Electronic Colloquium on Computational Complexity - Reports Series 1996 - available via:

Revision 01 of
ECCC TR96-040

FTP: ftp.eccc.uni-trier.de:/pub/eccc/
WWW: http://www.eccc.uni-trier.de/eccc/
Email: ftpmail@ftp.eccc.uni-trier.de with subject 'help eccc'

The Isomorphism Problem for One-Time-Only Branching Programs and Arithmetic Circuits

Thomas Thierauf *

Abt. Theoretische Informatik
Universität Ulm
89069 Ulm, Germany

thierauf@informatik.uni-ulm.de

Abstract

We investigate the computational complexity of the *isomorphism problem for one-time-only branching programs (1-BPI)*: on input of two one-time-only 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.

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.

We use the known arithmetization of one-time-only 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.

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 to probabilistically check polynomial identities.

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

*Supported in part by DAAD, Acciones Integradas.

1 Introduction

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

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 in a series of papers with respect to its computational complexity (see for example [AT96, BR93, BRS96, CK91]).

The equivalence problems for Boolean circuits, branching programs, and Boolean formulas are known to be coNP complete. Asking for isomorphism, puts, roughly speaking, an existential quantifier in front of the problem. Therefore, the corresponding isomorphism problems are in the second level of the polynomial hierarchy, Σ_2^p. An obvious question is whether these isomorphism problems are complete for Σ_2^p. This question was solved by Agrawal and Thierauf [AT96] who showed that *none* of these isomorphism problems is complete for Σ_2^p, unless the polynomial hierarchy collapses. Thus, loosely speaking, the existential quantifier we get by seeking for an isomorphism doesn't seem to add full NP power to the corresponding equivalence problem.

The most prominent example that supports the latter observation might be he *graph ismorphism 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. But GI is *not* NP-complete, unless the polynomial hierarchy collapses [GMW91, BHZ87] (see also [Sch89]).

For a restricted class of branching programs, the *one-time-only branching programs*, where, on each path, each variable is tested at most once (see next section for precise definitions), the equivalence problem is easier then for general ones: it can be efficiently solved by a randomized Las Vegas type algorithm [BCW80], it is in the complexity class coRP. Therefore, putting an existential quantifier in front, the isomorphism problem for one-time-only branching programs, 1-BPI, is in NP·coRP. Motivated by the examples above, we ask whether 1-BPI is NP-hard.

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.

One crucial point to prove these results is that both, one-time-only branching programs and arithmetic circuits can be arithmetized, yielding equivalent multivariate polynomials [BCW80, IM83]. In the case of one-time-only branching programs, these polynomials are linear. 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.

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 in order to hide the syntactic structure of the input polynomial that are manipulated. This problem doesn't occur when dealing with graphs. However, this 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.

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

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 hold more general for the corresponding congruence problems.

2 Preliminaries

2.1 Basic Definitions

We will use fairly standard notions of complexity theory. We refer the reader to [BDG88, BDG91, HU79] for definitions of complexity classes like P, NP, RP or BPP. Σ_k^p are the levels of the polynomial hierarchy. For any class $\mathcal{C}$, we denote the complement class by co $\mathcal{C}$.

BP · NP is the class of sets L such that there exists a set $A \in$ NP and a

polynomial p such that for every x

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

where y is chosen uniformly at random from $\Sigma^{p(|x|)}$.

NP $\cdot$ coRP is the class of sets L such that there exists a set $A \in$ coRP and a polynomial p such that for every x, we have

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

Interactive proofs were defined in [GMR89]. An *interactive proof system* for a set L consists of a prover P and 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.

IP denotes the class of sets that have an interactive proof system. IP[k] is the subclass of IP where the verifier and the prover exchange at most k messages.

Arthur-Merlin games were introduced in [Bab85]. They are similar 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, while the prover doesn't know the random bits of the verifier. AM[k] denotes when k many messages can be send, and AM = AM[2].

