivalence with probability more than $1/2$. It follows that 1-BPE $\in$ coRP.

2.3-10 Fortune, Hopcroft, and Schmidt [FHS78] have shown that if one of two given read-once branching programs is even ordered, then the equivalence can be decided in polynomial time. In particular, the equivalence problem for ordered branching programs is solvable in polynomial time.

2.3-11 Two branching programs $B$ and $B'$ are isomorphic, denoted by $B \cong B'$, if there exists a permutation $\varphi$ on $\{x_1, \ldots, x_n\}$ such that $B$ becomes equivalent to $B'$ when permuting the variables of $B'$ according to $\varphi$. That is, $B \equiv B' \circ \varphi$. In this case, we call $\varphi$ an *isomorphism between $B$ and $B'$*.

2.3-12 The isomorphism problem for branching programs is:

$$\text{BPI} \;=\; \{\, (B, B') \mid B \cong B' \,\}$$

The isomorphism problem for read-once branching programs, 1-BPI, is defined analogously. It follows directly from the definition that BPI $\in \Sigma_2^p$, the second level of the polynomial hierarchy. Agrawal and Thierauf [AT96] showed that BPNI is in BP $\cdot \Sigma_2^p$. By a result of Schöning [Sch89], it follows that BPI cannot be complete for $\Sigma_2^p$ unless the polynomial hierarchy collapses to its third level, $\Sigma_3^p$.

2.3-13 For read-once branching programs, we have 1-BPI $\in$ NP $\cdot$ coRP. An obvious question is whether 1-BPI is NP-hard. In this paper, we show that the problem of deciding whether two read-once branching programs are not isomorphic, 1-BPNI, is in BP $\cdot$ NP. Combined with the result of Boppana, Håstad, and Zachos [BHZ87] (see also Schöning [Sch89]), it follows that 1-BPI cannot be NP-hard unless the polynomial hierarchy collapses to its second level, $\Sigma_2^p$.

2.3-14 This result also covers the case of ordered branching programs. Note, however, that in that case, the isomorphism problem is in NP.

## 2.4 Arithmetic Circuits

2.4-1 An *arithmetic circuit over a field* **F** is a circuit where the inputs are field elements, and the (fan-in two) gates perform the field operations $+$, $-$, and $\times$. (We could also allow division, as long as a circuit guarantees not to divide by zero on any input.) Circuit size and depth are defined as usual.

2.4-2 Ibarra and Moran [IM83] considered the equivalence problem for arithmetic circuits (called straight-line programs there). They gave probabilistic polynomial-time algorithms for circuits over infinite fields. This was split into two cases, depending on whether the field had characteristic of 0 or greater than 0. If the field **F** had characteristic 0, it contained a subfield isomorphic to **Q**, the rational numbers. Therefore, it is enough to consider $\mathbf{F} = \mathbf{Q}$. We show how a zero test can be done in this case.

2.4-3 If a circuit $C$ has $n$ input variables $x_1, \ldots, x_n$, then $C$ computes a multivariate polynomial $p_C$ over **Q**. If $C$ has depth $d$, then $p_C$ has degree at most $2^d$. Therefore, to obtain a zero test for $p_C$, we have to choose $T$ in Corollary 2 as $T = \{1, \ldots, 2n2^d\}$ to detect a nonzero point with probability more than $1/2$ at a random point from $T^n$.

2.4-4 However, we do not have a polynomial-time procedure yet, because the function values of $p_C$ on $T^n$ could be as large as $(2n2^d)^{n2^d} \leq 2^{N2^N}$ for $N = nd$. Represented in binary, such numbers would be exponentially long. Instead, we evaluate $p_C$ modulo smaller numbers, namely from $M = \{1, \ldots, 2^{2N}\}$. (For a zero test, we can assume that all coefficients are integers, so that the function values of $p_C$ are integers too.) Notice that $p_C \pmod m$ might have more zeros than $p_C$, however, not too many:

**Lemma 1 ([IM83])** *For any $y \leq 2^{N2^N}$ and a randomly chosen $m \in M$, we have $y \not\equiv 0 \pmod m$ with probability at least $\frac{1}{3N}$, for large enough $N$.*

**Proof of Lemma 1** Any $y \leq 2^{N2^N}$ has at most $N2^N$ prime divisors. By the prime-number theorem, there are more than $\frac{2^{2N}}{2N}$ primes in $M$ for large enough $N$. Therefore, $M$ contains at least $\frac{2^{2N}}{2N} - N2^N$ primes that do not divide $y$. Hence, for $m$ randomly chosen from $M$, we have:

$$\mathbf{Pr}[y \not\equiv 0 \pmod m] \geq \frac{\frac{2^{2N}}{2N} - N2^N}{2^{2N}} = \frac{1}{2N} - \frac{N}{2^N} \geq \frac{1}{3N}$$

**Proof of Lemma 1** □

2.4-5 The probabilistic zero test now works as follows.

