TRYKKESTED:
MATEMATISK INSTITUT
AARHUS UNIVERSITET

PB - 343 Frandsen et al.: Uniform Arithmetic Circuit Complexity

ISSN 0105-8517

Some Results on Uniform Arithmetic Circuit Complexity

Gudmund S. Frandsen
Mark Valence
David A. Mix Barrington

DAIMI PB – 343
January 1991

COMPUTER SCIENCE DEPARTMENT
AARHUS UNIVERSITY
Ny Munkegade, Building 540
DK-8000 Aarhus C, Denmark

ISSN 0105-8517

Some Results on Uniform Arithmetic Circuit Complexity

Gudmund S. Frandsen
Mark Valence
David A. Mix Barrington

DAIMI PB – 343
January 1991

COMPUTER SCIENCE DEPARTMENT
AARHUS UNIVERSITY
Ny Munkegade, Building 540
DK-8000 Aarhus C, Denmark

Some Results on Uniform Arithmetic Circuit Complexity.

Gudmund S. Frandsen [1] [2] [3]
Computer Science Department
Aarhus University
DK-8000 Aarhus C, Denmark
gsfrandsen@daimi.aau.dk

Mark Valence
Department of Mathematics and Computer Science
Dartmouth College
Hanover, NH 03755, USA
kurash@dartmouth.edu

David A. Mix Barrington [4]
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003, USA
barrington@cs.umass.edu

January 23, 1991

[1]This research was partially carried out, while visiting Dartmouth College, New Hampshire.
[2]This research was partially supported by the Danish Natural Science Research Council (grant No. 11-7991).
[3]This research was partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).
[4]This research was supported by NSF Computer and Computation Theory grant CCR-8714714

Abstract

We introduce a natural set of arithmetic expressions and define the complexity class AE to consist of all those arithmetic functions (over the fields $\mathbf{F}_{2^n}$) that are described by these expressions.

We show that AE coincides with the class of functions that are computable with constant depth and polynomial size unbounded fan-in arithmetic circuits satisfying a natural uniformity constraint (DLOGTIME-uniformity).

A 1-input and 1-output arithmetic function over the fields $\mathbf{F}_{2^n}$ may be identified with an n-input and n-output Boolean function when field elements are represented as bit strings.

We prove that if some such representation is X-uniform (where X is P or DLOGTIME) then the arithmetic complexity of a function (measured with X-uniform unbounded fan-in arithmetic circuits) is identical to the Boolean complexity of this function (measured with X-uniform threshold circuits).

We show the existence of a P-uniform representation and we give partial results concerning the existence of representations with more restrictive uniformity properties.

Introduction

When a family of circuits is constructed to compute a function, some kind of uniformity restriction is usually applied, in order that the circuits may be feasible to construct. Traditionally, uniformity constraints have been specified by requiring some description of the circuits to be generated or accepted by a machine of bounded complexity [Bor77, Ruz81, BDG].

More recently, a different approach using expressibility has arisen. Some complexity classes defined in terms of functions computable by uniform circuit families coincide with classes of functions that are specified using logic expressions with Boolean operators and various predicates and quantifiers [Imm87, Imm89, BIS90].

The first author has participated in the development of an algebraic model of computation based on circuits that use arithmetic gates over characteristic 2 finite fields [BFS88]. By representing field elements as bit strings, it is shown that the (non-uniform) complexity measure of a Boolean function measured in the arithmetic model (size/depth of circuits using unbounded fan-in Π, Σ-gates) is identical to the (non-uniform) complexity measured in a Boolean model (size/depth of circuits using threshold circuits) up to a polynomial blow up in size and a constant factor in depth. In this sense, one may regard the arithmetic model of computation as an abstraction of the Boolean model.

The possibility of a similar uniform relationship depends on finding suitably uniform representations of the family of fields involved. In this paper, we consider P-uniformity and DLOGTIME-uniformity.

P-uniformity is a very basic kind of uniformity [All89]. A P-uniform family of circuits is computationally feasible to construct. Recent results show how to construct an irreducible polynomial over $\mathbf{F}_2$ in deterministic polynomial time [Sho90]. This fact allows us to find a P-uniform representation of the characteristic 2 finite fields, which in turn leads to a P-uniform version of the result in [BFS88].

It is not clear whether a P-uniform family of circuits corresponds to an algorithm of complexity proportional to the size/depth of the family, since constructing the circuit for a specific input size may require polynomial sequential time. When the computational powers are comparable, it is still possible that an essential part of the computation is carried out by the circuit constructor rather than the circuit itself. This has motivated the study of more restrictive kinds of uniformity, where the resources spent in accepting or generating a description of some circuit family correspond more closely to (or strictly dominate) the size/depth of the family itself [Ruz81].

We shall use the restrictive DLOGTIME-uniformity measure proposed for studying the substructure of NC^1 [BIS90]. Usually, time complexities below linear are not considered, but it is possible to give a meaningful definition of TIME($\log(n)$) using a random access input tape [Ruz81]. DLOGTIME-uniformity is defined in terms of this complexity class.

We construct a DLOGTIME-uniformly "strong" (see definition 3.3) representation of the fields $\mathbf{F}_{2^n}$ ($n \neq 0 \bmod 4$) using a recent result about the existence of both self-dual and normal bases for these fields [LeWe88]. This allows a DLOGTIME-uniform version of one of the two inclusions in the main equivalence result from [BFS88]. The use of DLOGTIME-uniformity has the advantage that our results are valid for any less restrictive kind of uniformity.

It is known that the class of functions computed by DLOGTIME-uniform families of circuits that have polynomial size and constant depth coincides with the class of functions that are described by logic expressions, provided the allowed types of quantifiers in the expressions correspond to the allowed types of gate in the circuits [Imm89, BIS90].

We prove an arithmetic analogue to this result involving a natural and robust class of arithmetic expressions. Basically, we allow the usual arithmetic operations in an expression: $+$, $\cdot$, $(\cdot)^{\text{power}}$, $\sum_{i=1}^{\text{bound}}$, $\prod_{i=1}^{\text{bound}}$ with the important restriction that *bound* must be a polynomial expression in indices and n. (If the restriction is omitted, we can express functions that are equivalent to the discrete logarithm, and there is little hope of finding polynomial size circuits for such functions using known techniques.)

All our results can be rephrased for some other fixed finite characteristic. If the characteristic is unbounded then we inherit various problems from the non-uniform case (see [FrSt88]) in addition to new ones concerning the construction of suitable prime numbers and irreducible polynomials.