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 easy to see that AM = BP $\cdot$ NP.

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 doesn't reveal any extra information to any verifier V^* besides the fact that x is in L: the messages exchanged with the prover look random to V^*. That is, these messages could have been produced by V^* himself.

Definition 2.1 *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 $c > 0$, 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.*

P and V are a perfectly zero-knowledge proof system for L *if it is an interactive proof system for L and P is perfectly zero-knowledge on L.*

The string H in the definition is needed for being able to compose two zero-knowledge protocols to one zero-knowledge protocol: the history H from the first protocol, which is known to the verifier in the second protocol, doesn't help the verifier. For a more detailed discussion of this definition see [GMR89, GMW91]. Also the need for *expected* polynomial time is explained there.

2.2 Verifying Polynomial Identities

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 monoms (i.e., products of the variables). Polynomials of degree 1 are called *multilinear*.

Given some polynomial p written as an arithmetic expression, we want to find out whether p is in fact the zero polynomial. Note that the obvious algorithm, namely to tranform the arithmetic expression in a sum of monoms 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 developped 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.

Theorem 2.2 [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 $p(a_1, \ldots, a_n) \neq 0$.*

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 2.3 *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 $p \not\equiv 0 \Longleftrightarrow \exists \mathbf{a} \in T^n \quad p(\mathbf{a}) \neq 0$.*

By enlarging T even further, we can achieve 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 2.4 *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 $\mathbf{r} = (r_1, \ldots, r_n)$ be a random element from T^n. Then $p(\mathbf{r}) \neq 0$ with probability at least $\frac{1}{2}$.*

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

2.3 Branching Programs

A *branching program* B in n Boolean variables $x_1, \ldots, x_n$ is a directed acyclic graph with the following type 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 labelled 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 labelled either by *accept* or *reject*.

We call a branching program *one-time-only*, 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.

A one-time-only branching program is called *ordered*, if the order of occurance of the variables on each path is consistent with some ordering on the set of variables.

Branching programs are also called *binary decision diagrams (BDD)* or *Boolean graphs*. One-time-only branching program and an ordered branching program are also called *free binary decision diagram (FBDD)* or *free Boolean graphs* and *ordered binary decision diagram (OBDD)*, respectively.

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

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 to decide 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, and hence, BPE is coNP-complete.

Blum, Chandra, and Wegman [BCW80] showed that the equivalence problem for *one-time-only branching programs*, 1-BPE, is easier because one can transform them 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 monom, the conjunction of the literals occuring along that path. Then the function computed by B is simply the disjunction of all monoms coming from accepting paths of B.

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 assignment $\mathbf{a} \in \{0,1\}^n$, exactly one path of B evaluates to true under $\mathbf{a}$. Therefore, at most one product term in p_B will be one on input $\mathbf{a}$. Hence, B and p_B agree on $\{0,1\}^n$. That is,

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

It is easy to get an arithmetic expression for p_B from B that has about the same size as B. Note however that, written as a sum of monoms, p_B may consist of exponentially many terms in the size of B. So 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.

Since B is one-time-only, p_B is a *multilinear* polynomial. Now let B' be another one-time-only branching program and $p_{B'}$ the corresponding polynomial. If B and B' are equivalent, then p_B and $p_{B'}$ agree on $\{0,1\}^n$, a 2 element set. By Corollary 2.3 (applied to $p = p_B - p_{B'}$), it follows that p_B and $p_{B'}$ agree on $\mathbf{Q}$. Now, choosing T in Corollary 2.4 as $T = \{1, \ldots, 2n\}$, we will detect an inequivalence with probability more than $\frac{1}{2}$. It follows that 1-BPE $\in$ coRP.

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

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

The *isomorphism problem for branching programs* is BPI $= \{ (B, B') \mid B \cong B' \}$. The isomorphism problem for one-time-only 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 Σ_2^p, unless the polynomial hierarchy collapses to its third level, Σ_3^p.