**Corollary 3** *Let $p(x_1, \ldots, x_n)$ be a nonzero polynomial of degree $2^d$ over $\mathbf{Q}$, $T = \{1, \ldots, 2n2^d\}$, and $M = \{1, \ldots, 2^{2N}\}$, where $N = nd$. Choose $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{6N}$ from $T^n$ and $m_1, \ldots, m_{6N}$ from $M$, independently at random. Then, $p(\boldsymbol{r}_i) \not\equiv 0 \pmod{m_i}$, for some $i$, with probability at least $1/2$.*

**Proof of Corollary 3** By Corollary 2 and Lemma 1:

$$\mathbf{Pr}[p(\boldsymbol{r}_i) \not\equiv 0 \pmod{m_i}] \geq \frac{1}{2}\frac{1}{3N}, \text{ for any pair } \boldsymbol{r}_i, m_i$$

Therefore,

$$\mathbf{Pr}[p(\boldsymbol{r}_i) \equiv 0 \pmod{m_i} \text{ for all } i\ ] \leq \left(1 - \frac{1}{6N}\right)^{6N} \leq \frac{1}{2}$$

**Proof of Corollary 3** □

2.4-6 We only sketch briefly the case of infinite fields with finite characteristic, and refer the reader to [IM83] for a more detailed treatment. Let $\mathbf{F}$ be a field with characteristic $q$ (which must, therefore, be a prime number). The trick is then to switch from $\mathbf{F}$ to the ring of polynomials over $\mathrm{GF}(q)$. This is certified by the the following lemma.

**Lemma 2 ([IM83])** *Let $p(x_1, \ldots, x_n)$ be a polynomial. $p \equiv 0$ over $\mathbf{F}$ if and only if $p \equiv 0$ over $\mathrm{GF}(q)[x]$.*

2.4-7 Since $q$ is prime, $\mathrm{GF}(q)$ is a field; therefore, the ring $\mathrm{GF}(q)[x]$ is a principal ideal domain, that is, a ring with a one and no zero divisors such that every ideal is principal. (In fact, $\mathrm{GF}(q)[x]$ is what is sometimes called a *Euclidean ring*.) One can easily verify that this already suffices in the assumption of Theorem 1 and its corollaries, instead of having a field. Hence, we can apply the zero test for $p$ over the polynomial ring $\mathrm{GF}(q)[x]$.

2.4-8 However, we can only deal with polynomials up to polynomial size in the input length. In the case of $\mathbf{Q}$, we did computations modulo small enough prime numbers. Now we do computations modulo polynomials of small degree. There is an analog of Lemma 1 bounding the probability that $p(a_1, \ldots a_n) \neq 0$, but $p(a_1, \ldots a_n) = 0 \pmod{r}$ for a randomly chosen polynomial $r \in \mathrm{GF}(q)[x]$ of small degree and $a_i \in \mathrm{GF}(q)[x]$, for $i = 1, \ldots, n$.

2.4-9 Putting things together, we get a zero test analogous to the one for polynomials over $\mathbf{Q}$, only the domain has changed to a polynomial ring instead of numbers.

2.4-10 Clearly, for any polynomial $p$ in $\mathbf{F}[x_1, \ldots, x_n]$ given as an arithmetic expression, one can construct an arithmetic circuit computing $p$ that has about the same size as $p$. In particular, it follows from the discussion in the preceding section that one can transform a read-once branching program into an equivalent arithmetic circuit of about the same size. Though arithmetic circuits are the more general concept, we prove our main result for read-once branching programs first, and then explain how to extend it to solve the isomorphism problem for arithmetic circuits.

# 3 An Interactive Proof for 1-BPNI

3-1 We show that there is a two-round interactive proof for the nonisomorphism problem for read-once branching programs, 1-BPNI.

3-2 We start by recalling the idea of the interactive proof for the graph non-isomorphism problem, GNI [GMR89] (see also [Sch88]). There, on input of two graphs $G_0$ and $G_1$, the verifier randomly picks $i \in \{0,1\}$ and a permutation $\varphi$, and sends $H = \varphi(G_i)$ to the prover. The prover is then asked to find out what the value of $i$ is. The verifier will accept only if the prover gives the right answer.