SECTION 1

Uniform Logic Circuits and Logical Expressions

Definition 1.1

Unbounded Fan-in Logic Circuits

An *unbounded fan-in logic circuit* is a directed acyclic graph with n external nodes (input gates) and arbitrarily many internal nodes (function gates). Each internal node computes a function of arbitrarily many Boolean inputs (fan-in) and one Boolean output that feeds into arbitrarily many other gates (fan-out). Gates whose outputs feed no other gate produce the final output for the circuit.

The *depth* of a logic circuit is the length of the longest path from the output gate to one of the input gates. The *size* of a circuit is the number of gates it has. Both of these values are measured as functions of the number of input gates, n.

A *family* of circuits $\{C_n\}$ is a set consisting of one circuit C_n for each input size $n \geq 1$. We denote by SIZE-DEPTH($S(n), D(n)$) the class of functions computed by circuit families with size $S(n)$ and depth $D(n)$.

When circuits use AND, OR and NOT gates the functions in SIZE-DEPTH ($n^{O(1)}, \log^i(n)$) form the complexity class AC^i, collectively known as the AC *hierarchy*.

Remark: The above definitions are standard (see [BIS90, CSV84, Imm89, Ruz81]). In this paper we are mainly concerned with circuit families of polynomial size.

The class AC and its subclass AC^0 are further characterised in [FSS84, BIS90, Imm89]. [FSS84] also prove that AC^0 is properly contained in AC^1.

We are interested in two functions with slightly more power than the standard functions of AC. These two functions, Threshold and Majority, are now defined.

Definition 1.2

Threshold and Majority Functions

Threshold functions are of the form:

$$Th_t^{\mathbf{w}}(\mathbf{x}) = 1 \text{ iff } \sum_{i=1}^{m} w_i x_i \geq t$$

where $\mathbf{x} = (x_1, ..., x_m)$ is the vector of Boolean inputs, $\mathbf{w} = (w_1, ..., w_m)$ is a vector of integers, called the weights, and the threshold, t, is also an integer.

Threshold circuits are logic circuits in which the function gates compute Threshold functions.

Majority functions are of the form:

$$Maj^m(\mathbf{x}) = 1 \text{ iff } \sum_{i=1}^{m} x_i > \lfloor \frac{m}{2} \rfloor$$

where $\mathbf{x} = (x_1, ..., x_m)$ is the vector of Boolean inputs.

We define a Majority/Negation circuit as a logic circuit utilizing arbitrary fan-in Majority gates and unary Negation ($\neg$) gates. We also restrict the inputs of the Majority gates by prohibiting any two inputs from coming out of the same source.

$\mathtt{TC}^i$ consists of those families of functions $\{f_n\}$ that are computed by a family of Majority/Negation circuits of polynomially bounded size and depth $O(\log^i(n))$.

Remark: Other definitions of Threshold functions allow the weights to be real-valued, put bounds on them, or do not include weights at all (see [BFS88, CSV84, SkVa85]). Nonuniform complexity measures based on these different types of threshold and majority/negation gates are all equivalent up to polynomial size and constant depth, i.e. $\mathtt{TC}^i$ is a robust class (See [BFS88, MTT60, Mur71]).

We shall use restrictive uniformity properties in the definition of complexity classes, and unbounded fan-in or unbounded weights at threshold gates can not be tolerated. Hence, we define $\mathtt{TC}^i$ in terms of the very simple Majority gate. This is similar to the approach in [BIS90].

It is a well known fact, that $\mathtt{TC}^i$ contains $\mathtt{AC}^i$ and for $i = 0$, the containment is proper [FSS84].

Definition 1.3

Direct Connection Languages and Uniform Circuit Families

The *Direct Connection Language* (*DCL*) of a circuit family $\{C_n\}$ is a set of tuples, each of which partially describes one gate of one circuit, C_n.

Given the *DCL* of a circuit family $\{C_n\}$, we should be able to construct every C_n with some Turing Machine. Typically (see [Ruz81, BIS90]), a tuple from a *DCL* is of the form (n, a, t, b, p), where n denotes the input size of the circuit to construct, and a is the number of a gate of type t, to which the output of gate number b is an input. A padding string p (containing arbitrary symbols) is included in the tuple so that its overall length is n. There may be other elements in the tuple, depending on the type of circuit being constructed.

A family of logic circuits $\{C_n\}$ is $\mathtt{X}$-*uniform*, for complexity class $\mathtt{X}$, exactly when the *DCL* describing $\{C_n\}$ is in $\mathtt{X}$.

Clearly, if $\mathtt{X}_1 \subseteq \mathtt{X}_2$ then an $\mathtt{X}_1$-uniform family of circuits is also $\mathtt{X}_2$-uniform.

Remark: The notion of the direct connection language of a circuit was introduced in [Ruz81] (for gates with fan-in 2) and in [BIS90] (for gates with unbounded fan-in).

Traditionally, $\mathtt{P}$-uniformity has been defined by requiring that a standard encoding of the circuit can be generated in polynomial time [All89]. For polynomial size circuits, that definition is equivalent to the above one.

The use of a single circuit language allows us to speak of $\mathtt{X}$-uniformity independently of $\mathtt{X}$ with more ease, though we shall only use $\mathtt{X} = \mathtt{P}$ or $\mathtt{X} = \mathtt{DLOGTIME}$. The latter is defined now.

Definition 1.4

Logarithmic Time Turing Machines and DLOGTIME

A logarithmic time Turing machine has one read-only tape containing the input of length n, a constant number of work tapes, and a special address tape, used for accessing specific cells of the input tape. The machine is required to run in $O(\log n)$ time. At each step of its computation, the machine can access one bit of the input by writing on its address tape the position of the bit it wishes to read (the position is written as a binary integer, so $O(\log n)$ time is needed to write it).

The class $\mathtt{DLOGTIME}$ is the class of languages accepted by log time Turing machines.

Remark: The Turing machines described above may at first appear severely limited in their computational power. They can, however, find the length n of their input by binary search and perform arithmetic and relational operations on the numbers in the range 0 to $n^{O(1)}$ (see [BIS90]).

Definition 1.5

First Order Expressions

The set of First Order expressions is defined inductively in terms of some infinite set I of indices (variables):

- Each index i ranges over values between 1 and n inclusive, where n is the input size.
- The basic expressions are:
  1. the bit values *false* and *true*
  2. the predicate $X(i)$ (written x_i) where $i \in I$, which is *true* iff the i'th bit of the input string is 1
  3. the binary predicate $\mathtt{BIT}(i, j)$, $i, j \in I$, which is *true* iff the i'th bit in the binary expansion of j is 1.
- For $i, j \in I \cup \{1, n\}$, we include the expressions $(i = j)$, and $(i \leq j)$.
- If e and f are expressions, then we may form new expressions $\exists i.(e)$, $\forall i.(e)$, $e \vee f$, $e \wedge f$, $\neg e$, where $i \in I$.