For one-time-only 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 to decide whether two one-time-only 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, Σ_2^p.

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

2.4 Arithmetic Circuits

An *arithmetic circuit over a field* $\mathbf{F}$ is a circuit, where the inputs are field elements and the (fan-in 2) gates perform the field operations $+$, $-$, and $\times$. (We could also allow division as long as a circuit guarantees to not divide by zero on any input.) *Circuit size* and *depth* are defined as usual.

Ibarra and Moran [IM83] considered the equivalence problem for arithmetic circuit (called *straight-line programs* there). They give probabilistic polynomial-time algorithms for circuits over infinite fields. This is split into two cases, depending on whether the field has characteristic 0 or greater than 0. If the field $\mathbf{F}$ has characteristic 0, it contains a subfield isomorphic to $\mathbf{Q}$, 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.

If a circuit C has n input variables $x_1, \ldots, x_n$, then C computes a multivariate polynomial p_C over $\mathbf{Q}$. If C has depth d then p_C has degree at most 2^d. Therefore, choosing T in Corollary 2.4 as $T = \{1, \ldots, 2n2^d\}$, we will detect an inequivalence with probability more than $\frac{1}{2}$ at a random point from T^n.

However, we don't have a polynomial-time procedure yet because the function values of p_C on T^n could be as large as $(2n2^d)^{2^d} \leq 2^{N2^N}$ for $N = n + d$. 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.) $p_C \pmod m$ might have more zeros than p_C, however, not too many:

Lemma 2.5 [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. 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 don't divide y. Hence, for m randomly chosen from M, we have

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

□

The probabilistic zero test now works as follows.

Corollary 2.6 *Let $p(x_1, \ldots, x_n)$ be a polynomial of degree 2^d over $\mathbf{Q}$, $T = \{1, \ldots, 2n2^d\}$ and $M = \{1, \ldots, 2^{2N}\}$, where $N = n + d$. Choose $\mathbf{r}_1, \ldots, \mathbf{r}_{6N}$ from T^n and $m_1, \ldots, m_{6N}$ from M independently at random. Then $p(\mathbf{r}_i) \not\equiv 0 \pmod{m_i}$, for some i, with probability at least $\frac{1}{2}$.*

Proof. By Corollary 2.4 and Lemma 2.5, $\mathbf{Pr}[p(\mathbf{r}_i) \not\equiv 0 \pmod{m_i}] \geq \frac{1}{2}\frac{1}{3N}$, for any pair $\mathbf{r}_i$, m_i. Therefore, $\mathbf{Pr}[p(\mathbf{r}_i) \equiv 0 \pmod{m_i}$ for all $i\,] \leq \left(1 - \frac{1}{6N}\right)^{6N} \leq \frac{1}{2}$. □

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 now is to switch from $\mathbf{F}$ to the ring of polynomials over GF(q). This is certified by the the following lemma.

Lemma 2.7 [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* GF(q)[x].

Since q is prime, GF(q) is a field and therefore, the ring 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, GF(q)[x] is what is sometimes called an *euclidian ring*.) One can easily verify that this already suffices in the assumption of Theorem 2.2 and its corollaries, instead of having a field. Hence, we can apply the zero test for p over the polynomial ring GF(q)[x].

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 2.5 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 choosen polynomial $r \in \mathrm{GF}(q)[x]$ of small degree and $a_i \in \mathrm{GF}(q)[x]$, for $i = 1, \ldots, n$.

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

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 preceeding section that one can transform a one-time-only branching program into an equivalent arithmetic circuit of about the same size. Though arithmetic circuit are the more general concept, we prove our main result for one-time-only branching program first, and then explain how to extend it to solve the isomorphism problem for arithmetic circuit.

3 An Interactive Proof for 1-BPNI

We show that there is a two round interactive proof for the one-time-only branching program nonisomorphism problem, 1-BPNI.