3-3 When the input graphs are not isomorphic, then the prover can identify $i$ easily. However, when the graphs are isomorphic, both could have been used by the verifier to compute $H$, so that no prover can identify $i$. For this reason, the answer of any prover is correct with probability at most $1/2$.

3-4 First of all, note that we cannot directly adapt this protocol to branching programs. The reason is that the syntactic structure of two given isomorphic branching programs might tell the prover which of two given branching programs was selected by the verifier, at least, if the verifier simply exchanges variables according to some permutation.

3-5 A way out of this would be a normal form for read-once branching programs that is easy to compute; however, such a normal form is not known. At this point, in the case of general branching programs, Agrawal and Thierauf [AT96] used a result from learning theory by Bshouty et al. [BCG$^+$96]: there is a randomized algorithm that uses an NP-oracle and outputs branching programs equivalent to a given one. The important point is that although the algorithm might output (syntactically) different branching programs depending on its random choices, the output does not depend on the syntactic structure of its input. However, in our case, the verifier does not have an NP-oracle available, and there is no analog learning result for read-once branching programs without an NP-oracle.

3-6 The idea for getting around this problem is as follows. On input of two given read-once branching programs $B_0$ and $B_1$ with $n$ variables, the verifier randomly chooses one of them and permutes it with a random permutation to obtain a branching program $B$. Instead of trying to manipulate all of $B$, the verifier takes the arithmetization $p_B$ of $B$ and evaluates $p_B$ at a randomly chosen point $\boldsymbol{r} \in T^n$, where $T$ is some appropriate test domain. The prover is then asked to tell which of $B_0$, $B_1$ was used to obtain the point $(\boldsymbol{r}, p_B(\boldsymbol{r}))$. If $B_0$ and $B_1$ are isomorphic, then the prover cannot detect this, and must guess; it will fail with probability $1/2$. On the other hand, if $B_0$ and $B_1$ are not isomorphic, then the prover has a good chance of detecting the origin of $(\boldsymbol{r}, p_B(\boldsymbol{r}))$. This is because, by Corollary 2, different multilinear polynomials can agree on $T$ on at most $1/2$ of the points for $|T| \geq 2n$. That is, the

origin of $(\boldsymbol{r}, p_B(\boldsymbol{r}))$ is unique with high probability. With an additional round of communication, the prover can always convince the verifier of the nonisomorphism of $B_0$ and $B_1$. We give the details below.

**Theorem 2** 1-BPNI $\in$ IP[4].

**Proof of Theorem 2** We give a protocol for 1-BPNI. The inputs are two read-once branching programs $B_0$, $B_1$, both in $n$ variables. Let $T = \{1, \ldots, 2n\}$.

**V1:** The verifier randomly picks $i \in \{0,1\}$, a permutation $\varphi$, and $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k \in T^n$, where $k = \lceil n \log n \rceil + 2$. Then it permutes the variables of $B_i$ according to $\varphi$, computes $y_l = p_{B_i} \circ \varphi(\boldsymbol{r}_l)$ for $l = 1, \ldots, k$, and sends the set of pairs $R = \{\, (\boldsymbol{r}_l, y_l) \mid l = 1, \ldots, k \,\}$ to the prover.

**P1:** The prover sends $j \in \{0,1\}$ and a permutation $\varphi'$ to the verifier.

**V2:** If $i = j$, then the verifier accepts. If $i \neq j$, the verifier checks whether $p_{B_j} \circ \varphi'$ matches the set $R$, that is, whether $p_{B_j} \circ \varphi'(\boldsymbol{r}_l) = y_l$ for $l = 1, \ldots, k$. If the test fails, the verifier rejects; otherwise, it sends $\varphi$ to the prover.

**P2:** The prover sends a point $\boldsymbol{r}' \in T^n$ to the verifier.

**V3:** Finally, the verifier accepts if and only if $p_{B_i} \circ \varphi(\boldsymbol{r}') \neq p_{B_j} \circ \varphi'(\boldsymbol{r}')$.

3-7 We show that the above protocol works correctly.

**Case 1** $B_0 \ncong B_1$*: There is a prover such that the verifier always accepts*

The prover can cycle through all permutations and check whether $p_{B_0}$ or $p_{B_1}$ match with the set $R$ sent by the verifier in step V1. Say that polynomial $p_{B_0} \circ \varphi'$ matches. The prover then sends $j = 0$ and $\varphi'$ to the verifier in step P1.