For an expression e to be *well formed* it should have no free variables, i.e. $FV(e) = \emptyset$, where FV is defined inductively:

- $FV(false) = FV(true) = \emptyset$,
- $FV(X(i)) = \{i\}$,
- $FV(\mathtt{BIT}(i,j)) = FV(i = j) = FV(i \leq j) = \{i, j\} \cap I$,
- $FV(\exists i.(e)) = FV(\forall i.(e)) = FV(e) \setminus \{i\}$,
- $FV(e \vee f) = FV(e \wedge f) = FV(e) \cup FV(f)$,
- $FV(\neg e) = FV(e)$.

A well formed expression describes a family of functions $\{f_n\}$ (one for each input size).

The complexity class `FO` consists of those families of functions $f_n : \{0,1\}^n \to \{false, true\}$ that are described by (well formed) First Order expressions.

We extend `FO` to form the complexity class `FO+Maj` by allowing the Majority quantifier M. If e is an `FO+Maj` expression, we can construct the new expression $Mi.(e)$, which is *true* iff e is *true* for more than half of the possible values of i.

Remark: The complexity class `FO` (and `FO+Maj`) has been studied in [BIS90]. The definition of the complexity class `FO` in terms of logic expressibility is a special instance of a more general framework developed in [Imm87, Imm89].

For later use, we mention some results:

Fact 1.1

1. $\mathtt{DLOGTIME} \subseteq \mathtt{FO}$
2. $\mathtt{FO+Maj} = \mathtt{DLOGTIME}$-uniform $\mathtt{TC}^0$
3. The problem of computing the sum of n n-bit numbers is in $\mathtt{DLOGTIME}$-uniform $\mathtt{TC}^0$

Remark: All these results are proved in [BIS90].

SECTION 2

Uniform arithmetic circuits and arithmetic expressions.

Definition 2.1

Uniform Arithmetic Circuits in Finite Fields of Characteristic 2.

An *unbounded fan-in* arithmetic circuit in the field $\mathbf{F}_{2^n}$ uses the following types of gate

1. unbounded fan-in *sum* (Σ) gates,
2. unbounded fan-in *product* (Π) gates,
3. single input *conjugation* ($\cdot^{2^j}$, $0 \leq j < n$) gates.

It may use a constant g for which it is only known that g generates the field, i.e. $\mathbf{F}_{2^n} = \mathbf{F}_2(g)$.

No multiple wires are allowed, i.e. two wires into the same gate must originate at different gates or inputs.

The *size* of a circuit is the number of gates, and the *depth* is the length of the longest directed path from an input to an output. A *family* of arithmetic circuits $\{C_n\}$ consists of a circuit C_n for each field $\mathbf{F}_{2^n}$ ($n \geq 1$). Size and depth of the family are measured as functions of n.

The *DCL* for a family of arithmetic circuits consists of all tuples of the form (n, a, t, j, b, p) for which gate number a in C_n (which uses arithmetic over $\mathbf{F}_{2^n}$) is of type $t \in$ {sum, product, conjugation, constant, input}. The field j holds an integer when the type is conjugation ($\cdot^{2^j}$) or input (x_j). b is the number of some gate whose output is one of the inputs to gate number a. Each tuple contains a padding string so that its total length is n (the numbers n, a, t, j, b and p are written as binary integers). We shall look at polynomial size circuits.

A family of arithmetic circuits is DLOGTIME-uniform (resp. P-uniform), if its *DCL* is accepted in time $O(\log(n))$ (resp. $n^{O(1)}$) by some deterministic Turing Machine.

Remark: This definition is a uniform version of one given in [BFS88], with some important modifications. We include conjugation gates and exclude multiple wires. Multiple wires are irrelevant for Σ-gates since

the characteristic is two, and multiple wires into a product gate can be replaced by a small circuit of depth 2 using conjugation gates in the first level and a single product gate in the second level. Hence, our definition and the definition in [BFS88] yield equivalent complexity measures up to polynomial size and constant depth. Our version allows a *DCL* with words of length $O(\log(n))$ for polynomial size circuits, whereas a multiplicity of up to $2^n - 1$ might require n bits to specify.

Definition 2.2

Arithmetic expressions.

The set of arithmetic expressions is defined inductively in terms of some infinite set I of indices.

- The basic expressions are:
  1. g, a constant that generates the field, $\mathbf{F}_{2^n} = \mathbf{F}_2(g)$.
  2. x_i, where $i \in I$, the ith argument (input) to f_n.
- If e and f are expressions, then we may form new composite expressions $\Sigma_{i=1}^{b} e$, $\Pi_{i=1}^{b} e$, $e + f$, $e \cdot f$ and e^{2^i}, where $i \in I$ and $b \in I \cup \mathrm{Z}[n]$.

For an expression e to be *well formed* it should have no free variables, i.e. $FV(e) = \emptyset$, where FV is defined inductively:

- $FV(g) = \emptyset$,
- $FV(x_i) = \{i\}$,
- $FV(\Sigma_{i=1}^{j} e) = FV(\Pi_{i=1}^{j} e) = (FV(e) \setminus \{i\}) \cup \{j\}$, where $j \in I$,
- $FV(\Sigma_{i=1}^{p(n)} e) = FV(\Pi_{i=1}^{p(n)} e) = FV(e) \setminus \{i\}$, where $p(n) \in \mathrm{Z}[n]$,
- $FV(e + f) = FV(e \cdot f) = FV(e) \cup FV(f)$,
- $FV(e^{2^i}) = FV(e) \cup \{i\}$.

A well formed expression describes a family of functions $\{f_n\}$ (one for each field).

The complexity class AE consists of those families of functions $f_n : \mathbf{F}_{2^n}^{p(n)} \to \mathbf{F}_{2^n}$ that are described by arithmetic expressions.

Example 2.1

The canonical field functions *norm* and *trace* are in AE, as are the constant functions 0 and 1:

- $N(x) = \Pi_{i=1}^{n} x^{2^i}$,
- $Tr(x) = \Sigma_{i=1}^{n} x^{2^i}$,
- $1 = N(g)$,
- $0 = 1 + 1 = g + g$.

Division is also expressible:

- $\frac{1}{x} = \Pi_{j=1}^{2^n-2} x = \Pi_{j=1}^{n-1} x^{2^j}$.

Lemma 2.1

Given a polynomial $p(n)$, there exists a family of constant functions f_n in AE and a constant n_0 such that the value of f_n is an element in F_{2^n} of order at least $p(n)$ provided $n > n_0$.