We start by recalling the idea of the interactive proof for the graph nonisomorphism 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 φ, and sends $H = \varphi(G_i)$ to the prover. Now the prover is asked to find out what the value of i is. The verifier will accept only if the prover gives the right answer.

When the input graphs are *not* isomorphic, then the prover can find out i easily. However, when the graphs are isomorphic, both could have been used by the verifier to compute H, so that no prover can find out i. Therefore, the answer of any prover is correct with probability at most $\frac{1}{2}$.

First of all note that we cannot directly adapt this protocol to branching programs. The reason for this 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.

A way out of this would be a normal form for one-time-only 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 theorie 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 doesn't have an NP oracle available and there is no analog learning result for one-time-only branching programs without an NP oracle.

The idea to get around this problem is as follows. On input of two given one-time-only 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 whole B, *the verifier evaluates B at a randomly choosen point* $\mathbf{r} \in T^n$, where T is some appropriate test domain. The prover is now asked to tell which of B_0, B_1 was used to obtain the point $(\mathbf{r}, B(\mathbf{r}))$. If B_0 and B_1 are isomorphic, then the prover cannot detect and has to guess. So she will fail with probability $\frac{1}{2}$. 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 $(\mathbf{r}, B(\mathbf{r}))$. This is because, by Corollary 2.4, different multilinear polynomials can agree on T on at most $\frac{1}{2}$ of the points for $T \geq 2n$. That is, the origin of $(\mathbf{r}, B(\mathbf{r}))$ is unique with high probability. With an additional round of communication the prover can *always* convince the verifier from the *non-isomorphism* of B_0 and B_1. We give the details below.

Theorem 3.1 1-BPNI $\in$ IP[4].

Proof. The IP-protocol described in Figure 1 accepts 1-BPNI. The input are two one-time-only branching programs B_0 and B_1, both over n variables. Let $T = \{1, \ldots, 2n\}$.
We show that the above protocol works correctly.

Case 1: $B_0 \ncong B_1$. We show that there is a prover such that the verifier always accepts.

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

P: the prover is expected to send $j \in \{0,1\}$ and a permutation φ' to the verifier.

V: 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 $p_{B_j} \circ \varphi'(\mathbf{r}_l) = y_l$, for $l = 1, \ldots, k$. If the test fails, the verifier rejects. Otherwise, he sends φ to the prover.

P: the prover is expected to send a point $\mathbf{r}' \in T^n$ to the verifier.

V: finally, the verifier accepts if and only if $p_{B_i} \circ \varphi(\mathbf{r}') \neq p_{B_j} \circ \varphi'(\mathbf{r}')$.

Figure 1: Interactive proof for 1-BPNI.

The prover can cycle through all permutations and check for both, p_{B_0} and p_{B_1}, whether it matches with the set R sent by the verifier. Say that polynomial $p_{B_0} \circ \varphi'$ does so. Then the prover sends $j = 0$ and φ' to the verifier.

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.

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 anyway been zero and the verifier accepts. On the other hand, if $i \neq j$, then the verifier will send φ to the prover 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 $\frac{1}{2}$ of the points of T^n by Corollary 2.4. Therefore, the prover can find a point $\mathbf{r}' \in T^n$ such that $p_{B_j} \circ \varphi'(\mathbf{r}') \neq p_{B_i} \circ \varphi(\mathbf{r}')$, and send it to the verifier who will accept. In summary, the verifier accepts with probability one.

Case 2: $B_0 \cong B_1$. We show that for any prover, the verifier accepts with probability at most $\frac{3}{4}$. By executing the protocol several times in parallel, the acceptance probability can be made exponentially small.