3-8 If no permutation of polynomial $p_{B_1}$ matches $R$ as well, then $i$ must have been 0, and therefore, the verifier will accept in the first round.

3-9 On the other hand, if some permutation of polynomial $p_{B_1}$ matches $R$, then the prover cannot tell which one was used by the verifier. If the prover is lucky, $i$ has been zero anyway, and the verifier accepts. On the other hand, if $i \neq j$, then the verifier will send $\varphi$ to the prover in step V2, because $p_{B_j} \circ \varphi'$ matches $R$. Since $p_{B_j} \circ \varphi' \neq p_{B_i} \circ \varphi$, these polynomials can agree on at most $1/2$ of the points of $T^n$ by Corollary 2. Therefore, the prover can find a point

$\boldsymbol{r}' \in T^n$ such that $p_{B_j} \circ \varphi'(\boldsymbol{r}') \neq p_{B_i} \circ \varphi(\boldsymbol{r}')$, and send it to the verifier in step P2, which will accept in step V3. In summary, the verifier accepts with probability one.

**Case 2** $B_0 \cong B_1$*: For any prover, the verifier accepts with probability at most* $3/4$

By executing the protocol several times in parallel, the acceptance probability can be made exponentially small.

3-10 The prover will always find permutations of $p_{B_0}$ *and* $p_{B_1}$ that match the set $R$ sent by the verifier in step V1. Therefore, with respect to the test $i = j$ made by the verifier, the best the prover can do is guess $j$ randomly. This will make the verifier accept with probability $1/2$ in step V2. However, the prover can improve its chances by the condition checked in the second round by the verifier, i.e., by fixing $i$ and $\varphi$ chosen by the verifier, say $i = 0$. Then there might exist a permutation $\varphi'$ such that $p_{B_1} \circ \varphi'$ matches $R$, but $p_{B_0} \circ \varphi \neq p_{B_1} \circ \varphi'$. Then the prover could choose a point $\boldsymbol{r}'$ such that $p_{B_0} \circ \varphi(\boldsymbol{r}') \neq p_{B_1} \circ \varphi'(\boldsymbol{r}')$, and make the verifier accept in step V3 by sending $j = 1$, $\varphi'$, and $\boldsymbol{r}'$. We give an upper bound on the probability of this event.

3-11 By Corollary 2, for any $\varphi'$ such that $p_{B_0} \circ \varphi \neq p_{B_1} \circ \varphi'$, we have:

$$\mathbf{Pr}[p_{B_0} \circ \varphi(\boldsymbol{r}) = p_{B_1} \circ \varphi'(\boldsymbol{r})] \;<\; \frac{1}{2}$$

for a randomly chosen $\boldsymbol{r} \in T^n$. Because points $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k \in T^n$ are chosen independently and uniformly at random from $T^n$, we have:

$$\mathbf{Pr}[p_{B_1} \circ \varphi' \text{ matches } R] \;<\; 2^{-k}$$

Therefore, considering all such $\varphi'$, we get that:

$$\mathbf{Pr}[\exists \varphi' \; (p_{B_0} \circ \varphi \neq p_{B_1} \circ \varphi' \text{ and } p_{B_1} \circ \varphi' \text{ matches } R] \;<\; n!\, 2^{-k} \;\leq\; \frac{1}{4}$$

by our choice of $k$. We conclude that the probability that any of the conditions tested by the verifier is satisfied is bounded by $(1/2) + (1/4) = 3/4$. That is, the verifier accepts with probability at most $3/4$, irrespective of the prover.

**Proof of Theorem 2** □

3-12 We can directly reduce to a one-round interactive proof by choosing $T$ large, for example $T = \{1, \ldots, 2nn!\}$. Then, in case $B_0 \ncong B_1$, the prover can always find a point $\boldsymbol{r}'$ (as above) without knowing $\varphi$, and hence can already send it in the first round to the verifier, which can then make all its tests. However, another difficulty arises: when $T$ has exponential size, the values of the polynomials might be up to double exponential, in which case the polynomial time verifier can no longer deal with the numbers. We will show in the next section how the verifier can still manage its task.

3-13 As mentioned in Section 2.1, the class of sets that can be decided by a constant-round interactive proof system coincides with the Arthur-Merlin class AM which, in turn, is the same as $\mathrm{BP} \cdot \mathrm{NP}$ [Bab85, GS89].