Proof:

Define the expression

$$A(i_1, i_2, ..., i_l) = g^{i_1} + g^{i_2} + ... + g^{i_l}$$

By varying $i_1, i_2, ..., i_l$ the expression A can denote at least $\begin{pmatrix} n \\ l \end{pmatrix}$ different elements, since $g, g^2, g^3, ..., g^n$ are linearly independent (this is a consequence of g being a generating element).

The number of elements in $\mathbf{F}_{2^n}$ that have order at most $p(n)$ is bounded by $\Sigma_{i=1}^{p(n)} \phi(i) \leq p(n)^2$, where ϕ denotes the totient function.

By choosing $l = 2 \cdot deg(p) + 1$, we will have $\begin{pmatrix} n \\ l \end{pmatrix} > p(n)^2$ for n sufficiently large, and some choice of parameters for A will give an element of order at least $p(n)$.

The following expression checks whether a possible choice of parameters leads to an element of sufficiently high order:

$$B(i_1, i_2, ..., i_l) = N(\prod_{m=1}^{p(n)} (A(i_1, i_2, ..., i_l)^m + 1)),$$

which equals 1 iff the order of $A(i_1, i_2, ..., i_l)$ is greater than $p(n)$.

There may be several elements of high order and we will construct an expression that chooses the lexicographic first string $i_1, i_2, ..., i_l$ that leads to an element of sufficiently high order.

$$\begin{aligned} & C(i_1, i_2, ..., i_l) \\ = & \begin{cases} \Pi_{j_1=1}^{i_1-1} \Pi_{j_2=1}^{n} \Pi_{j_3=1}^{n} \cdots \Pi_{j_l=1}^{n} & (1 + B(j_1, j_2, ..., j_l)) \\ \quad \cdot \Pi_{j_2=1}^{i_2-1} \Pi_{j_3=1}^{n} \cdots \Pi_{j_l=1}^{n} & (1 + B(i_1, j_2, ..., j_l)) \\ \vdots & \vdots \\ \quad \cdot \Pi_{j_l=1}^{i_l-1} & (1 + B(i_1, i_2, ..., j_l)) \end{cases} \\ = & \begin{cases} 0 & \text{some string } j_1, j_2, ..., j_l \ (<_{lex} i_1, i_2, ..., i_n) \text{ satisfies} \\ & \text{order}(A(j_1, j_2, ..., j_l)) > p(n) \\ 1 & \text{otherwise} \end{cases} \end{aligned}$$

We have used index bounds $i_1 - 1$ in the summation above. However that does not increase our expressive power:

$$\prod_{i=1}^{j-1} e = \prod_{i=1}^{j} (1 + (e-1)N(g^i + g^j)),$$

for $j \leq n$, since g has order at least $n + 1$.

Our complete expression for an element of high order is

$$f_n = \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} \cdots \sum_{i_l=1}^{n} C(i_1, i_2, ..., i_l) \cdot B(i_1, i_2, ..., i_l) \cdot A(i_1, i_2, ..., i_l)$$

□

Lemma 2.2

Identify bits with the field elements 0, 1. Then FO+Maj $\subseteq$ AE in the following sense. Let $f_n \in$ FO+Maj and let $p(n)$ be some polynomial. Define a function h_n on the field $\mathbf{F}_{2^n}$ by $h_n = f_{p(n)}$, i.e. h_n takes $p(n)$ arguments. Then $h_n \in$ AE.

Proof

We know h_n can be described by a logic expression, so it suffices to provide an inductive translation $\mathcal{E}$ from logic expressions to arithmetic expressions that preserves 0, 1 under the usual semantics. Below we will assume that b denotes a field element of order at least $2p(n)^2$ (by Lemma 2.1). Translations for the bits 0 and 1 were presented in Example 2.1.

1. $\mathcal{E}(\neg e) = 1 + \mathcal{E}(e)$
2. $\mathcal{E}(e_1 \wedge e_2) = \mathcal{E}(e_1)\mathcal{E}(e_2)$
3. $\mathcal{E}(\forall i.e) = \Pi_{i=1}^{p(n)} \mathcal{E}(e)$
4. $\mathcal{E}(i < j) = \Pi_{k=1}^{i} N(b^k + b^j)$, where k is not in $\{i, j\}$.
5. $\mathcal{E}(i = j) = 1 + N(b^i + b^j)$

   To specify the BIT-predicate, we first define translations for less complex logic expressions:

   $\mathcal{E}(i = j + k) = 1 + N(b^i + b^j \cdot b^k)$

   $\mathcal{E}(i = j \cdot k) = 1 + N(b^i + (b^j)^k)$

   $\mathcal{E}(\text{“}i \text{ is odd”}) = \mathcal{E}(\exists j.i = 2j + 1)$

   $\mathcal{E}(\text{“}i \text{ is power of 2”}) = \mathcal{E}(\forall j \forall k.(i = j \cdot k \wedge \text{“}j \text{ is odd”}) \rightarrow j = 1)$

   $$\begin{aligned}\mathcal{E}(i = 2^j) = \ & \mathcal{E}(\text{“}i \text{ is power of 2”})\cdot \\ & (1 + N(b \cdot b^j + \Pi_{k=1}^{i} b^{\mathcal{E}(\text{“}k \text{ is power of 2”})}))\end{aligned}$$

   where $b^x = 1 + x(b-1)$ for $x \in \{0, 1\}$

6. $\mathcal{E}(\mathtt{BIT}(i,j)) = \mathcal{E}(\exists k \exists l.(k < 2^i) \wedge (\text{“}l \text{ is odd”}) \wedge (j = k + l \cdot 2^i))$
7. $\mathcal{E}(Mi.e) = \Sigma_{j=1}^{p(n)}(1 + \mathcal{E}(2j < p(n))) \cdot (1 + N(b^j + \Pi_{i=1}^{p(n)} b^{\mathcal{E}(e)}))$

$\square$

Remark: The logic expression for BIT is taken from [BIS90].

Lemma 2.3

DLOGTIME $\subseteq$ AE, when bits are identified with the field elements 0, 1.

Proof:

DLOGTIME $\subseteq$ FO by fact 1.1, so the result is a consequence of the above lemma.

$\square$

Lemma 2.4

Let $\{f_n\}$ be computed by a DLOGTIME-uniform family of arithmetic circuits C_n of polynomially bounded size and constant depth. Then $\{f_n\}$ is in AE.