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

By Corollary 2.4, for any φ' such that $p_{B_0} \circ \varphi \neq p_{B_1} \circ \varphi'$ we have

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

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

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

Therefore, considering all such φ', we get that

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

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

We can directly come down 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 $\mathbf{r}'$ as above without knowing φ, and hence can already send it in the first round to the verifier, who can then make all his tests. However, then we get another difficulty: when T has exponential size, the values of the poynomials might be up to double exponential. Then the polynomial time verifier can no longer deal with such numbers. We will show in the next section how the verifier can still manage his task.

As already 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[2] which, in turn, is the same as BP · NP [Bab85, GS89].

Corollary 3.2 1-BPNI $\in$ BP · NP.

Schöning [Sch88] gives a direct proof that the graph isomorphism problem is in AM by using hash functions. By combining the technique from Theorem 3.1 and that in [Sch88], we can as well obtain Corollary 3.2 directly.

Note that both classes, BP · NP and NP · coRP can, very loosly, be considered as some, more or less slight extensions of NP. In this sense, we have shown that 1-BPI is in a slight extension of NP ∩ coNP.

Corollary 3.3 1-BPI ∈ NP · coRP ∩ BP · coNP.

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

Corollary 3.4 1-BPI *is not* NP*-hard, unless* PH $= \Sigma_2^p$.

Since *ordered branching programs*, OBDD's, are a restricted form of one-time-only branching programs, Corollary 3.4 can be applied. Since the equivalence problem for OBDD's is in P [FHS78], the isomorphism problem for OBDD's is in NP.

Corollary 3.5 *The isomorphism problem for OBDD's is not* NP*-complete, unless* PH $= \Sigma_2^p$.

4 Extension to Arithmetic Circuits

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.

Given two arithmetic circuits C_0, C_1 with n inputs and of depth d. We take the protocol from the previous section and modify it according to Corollary 2.6. That is, let $T = \{1, \ldots, 2n2^d\}$ and $M = \{1, \ldots, 2^{2N}\}$, where $N = n + d$. The verifier starts by randomly choosing $i \in \{0, 1\}$ and a permutation φ as before, and now $6Nk$ points $\mathbf{r}_1, \ldots, \mathbf{r}_{6Nk} \in T^n$, where $k = \lceil n \log n \rceil + 2$, and for each point $\mathbf{r}_l$ a number $m_l \in M$. Then the verifier computes $y_l = p_{B_i} \circ \varphi(\mathbf{r}_l) \bmod m_l$, for $l = 1, \ldots, 6Nk$, and sends the set of triples $R = \{ (\mathbf{r}_l, y_l, m_l) \mid l = 1, \ldots, 6Nk \}$ to the prover. The rest of the protocol is adapted in the obvious way.

Combining the argument in the previous section with Corollary 2.6, the verifier will accept two isomorphic arithmetic circuits with probability at most $\frac{3}{4}$. Two nonisomorphic arithmetic circuits are still accepted with probability one.

The case of infinite fields of characteristic greater than 0 is analogous. As briefly explained in Section 2.4 the test domain becomes now 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. We omit further details.

Theorem 4.1 *The nonisomorphism problem for arithmetic circuits over infinite fields is in* $\mathrm{BP} \cdot \mathrm{NP}$.

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 in order 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 2.3, when $q > 2^d$, we have that two polynomials over $\mathrm{GF}(q)$ are equivalent if and only if they are equivalent over any extension field.

The verifier must be able to evaluate a polynomial at a given point in the extension field. For this, he 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 $\mathrm{GF}(q)[x]$ of degree t and check irreducibility in polynomial time using the Berlekamp algorithm [Ber70]. So the verifier will find an irreducible polynomial $\phi(x)$ in polynomial time . Then $\mathrm{GF}(q^t)$ is isomorphic to $\mathrm{GF}(q)[x]/\phi(x)$. Therefore, knowing $\phi(x)$, the verifier can do all computation needed in polynomial time. Now, the protocol can proceed as in the case of branching programs.