**Corollary 4** 1-BPNI $\in \mathrm{BP} \cdot \mathrm{NP}$.

3-14 Schöning [Sch88] gives a direct proof that the graph isomorphism problem is in AM (i.e., without using the relationship between IP and AM) by using the Sipser hashing technique [Sip83]. We remark that we can modify Schöning's proof based on our technique, and also directly obtain Corollary 4.

3-15 Note that both classes, $\mathrm{BP} \cdot \mathrm{NP}$ and $\mathrm{NP} \cdot \mathrm{coRP}$, can very loosely be considered as some slight extensions of NP. In this sense, we have shown that 1-BPI is in a slight extension of $\mathrm{NP} \cap \mathrm{coNP}$.

**Corollary 5** 1-BPI $\in \mathrm{NP} \cdot \mathrm{coRP} \ \cap \ \mathrm{BP} \cdot \mathrm{coNP}$.

3-16 Boppana, Håstad, and Zachos [BHZ87] (see also Schöning [Sch89]) have shown that a coNP-complete set cannot be in $\mathrm{BP} \cdot \mathrm{NP}$ unless the polynomial hierarchy collapses to the second level, in fact, even to $\mathrm{BP} \cdot \mathrm{NP}$. Hence, we get the main result of this section.

**Corollary 6** 1-BPI *is not* NP*-hard, unless* $\mathrm{PH} = \Sigma_2^p$.

3-17 Because ordered branching programs are a restricted form of read-once branching programs, Corollary 6 can be applied. The equivalence problem for ordered branching programs is in P [FHS78]; therefore, the isomorphism problem for ordered branching programs is in NP.

**Corollary 7** *The isomorphism problem for ordered branching programs is not* NP*-complete unless* $\mathrm{PH} = \Sigma_2^p$.

3-18 Because computational models such as branching programs work over inputs from $\{0,1\}$, a set of size two, Corollary 1 restricts our techniques to multilinear polynomials. However, if we start with polynomials of degree $d$ over $\mathbf{Q}$ for some constant $d > 0$, then we can apply the above protocol for testing the nonisomorphism of two such polynomials. Just take the test domain $T$ of size $2dn$ for polynomials with $n$ variables. For the representation of the polynomials it is enough that we can evaluate them efficiently at any point. Therefore, the nonisomorphism problem for polynomials over $\mathbf{Q}$ is in $\mathrm{BP} \cdot \mathrm{NP}$.

**Corollary 8** *The isomorphism problem for polynomials of degree $d$ over $\mathbf{Q}$ is not* NP*-hard unless* $\mathrm{PH} = \Sigma_2^p$.

3-19 Our interactive proof system for 1-BPNI was motivated by the one for GNI. However, it is more general now, in the sense that it can be used to solve GNI as a special case. Specifically, one can assign a polynomial to a graph such that nonisomorphic graphs are mapped to nonisomorphic polynomials: e.g., let $G = (V, E)$ be a graph, where $V = \{1, \ldots, n\}$. We take one variable $x_i$ for each node $i \in V$. Define

$$\begin{aligned} e_i(x_1, \ldots, x_n) &= x_i^2 \prod_{(i,j) \in E} x_j, \text{ and} \\ p_G(x_1, \ldots, x_n) &= \sum_{i=1}^{n} e_i(x_1, \ldots, x_n) \end{aligned}$$

3-20 For graphs $G_0, G_1$ we have that $G_0 \cong G_1 \Longleftrightarrow p_{G_0} \cong p_{G_1}$. Therefore, we again obtain the result about GI from Corollary 8.

# 4 Extension to Arithmetic Circuits

4-1 In this section, we extend the above protocol for branching programs to an interactive proof to decide the nonisomorphism of two arithmetic circuits over a large enough field, $\mathbf{F}$. We start with $\mathbf{F} = \mathbf{Q}$ for an infinite field of characteristic 0.

4-2 Let $C_0$, $C_1$ be two arithmetic circuits with $n$ inputs that are of depth $d$. We take the protocol from the previous section and modify it according to Corollary 3. Let $T = \{1, \ldots, 2n2^d\}$ and $M = \{1, \ldots, 2^{2N}\}$, where $N = nd$.

**V1:** The verifier starts by randomly choosing $i \in \{0,1\}$ and a permutation $\varphi$ as before, and now $6Nk$ points $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{6Nk} \in T^n$, where $k = \lceil n \log n \rceil + 2$, and for each point $\boldsymbol{r}_l$, a number $m_l \in M$. Then the verifier computes $y_l = p_{C_i} \circ \varphi(\boldsymbol{r}_l) \bmod m_l$, for $l = 1, \ldots, 6Nk$, and sends the set of triples $R = \{\, (\boldsymbol{r}_l, y_l, m_l) \mid l = 1, \ldots, 6Nk \,\}$ to the prover.