Proof:

The direct connection language L for $\{C_n\}$ is described by an arithmetic expression e_L according to lemma 2.3. Without loss of generality, we may assume that, for some constant k, each of the first 5 components in a 6-tuple in L takes exactly $\lceil \log(n^k) \rceil$ bits to specify. Define

$$\begin{aligned} e'_L(n,a,t,j,b) = \ & \text{original expression } e_L, \text{ with } x_i \text{ replaced by} \\ & \Sigma_{l=1}^{n^k} \mathcal{E}(2^{l-1} \le n^k < 2^l) \cdot (\\ & \mathcal{E}(i \le l \wedge \mathtt{BIT}(i,n)) \\ & + \mathcal{E}(l < i \le 2l \wedge (\exists j.j + l = i \wedge \mathtt{BIT}(j,a))) \\ & + \ldots + \mathcal{E}(4l < i \le 5l \wedge (\exists j.j + 4l = i \wedge \mathtt{BIT}(j,b))) \\ &) \end{aligned}$$

The AE macros below are constructed so that our final expression is simpler to read:

$$type(a,t) = 1 + \prod_{j=1}^{n^k} \prod_{b=1}^{n^k} (1 + e'_L(n,a,t,j,b))$$

$$index(a,j) = 1 + \prod_{t=1}^{n^k} \prod_{b=1}^{n^k} (1 + e'_L(n,a,t,j,b))$$

$$input(a,b) = 1 + \prod_{t=1}^{n^k} \prod_{j=1}^{n^k} (1 + e'_L(n,a,t,j,b))$$

Using these macros we can give a recursive expression for the value computed at a particular gate:

$$\begin{aligned} value(a) = {} & type(a, sum) \cdot \Sigma_{b=1}^{n^k} input(a,b) \cdot value(b) \\ + {} & type(a, product) \cdot \Pi_{b=1}^{n^k}(1 + (value(b)+1) \cdot input(a,b)) \\ + {} & type(a, conjugation) \cdot \\ & \quad \Sigma_{b=1}^{n^k} \Sigma_{j=1}^{n} (input(a,b) \cdot value(b))^{2^j} \cdot index(a,j) \\ + {} & type(a, input) \cdot \Sigma_{j=1}^{n^k} x_j \cdot index(a,j) \\ + {} & type(a, constant) \cdot g \end{aligned}$$

By convention the output is computed by gate number 0, so the arithmetic expression for f_n is $value(0)$, where the expression for $value$ has been substituted in recursively, but only a constant number of times, since the depth of the circuits C_n are known to be bounded by a constant.

□

Lemma 2.5

Every arithmetic function f_n in AE has DLOGTIME-uniform arithmetic circuits of polynomially bounded size and constant depth.

Proof:

We define an operator C_n inductively, that creates the n'th circuit, given an expression without free variables:

1. $C_n(e_1 \cdot e_2)$ = A Π-gate with two inputs, namely the outputs from $C_n(e_1)$ and $C_n(e_2)$.
2. $C_n(e_1 + e_2)$: This is similar.
3. $C_n(\Pi_{i=1}^{b} e)$ = A Π-gate with b inputs, one input from the output of each of the b circuits $C_n(e[i \to 1]), C_n(e[i \to 2]), ..., C_n(e[i \to b])$, where b is an integer that may depend on n.
4. $C_n(\Sigma_{i=1}^{b} e)$: This is similar.
5. $C_n(e^{2^j})$ = A conjugation gate with the appropriate integer parameter. Note that j is an integer, since expressions has no free variables.
6. $C_n(x_j)$ = A gate that connects up to the appropriate input. (As above, j is an integer).

7. $C_n(g)$ = A constant gate.

It may be necessary to add a polynomial number of dummy gates to satisfy the "no multiple wires" requirement, e.g. when forming circuits from the expression $\Pi_{i=1}^{n} x_1$.

The depth of the circuits $C_n(e)$ will be some constant c_1 dependent on e, and the maximal number of inputs to any gate will be n^{c_2}, hence we may assign gate numbers of c_1 blocks of $c_2\lceil\log(n)\rceil$ bits each to specify the path from the output gate (number 0) to the gate in question.

The log-time Turing Machine that recognises the DCL starts by finding the length n of its input (n, a, t, j, b, p) by binary search (see Definition 1.5), and checks that it starts with n. The expression e is stored in the finite control and allows the machine to check that a, b are legal gate numbers, that b is connected to a (the significant part of a is a prefix of b), that index j is correct (it is identical to one of the bit blocks of a) and that the type t is correct.

□

Theorem 2.1

The class of functions computed by DLOGTIME-uniform arithmetic circuits of polynomially bounded size and constant depth coincides with AE.

Proof:

This is immediate from Lemmas 2.4 and 2.5.

□

Remark: The theorem is not sensitive to the exact definition of AE. If arbitrary polynomial expressions in indices and n are allowed in upper and lower bounds of summations and products, then it is not difficult to see that the proof of lemma 2.5 still holds.

For exponential bounds the case is different. The power series $f_g(x) = \Sigma_{i=1}^{2^n-2} \frac{1}{1+g^i} x^i$ yields the last bit of the discrete logarithm of x ($\neq 0$) with respect to the the base g, when g is a primitive root in $\mathbf{F}_{2^n}$. It is an open (and presumably difficult) problem, whether f_g has polynomial size circuits.

SECTION 3

Representations of Finite Fields of Characteristic 2.

Fact 3.1

$\mathbf{F}_{2^n}$ is a vector space.

$\mathbf{F}_{2^n}$ can be regarded as an n-dimensional vector space over $\mathbf{F}_2$.

Let f_n be an irreducible polynomial over $\mathbf{F}_2$ of degree n. Then $\mathbf{F}_2[x]/(f_n)$ is isomorphic to $\mathbf{F}_{2^n}$. If g is a root of f_n then $1, g, g^2, g^3, ..., g^{n-1}$ are linearly independent and form a *polynomial* basis for $\mathbf{F}_{2^n}$ over $\mathbf{F}_2$.

If the elements $g, g^2, g^4, g^8, ..., g^{2^{n-1}}$ (all the conjugates of a single element $g \in \mathbf{F}_{2^n}$) are linearly independent then they form a *normal* basis for $\mathbf{F}_{2^n}$.

The trace function $Tr : \mathbf{F}_{2^n} \to \mathbf{F}_2$ is defined by $Tr(x) = \Sigma_{i=0}^{n-1} x^{2^i}$. Two bases $b_1, b_2, ..., b_n$ and $c_1, c_2, ..., c_n$ are *dual* (complementary, trace-orthogonal) if

$$Tr(b_i c_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

A basis is *self-dual* if it is its own dual.

Every finite field of characteristic 2 has a polynomial basis, a normal basis and a self-dual basis (and possibly many of each).