Theorem 4.2 *The nonisomorphism problem for arithmetic circuits of depth d over a finite field of size more than 2^d is in* $\mathrm{BP} \cdot \mathrm{NP}$.

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

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

Proof. The IP-protocol described in Figure 2 accepts 1-BPI and has the perfect zero-knowledge property. The input are two one-time-only 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.

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

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

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

P: the prover is expected to send a permutation π to the verifier such that $p_{B_j} \circ \pi(\mathbf{r}_l) = y_l$, for $l = 1, \ldots, k$.

V: finally, the verifier accepts if and only if this latter condition about π in fact holds.

Figure 2: Perfect zero-knowledge interactive proof for 1-BPI.

By arguments similar to those in Section 3, the protocol in Figure 2 is 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 $\frac{3}{4}$ in each round, no matter what the prover does, and hence, with probability at most $(\frac{3}{4})^m$ after m rounds.

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 $j \in \{0,1\}$, $\mathbf{r}_1, \ldots, \mathbf{r}_k \in T^n$, and a permutation φ and output $\mathbf{r}_l$ and $p_{B_j} \circ \varphi(\mathbf{r}_l)$, for $l = 1, \ldots, k$, and furthermore j and φ.

By arguments similar to those in [GMW91] one can show that P in fact conveys no knowledge to *any* verifier, even ones 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.

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

The way M_{V^*} works is as follows. M_{V^*} starts by simulating V^* to produce the points $\mathbf{r}_1, \ldots, \mathbf{r}_k \in T^n$. Then M_{V^*} randomly picks $j \in \{0,1\}$ and a permutation φ. This is like M_V above, but j is now considered only as a *candidate* for the value that will actually produced by V^*. Next, M_{V^*} simulates V^* when V^* got the points $(\mathbf{r}_1, y_1), \ldots, (\mathbf{r}_k, y_k)$ from the prover. Thereby M_{V^*} obtains the value j_{V^*} that V^* will send to the prover after the first round. Now, in case that $j = j_{V^*}$, M_{V^*} was lucky and can make the same output as M_V above, namely $\mathbf{r}_l$ and $p_{B_j} \circ \varphi(\mathbf{r}_l)$, for $l = 1, \ldots, k$, and j and φ. If $j \neq j_{V^*}$, then φ 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.

The probability that M_{V^*} is lucky is $\frac{1}{2}$. Therefore we expect M_{V^*} to repeat this process twice, and hence, M_{V^*} runs in expected polynomial time.

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 one has to take care of, but a formal argument can now easily be adapted from [GMW91]. Note also that in fact M_{V^*} has to produce the conversation of several rounds of the protocol. □

Clearly, Theorem 5.1 extends to to the isomorphism problem for arithmetic circuits.

The interactive proof for 1-BPNI presented in Figure 1 might not be zero-knowledge since in the first step, the verifier can present points to the prover that are obtained in a different way than by random guesses. Then 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 he has a permutation in hand which was used to produce the the graph sent to the prover [GMW91]. However, there are some problems to adapt 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.

Acknowledgements

I want to thank Manindra Agrawal, Bernd Borchert, Lance Fortnow, and Uwe Schöning for helpful discussions.

References

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

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

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

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

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

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity Theory I.* Springer, 1988.

[BDG91] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity Theory II.* Springer, 1991.

[Ber70] E. R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24:713–735, 1970.

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

[BR93] B. Borchert and D. Ranjan. The subfunction relations are σ_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, ECCC: Electronic Colloquium on Computational Complexity, http://www.eccc.uni-trier.de/eccc/, 1996.

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

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

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

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

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

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

[HU79] J. E. Hopcroft and J. D. Ullman. *Introdction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, Mass., USA, 1979.

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

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

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

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

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. In *ISSAC '79: Proc. Int'l. Symp. on Symbolic and Algebraic Computation,* Lecture Notes in Computer Science, Vol. 72. Springer-Verlag, 1979.