**P1:** The prover sends $j \in \{0,1\}$ and a permutation $\varphi'$ to the verifier.

**V2:** If $i = j$, then the verifier accepts. If $i \neq j$, the verifier checks whether $p_{C_j} \circ \varphi'$ matches the set $R$, that is, whether $y_l = p_{C_i} \circ \varphi'(\boldsymbol{r}_l) \bmod m_l$, for $l = 1, \ldots, 6Nk$. If the test fails, the verifier rejects; otherwise, it sends $\varphi$ to the prover.

**P2:** The prover sends a point $\boldsymbol{r}' \in T^n$ and $m' \in M$ to the verifier.

**V3:** Finally, the verifier accepts if and only if $p_{C_i} \circ \varphi(\boldsymbol{r}') \not\equiv p_{C_j} \circ \varphi'(\boldsymbol{r}') \pmod{m'}$.

4-3 Combining the argument in the previous section with Corollary 3, the verifier will accept two isomorphic arithmetic circuits with probability at most 3/4. Note that the prover in step P2 must prove to the verifier that two numbers differ; therefore, the computation modulo some number does not work in favor of the prover in that case. Two nonisomorphic arithmetic circuits are still accepted with probability one.

4-4 The case of infinite fields of characteristic greater than 0 is analogous. As briefly explained in Section 2.4, the test domain becomes the polynomial ring $\mathrm{GF}(q)[x]$ when the field has characteristic $q$. Computations are done modulo randomly chosen polynomials of small degree, and can therefore be done in polynomial time.

4-5

**Theorem 3** *The nonisomorphism problem for arithmetic circuits over infinite fields is in* $\mathrm{BP} \cdot \mathrm{NP}$.

4-6 If the arithmetic circuits are over a finite field, say $\mathrm{GF}(q)$, where $q$ is some prime power, we run into the problem that there might not be enough elements for our set $T$ to make the above protocol work. Instead of $\mathrm{GF}(q)$, we take the extension field $\mathrm{GF}(q^t)$, where $t$ is the smallest integer such that $q^t \geq 2n2^d$, so that $t = \lceil \log_q 2n2^d \rceil$. Then we can set $T = \mathrm{GF}(q^t)$. By

Corollary 1, when $q > 2^d$, we have that two polynomials over GF($q$) are equivalent if and only if they are equivalent over any extension field.

*4-7* The verifier must be able to evaluate a polynomial at a given point in the extension field. For this, it needs an irreducible polynomial $\phi(x) \in \mathrm{GF}(q)[x]$ of degree $t$. The verifier can cycle through all the $q^{t+1} \leq 2n2^d q^2 < 2nq^3$ polynomials in GF($q$)[$x$] of degree $t$, and check irreducibility in polynomial time using the Berlekamp algorithm (see [Ber68], chapter 6), and the verifier will find an irreducible polynomial $\phi(x)$ in polynomial time. Then GF($q^t$) is isomorphic to $\mathrm{GF}(q)[x]/\phi(x)$. Therefore, knowing $\phi(x)$, the verifier can do all computation needed in polynomial time, and the protocol can proceed as in the case of branching programs.

**Theorem 4** *The nonisomorphism problem for arithmetic circuits of depth $d$ over a finite field of size more than $2^d$ is in* BP $\cdot$ NP.

*4-8* The lower bound on the field size is crucial: for small fields, the equivalence problem for arithmetic circuits is coNP-complete [IM83].

# 5 Perfect Zero-Knowledge Interactive Proofs

*5-1* Goldreich, Micali, and Wigderson [GMW91] show that there are perfect zero-knowledge interactive proofs for the graph isomorphism problem, GI, and its complement, GNI. Adapting their ideas, we show the existence of a perfect zero-knowledge interactive proof for the isomorphism of branching programs or arithmetic circuits. Fortnow [For89] and Aiello and Håstad [AH91] have shown that any set that has a perfect zero-knowledge interactive proof is in AM $\cap$ coAM. Thus it follows again from the result in this section that 1-BPI $\in$ coAM.

**Theorem 5** *There is a perfect zero-knowledge interactive proof system for* 1-BPI.

*Proof of Theorem 5-1* **Proof of Theorem 5** The IP protocol described below accepts 1-BPI and has the perfect zero-knowledge property. The inputs are two read-once branching programs $B_0$ and $B_1$, both over $n$ variables. Let $T = \{1, \ldots, 2n\}$. The following steps are repeated $m$ times, each time using independent random bits.