The dual basis to a normal basis is also normal and the field $\mathbf{F}_{2^n}$ has a both normal and self-dual basis precisely when $n \not\equiv 0 \bmod 4$

Remark: The last statement about the existence of a basis that is both normal and self-dual was only proved recently [LeWe88]. The remaining facts with historical notes may be found in [LiNi83].

The following definitions 3.1, 3.2, 3.3 and 3.4 are all adapted from non uniform versions in [BFS88], as are lemmas 3.1 and 3.2.

Definition 3.1

Representations.

A *representation* of $\mathbf{F}_{2^n}$ is a family of bijections $\phi_n : \mathbf{F}_{2^n} \to \{0,1\}^n$. It defines which element is associated with which bit string. In particular it

defines the representation of the generating element g used in definitions 2.1 and 2.2.

Remark: One might consider more general representations, e.g. involving redundancy [FrSt88]. For technical simplicity, we stick to the defined set of representations, though our results are valid in a more general setting.

Definition 3.2

Good representations

Let X $\in$ {P,DLOGTIME}. $\{\phi_n\}$ is a X-uniformly *good* representation if there exists X-uniform families of majority/negation circuits ($1 \leq j < n$) ($N \geq 2$)

$$S_{n,N} : \{0,1\}^{nN} \to \{0,1\}^n$$
$$P_{n,N} : \{0,1\}^{nN} \to \{0,1\}^n$$
$$C_n^j : \{0,1\}^n \to \{0,1\}^n$$
$$G_n : \epsilon \to \{0,1\}^n$$

all of constant depth, the first two of size $(nN)^{O(1)}$, the last two of size $n^{O(1)}$, satisfying

$$S_{n,N}(\phi_n(z_1), ..., \phi_n(z_N)) = \phi_n(\sum_{i=1}^{N} z_i)$$

$$P_{n,N}(\phi_n(z_1), ..., \phi_n(z_N)) = \phi_n(\prod_{i=1}^{N} z_i)$$

$$C_n^j(\phi_n(z_1)) = \phi_n(z_1^{2^j})$$

$$G_n = \phi_n(g)$$

for all $z_i \in \mathbf{F}_{2^n}$

Remark: Intuitively field arithmetic can be implemented efficiently in a good representation.

Definition 3.3

Strong Representations

Let X $\in$ {P,DLOGTIME}. $\{\phi_n\}$ is a X-uniformly *strong* representation, if there exist X-uniform constant depth, polynomially bounded size (in n)

arithmetic circuits

$$i_n : \mathbf{F}_{2^n}^n \to \mathbf{F}_{2^n}$$
$$o_n : \mathbf{F}_{2^n} \to \mathbf{F}_{2^n}^n$$

satisfying

$$o_n(z) = \phi_n(z)$$
$$i_n(\phi_n(z)) = z$$

for all field elements z. Here we have regarded $\{0,1\}$ both as Boolean values and as members of $\mathbf{F}_{2^n}$.

Remark: Intuitively a strong representation is one that can be accessed efficiently from within the field.

Definition 3.4

Equivalent Representations

Let $\mathtt{X} \in \{\mathtt{P},\mathtt{DLOGTIME}\}$. A representation $\phi_n : \mathbf{F}_{2^n} \to \{0,1\}^n$ *translates* X-uniformly into a representation $\theta_n : \mathbf{F}_{2^n} \to \{0,1\}^n$ (written $\phi_n \leq_{\mathtt{X}} \theta_n$) iff there exists a family of X-uniform constant depth, $n^{O(1)}$ size majority/negation circuits $T_n : \{0,1\}^n \to \{0,1\}^n$ satisfying $T_n(\phi_n(z)) = \theta_n(z)$ for all $z \in \mathbf{F}_{2^n}$. ϕ_n and θ_n are X-uniformly *equivalent* representations (written $\phi_n \equiv_{\mathtt{X}} \theta_n$) iff $\phi_n \leq_{\mathtt{X}} \theta_n$ and $\theta_n \leq_{\mathtt{X}} \phi_n$.

Remark: Two equivalent representations can be regarded as being one, since conversion between them is very fast.

Remark 3.1

In the following, we will several times assume that we can replace (simulate) all gates in a X-uniform family of arithmetic (majority/negation) circuits by small majority/negation (arithmetic) circuits taken from a X-uniform family, and as a result still have a X-uniform family of circuits.

We shall in each case assume that gate numbers in the new circuit are pairs (a, b), where a is a gate number in the original circuit, and b is a gate number in the small circuit that simulates gate number a. Furthermore, we construct a machine to accept the connection language of the new circuits by letting the finite control contain the cross product of all the finite controls in the machines that exist due to the assumed X-uniformity of the constituent circuit families.

Lemma 3.1

Let $\mathtt{X} \in \{\mathtt{P},\mathtt{DLOGTIME}\}$. If ϕ_n is X-uniformly good and ψ_n is X-uniformly strong then ϕ_n and ψ_n are X-uniformly equivalent.

Proof:

To prove the equivalence, we show the existence of translations both ways: The arithmetic circuits i_n (o_n) that exists due to ψ_n being strong may be simulated by majority/negation circuits via the representation ϕ_n by using the circuits $S_{n,M}$, $P_{n,M}$, C_n^j and G_n that exist due to ϕ_n being good. The simulation preserves uniformity by remark 3.1. To make the constructed circuits be a complete translation proving $\psi_n \leq \phi_n$ ($\phi_n \leq \psi_n$), we must preprocess the input string by attaching small circuits that when input a bit 0/1 outputs the ϕ_n representation of the field element 0/1 (postprocess the output string by attaching small circuits that outputs the bit 0/1 when input the ϕ_n representation of the field element 0/1). The pre- and post-processing can be done uniformly by using the expressions for 0 and 1 that are given in example 2.1 to generate circuits (lemma 2.5). For the post-processing we use in addition that the representation is a bijection.

□

Lemma 3.2

Let $\mathtt{X} \in \{\mathtt{P},\mathtt{DLOGTIME}\}$. If ϕ_n is X-uniformly good(strong) and ϕ_n and ψ_n are X-uniformly equivalent, then ψ_n is X-uniformly good(strong).

Proof:

We must prove that ψ_n is good. The proof is similar for the different kind of gates, so we will just consider how to implement summation. Since ϕ_n and ψ_n are equivalent, we can translate the ψ_n representation of the input elements into a ϕ_n representation of the same elements, and do the summation using that ϕ_n is good, followed by a translation back to the ψ_n representation. Composition of circuits preserves uniformity.

To prove "strong" version of the statement, we need to make similar considerations, with one complication introduced: The majority/negation circuits that exist for translation between the representations must be simulated by arithmetic. If the translation is known to be DLOGTIME-uniform then we can use fact 1.1 and lemmas 2.2, 2.5. In the general case, we may use lemma 2.5 and the proof of lemma 2.2 to obtain small arithmetic circuits to simulate single majority/negation gates and then call upon remark 3.1 to ensure that the resulting family of arithmetic circuits is X-uniform.