**V1:** The verifier randomly picks points $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k \in T^n$, where $k = \lceil n \log n \rceil + 2$, and sends them to the prover.

**P1:** The prover randomly chooses a permutation $\varphi$ and sends $y_l = p_{B_0} \circ \varphi(\boldsymbol{r}_l)$, for $l = 1, \ldots, k$, to the verifier.

**V2:** The verifier randomly picks $j \in \{0, 1\}$ and sends it to the prover.

**P2:** The prover sends a permutation $\pi$ to the verifier such that $p_{B_j} \circ \pi(\boldsymbol{r}_l) = y_l$, for $l = 1, \ldots, k$.

**V3:** Finally, the verifier accepts if and only if this latter condition regarding $\pi$ in fact holds.

*Proof of Theorem 5-2* By arguments similar to those in Section 3, the above protocol constitutes an interactive proof system for 1-BPI: if $B_0 \cong B_1$ and the prover behaves as described in the protocol, then the verifier will always accept. If $B_0 \ncong B_1$, then the verifier will accept with probability at most 3/4 in each round, no matter what the prover does, and hence, with probability at most $(3/4)^m$ after $m$ rounds.

*Proof of Theorem 5-3* For the zero-knowledge property, it is easy to see that the specific verifier in the protocol gets no extra information. The communication between $P$ and $V$ can be produced with equal distribution by the following algorithm $M_V$: randomly pick $j \in \{0, 1\}$, $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k \in T^n$, and a permutation $\varphi$ and output $\boldsymbol{r}_l$ and $p_{B_j} \circ \varphi(\boldsymbol{r}_l)$, for $l = 1, \ldots, k$, and furthermore, $j$ and $\varphi$.

*Proof of Theorem 5-4* By arguments similar to those in [GMW91], one can show that $P$ in fact conveys no knowledge to any verifier, even verifiers that deviate from the above protocol. We give a very short description so that a reader familiar with [GMW91] can easily fill in the details.

*Proof of Theorem 5-5* Let $V^*$ be an interactive machine. We cannot simply define $M_{V^*}$ the same way as for the specific verifier $V$ above, because it has chosen $j$ uniformly at random in step V2, and therefore $M_V$ could do the same thing. However, in general, $V^*$ can make its choice of $j$ dependent on the points $((\boldsymbol{r}_1, y_1), \ldots, (\boldsymbol{r}_k, y_k))$ that it gets from the prover after the first round. On the other hand, by only knowing $j$ in advance, $M_V$ could produce an isomorphism $\varphi$ as above.

*Proof of Theorem 5-6* The way $M_{V^*}$ works is as follows. The $M_{V^*}$ algorithm starts by simulating $V^*$ to produce the points $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_k \in T^n$. Then $M_{V^*}$ randomly picks $j \in \{0, 1\}$, and a permutation $\varphi$. This is similar to $M_V$ above, except that $j$ is now considered only as a candidate for the value that will actually be produced by $V^*$. Next, $M_{V^*}$ simulates $V^*$ when $V^*$ gets the points $(\boldsymbol{r}_1, y_1), \ldots, (\boldsymbol{r}_k, y_k)$ from the prover, thereby obtaining the value $j_{V^*}$ that $V^*$ will send to the

prover after the first round. Then, in case that $j = j_{V^*}$, $M_{V^*}$ is "lucky," and can make the same output as $M_V$ above, namely, $\boldsymbol{r}_l$ and $p_{B_j} \circ \varphi(\boldsymbol{r}_l)$, for $l = 1, \ldots, k$, and $j$ and $\varphi$. If $j \neq j_{V^*}$, then $\varphi$ is the wrong permutation, and $M_{V^*}$ cannot make a legal output. Instead, $M_{V^*}$ repeats this whole process until it gets lucky, i.e., until $j = j_{V^*}$, and then makes an output.

*Proof of Theorem 5-7* The probability that $M_{V^*}$ is lucky is $1/2$. Therefore we expect $M_{V^*}$ to repeat this process twice; hence, $M_{V^*}$ runs in expected polynomial time.

*Proof of Theorem 5-8* Finally, the output distribution of $M_{V^*}$ is identical to that of the conversation of $P$ and $V^*$. Intuitively this is clear, because roughly speaking, $M_{V^*}$ simply waits until it can do the same trick as $M_V$ from above. Therefore, the output of $M_{V^*}$ might be delayed, but has the same distribution. There are some subtleties that one must take care of, but a formal argument can then easily be adapted from [GMW91]. Note also that in fact $M_{V^*}$ must produce the conversation of several rounds of the protocol.

**Proof of Theorem 5** □

*5-2* Clearly, Theorem 5 extends to the isomorphism problem for arithmetic circuits.

*5-3* The interactive proof for 1-BPNI presented in Section 3 might not be zero knowledge, because in the first step the verifier can present points to the prover that are obtained in a different way than by random guesses, and the answers from the prover later on might give some extra information to the verifier. For the graph-nonisomorphism problem GNI, this problem is solved by letting the verifier "prove to the prover" that it has a permutation in hand that was used to produce the graph sent to the prover [GMW91]. However, there are some problems in adapting this method to 1-BPNI that arise from the way we describe polynomials in the interactive proofs, namely, as a set of points. We leave it as an open problem to show that 1-BPNI has a zero-knowledge interactive proof.

# Acknowledgments

We thank Manindra Agrawal for many enjoyable discussions. In particular, the observation that the isomorphism problem for polynomials is more general than the one for graphs (see end of Section 3) is due to him. Thanks also go to Bernd Borchert, Lance Fortnow, and Uwe Schöning for very helpful comments.

**Acknowledgment of support:** Supported in part by DAAD, Acciones Integradas.

# References

[AH91] W. Aiello and J. Håstad. Statistical zero-knowledge languages can be recognized in two rounds. *Journal of Computer and System Sciences*, 42:327–345, 1991.

[AT96] M. Agrawal and T. Thierauf. The Boolean isomorphism problem. In *37th Symposium on Foundation of Computer Science*, pages 422–430, New York, 1996. IEEE Computer Society Press.

[Bab85] L. Babai. Trading group theory for randomness. In *17th ACM Symposium on Theory of Computing*, pages 421–429, New York, 1985. ACM Press.

[BCG+96] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52:421–433, 1996.

[BCW80] M. Blum, A. Chandra, and M. Wegman. Equivalence of free Boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters*, 10:80–82, 1980.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity Theory I.* EATCS Monographs on Theoretical Computer Science. Berlin, 1988. Springer-Verlag.

[BDG91] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity Theory II.* EATCS Monographs on Theoretical Computer Science. Berlin, 1991. Springer-Verlag.

[Ber68] E. Berlekamp. *Algebraic Coding Theory.* New York, 1968. McGraw-Hill.

[BHZ87] R. Boppana, J. Håstad, and S. Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 25:27–32, 1987.

[BR93] B. Borchert and D. Ranjan. The subfunction relations are $\Sigma_2^p$-complete. Technical Report MPI-I-93-121, MPI Saarbrücken, 1993.

[BRS96] B. Borchert, D. Ranjan, and F. Stephan. On the computational complexity of some classical equivalence relations on Boolean functions. Technical Report TR96-033, Electronic Colloquium on Computational Complexity, http://www.eccc.uni-trier.de/eccc/, 1996.

[CK91] P. Clote and E. Kranakis. Boolean functions, invariance groups, and parallel complexity. *SIAM Journal on Computing*, 20:553–590, 1991.

[FHS78] S. Fortune, J. Hopcroft, and E. Schmidt. The complexity of equivalence and containment for free single variable program schemes. In *5th Annual International Colloquium on Automata, Languages and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 227–240, Berlin, 1978. Springer-Verlag.

[For89] L. Fortnow. The complexity of perfect zero-knowledge. *Advances in Computing Research*, 5:327–343, 1989.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18:186–208, 1989.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38:691–729, 1991.

[GS89] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. *Advances in Computing Research*, 5:73–90, 1989.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Reading, MA, 1979. Addison-Wesley.

[IM83] O. Ibarra and S. Moran. Probabilistic algorithms for deciding equivalence of straight-line programs. *Journal of the ACM*, 30:217–228, 1983.

[Sch80] J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27:701–717, 1980.

[Sch88] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37:312–323, 1988.

[Sch89] U. Schöning. Probabilistic complexity classes and lowness. *Journal of Computer and System Sciences*, 39:84–100, 1989.

[Sip83] M. Sipser. A complexity theoretic approach to randomness. In *15th ACM Symposium on Theory of Computing*, pages 330–335, New York, 1983. ACM Press.

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. In *International Symposium on Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226, Berlin, 1979. Springer-Verlag.