□

Definition 3.5

Polynomial, Normal and Self-dual Representations.

A family of bases (one base for each field) gives rise to a corresponding *basis* representation ϕ_n defined by $\phi_n(\sum_{i=1}^{n} a_i b_i) = (a_1, a_2, ..., a_n)$, where $a_i \in \{0,1\}$ and $b_1, b_2, ..., b_n$ is the basis for $\mathbf{F}_{2^n}$.

When all the bases are either polynomial, normal or self-dual then a *polynomial, normal* or *self-dual* basis representation is obtained. For a polynomial $(1, h, h^2, ..., h^{n-1})$ or normal $(h, h^2, h^4, ..., h^{2^{n-1}})$ representation, the constant g, is identified with the basis generator h: $\phi(g) = \phi(h)$.

Lemma 3.3

Construction of basis representations.

There is a deterministic algorithm that on input n in time $O(n^{4+\epsilon})$ returns a normal basis $g, g^2, ..., g^{2^{n-1}}$ (represented by an irreducible polynomial of degree n of which g is a root) for $\mathbf{F}_{2^n}$. If $n \neq 0 \bmod 4$ then the normal basis will be self-dual.

Proof:

Shoup has recently shown how to find an irreducible polynomial over $\mathbf{F}_2$ of specified degree n in time $O(n^{4+\epsilon})$ deterministically [Sho90].

Given the irreducible polynomial we may simulate arithmetic in the field via a polynomial basis. This allows to apply some other recent results:

Bach, Driscoll and Shallit show how to find a normal basis (expressed in terms of the polynomial basis) in time $O(n^4)$ deterministically [BDS90].

Lempel and Weinberger show how to find a self-dual normal basis in $\mathbf{F}_{2^n}$ (when one exists) given a normal basis [LeWe88]. They do not analyse the time efficiency of their algorithm. However, a simple inspection reveals that it may all be done in time $O(n^4)$.

We have now found g expressed in terms of the original polynomial basis. To find the irreducible polynomial satisfied by g, we simply compute the expression $(x-g)(x-g^2)\cdot ... \cdot(x-g^{2^{n-1}})$

□

Lemma 3.4

There exists a P-uniformly strong representation that is DLOGTIME-uniformly strong for the subclass of fields $\mathbf{F}_{2^n}$, where $n \neq 0 \bmod 4$.

Proof

Let ϕ_n be the normal representation corresponding to the normal bases that are generated by the algorithm in the proof of lemma 3.3.

We note that i_n (from Definition 3.3) is characterised by

$$i_n(a_0, a_1, ..., a_{n-1}) = \sum_{i=0}^{n-1} a_i g^{2^i}.$$

By lemma 2.5 it follows that i_n has appropriately uniform circuits.

The dual basis to $g, g^2, ..., g^{2^{n-1}}$ is also normal and let it be generated by h. In terms of h, we can characterise the j'th output of o_n (from Definition 3.3) as follows

$$o_n(a)_j = \sum_{i=0}^{n-1} Tr(g^{2^i}h^{2^j})a_i = Tr((\sum_{i=0}^{n-1} a_i g^{2^i})h^{2^j}) = Tr(a \cdot h^{2^j}).$$

The last part of the lemma now follows by noting that for $n \neq 0$ mod 4, $h = g$, and the use of lemma 2.5 on the above expression for o_n.

In the case $n = 0 \bmod 4$ we may compute h from g. The trace equation

$$Tr(g \cdot h^{2^i}) = \begin{cases} 1 & i = 0 \\ 0 & i = 1, .., n-1 \end{cases}$$

yields n linear equations with the n unknowns $h, h^2, ..., h^{2^{n-1}}$. So an expression for h may be found in polynomial time by using linear algebra. Intermediate expressions are reduced using the minimal polynomial for g, which is found in polynomial time by lemma 3.3.

□

Remark: There may well be a DLOGTIME-uniformly strong representation of all fields $\mathbf{F}_{2^n}$, but the non-existence of a both self-dual and normal basis for $\mathbf{F}_{2^n}$, when $n = 0 \bmod 4$, makes a different proof technique necessary. A self-dual basis exists in each of these fields, but it is not clear how to find one efficiently [Lem75, SeLe80]. A self-dual basis in the hand might be of no help anyway, since it could be difficult to express all basis elements $\{b_1, b_2, ..., b_n\}$ uniformly in terms of g.

A partial solution is known, however. If $\{a_1, ..., a_k\}$ and $\{b_1, ..., b_l\}$ are self-dual bases in $\mathbf{F}_{2^k}$ and $\mathbf{F}_{2^l}$ respectively, where $\gcd(k, l) = 1$, then $\{a_i b_j | i = 1, ..., k \text{ and } j = 1, ..., l\}$ is a self-dual basis for $\mathbf{F}_{2^{kl}}$. Hence, it suffices to construct self-dual bases for the fields $\mathbf{F}_{2^{2^n}}$, $n \geq 2$. This is of course possible for any specific of these fields, e.g. if h is a root in $x^4+x^3+x^2+x+1$ (i.e. h is of order 5) then $\{h, 1+h^2, h^4, 1+h^8\}$ is a self-dual basis in $\mathbf{F}_{2^4}$. Based on h, one may construct a DLOGTIME-uniformly strong representation for the fields $\mathbf{F}_{2^n}$, where $n = 4 \bmod 8$.

Lemma 3.5

There exists a P-uniformly good representation.

Proof:

Let ϕ_n be the polynomial representation defined by the basis $1, g, ..., g^{n-1}$ and let ψ_n be the normal representation defined by the basis $g, g^2, ..., g^{2^{n-1}}$, where g in both cases is the element found in the proof of lemma 3.3.

By lemma 3.3, we can find the irreducible polynomial f_n of g in polynomial time and hence we can also compute the matrices to convert between the two bases in that time showing that $\phi_n \equiv_{\mathrm{P}} \psi_n$.

We must show how to construct the circuits $S_{n,N}$, $P_{n,N}$, C_n^j and G_n in time $(nN)^{O(1)}$. Since the two representations are equivalent and by the proof of lemma 3.2, we need only construct each circuit for either ψ_n or ϕ_n in order to prove that both ϕ_n and ψ_n are P-uniformly good.

1. Constant gate G_n: This gate outputs $(1, 0, 0, ..., 0)$ in the normal representation.
2. Conjugation gate C_n^j: This gate makes a cyclic shift of the bits in the normal representation.

3. Sum gate $S_{n,N}$: This gate is identical in both representations. It uses n parity sub gates, each of fan-in N.
4. Product gate $P_{n,N}$: This gate is the most complicated to implement. We choose the polynomial representation.

   In the non uniform version [BFS88], the product polynomial is computed modulo a lot of low degree, pairwise prime, irreducible polynomials and reconstructed using the Chinese remainder theorem followed by a *modulo* f_n operation. Since polynomial arithmetic modulo a low degree irreducible polynomial represents arithmetic in a smaller characteristic two field, it is implemented by converting to discrete logarithms in this field and doing an addition of the resulting binary integers.

   This proof carries over to the uniform case without too great difficulty. The most important uniformisation details are:

   - Finding the necessary irreducible polynomials: To implement a product gate $P_{n,N}$ with N inputs, we need to form the product of N polynomials each of degree at most $n-1$. Hence if we form the product modulo some polynomial h of degree at least nN, we can recover the full product.

     We choose $h(x) = x^{2^{\lceil \log(nN) \rceil}} - x$. $h(x)$ factors completely over the field $\mathbf{F}_{2^{\lceil \log(nN) \rceil}}$. Hence over $\mathbf{F}_2$, all irreducible factors of $h(x)$ are pairwise prime and each have degree at most $\lceil \log(nN) \rceil$. All factors may be found in time $(nN)^{O(1)}$ by an exhaustive search using trial division.
   - Taking discrete logarithms (and anti-logarithms) in small fields: Since each of the small fields contain at most $2^{\lceil \log(nN) \rceil} < 2nN$ elements, we can afford to find a primitive element by exhaustive search and compute a complete table of the discrete logarithms (and antilogarithms) with respect to this primitive element. All of this can be done in time $(nN)^{O(1)}$
   - Adding binary integers: For each of the small fields the logarithms are added using the circuits from fact 1.1.
   - Applying the Chinese Remainder Theorem: If $r_1(x), r_2(x), ..., r_k(x)$ are the remainders modulo $h_1(x), h_2(x), ..., h_k(x)$ of $r(x)$ then $r(x) = \Sigma_{i=1}^k r_i(x) c_i(x) d_i(x)$ modulo $(h(x) = \Pi_{i=1}^k h_i(x))$,

where $c_i(x) = h(x)/h_i(x) \bmod h_i(x)$ and $d_i(x)$ is chosen such that $c_i(x)d_i(x) = 1 \bmod h_i(x)$. Clearly all of $c_i(x)$ and $d_i(x)$ can be computed in time $(Nn)^{O(1)}$.

□

Remark: It may be that the notion of "P-uniformity" in the statement of lemma 3.5 can be replaced by a more restrictive kind of uniformity, but the basic problem of finding an appropriate irreducible polynomial seems to require the power of P with present techniques [Sho90].

Theorem 3.1

Regard the elements of $\mathbf{F}_{2^n}$ as bit strings (by virtue of some P-uniformly good and strong representation that is DLOGTIME-uniformly strong for the subclass of fields determined by $n \not\equiv 0 \bmod 4$). Let $\{f_n\}$ be an n-input, n-output Boolean function and let $Z(n) \geq n$, $D(n) \geq 1$. Then

1. $\{f_n\}$ is computed by P-uniform majority/negation circuits of size $Z(n)^{O(1)}$ and depth $O(D(n))$ iff $\{f_n\}$ is computed by (single input and output) P-uniform arithmetic circuits of size $Z(n)^{O(1)}$ and depth $O(D(n))$.

2. For the subclass of fields $\mathbf{F}_{2^n}$ where $n \not\equiv 0 \bmod 4$ a stronger statement is valid:

   If $\{f_n\}$ is computed by DLOGTIME-uniform majority/negation circuits of size $Z(n)$ and depth $D(n)$ then $\{f_n\}$ is computed by (single input and output) DLOGTIME-uniform arithmetic circuits of size $Z(n)^{O(1)}$ and depth $O(D(n))$.

Proof:

By lemma 3.4 there exists a P-uniformly strong representation ϕ_n that is DLOGTIME-uniformly strong on the restricted class of fields.

By Lemmas 3.1, 3.2 and 3.5 we know that any such ϕ_n is also P-uniformly good. Hence it is possible to identify field elements and bit strings as assumed in the initial statement of the theorem.

Assume that C_n is a family of X-uniform (where X is P or DLOGTIME) n-input, n-output majority/negation circuits. By remark 3.1 and lemmas

2.2, 2.5, we obtain an equivalent X-uniform n-input, n-output family of arithmetic circuits. The latter circuits are transformed into circuits with a single input and a single output, by attaching the circuits i_n and o_n that exists due to ϕ_n being strong. This proves part 2 and half of part 1.

Assume that D_n is a family of P-uniform arithmetic circuits each with a single input and a single output. Using the circuits $S_{n,N}$, $P_{n,N}$, C_n^j and G_n that exists due to ϕ_n being good, we simulate the arithmetic circuits (using remark 3.1) to obtain n-input, n-output majority/negation circuits as stated in part 1.

□

Corollary 3.1

Regard the elements of $\mathbf{F}_{2^n}$ ($n \not\equiv 0 \bmod 4$) as bit strings (by virtue of some DLOGTIME-uniform strong representation). Then $\mathrm{TC}^0 \subseteq \mathrm{AE}$.

Proof:

This is a consequence of Theorems 2.1 and 3.1 (part 2).

□

References

[All89] Allender, E. W., P-Uniform Circuit Complexity. *Journal of the ACM,* **36** (1989), pp. 912-928.

[BDS90] Bach, E., Driscoll, J. and Shallit, J., Factor Refinement. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms,* SIAM 1990, pp. 201-211.

[BDG] Balcázar, J. L., Díaz, J. and Gabarró, J., *Structural Complexity.* Springer Verlag, 1988 (Vol. 1), 1990 (Vol. 2).

[BIS90] Barrington, D. A. M., Immerman, N. and Straubing, H., On Uniformity within NC^1. *Journal of Computer and System Sciences,* **41** (1990), pp. 274-306.

[Bor77] Borodin, A., On Relating Time and Space to Size and Depth. *SIAM Journal on Computing,* **6** (1977), pp. 733-744.

[BFS88] Boyar, J., Frandsen, G. S. and Sturtivant, C., An Arithmetic Model of Computation Equivalent to Threshold Circuits. To appear in *Theoretical Computer Science.* Early version as *Technical Report, DAIMI PB-239,* Aarhus University, 1988.

[CSV84] Chandra, A. K., Stockmeyer, L. and Vishkin, U., Constant Depth Reducibility. *SIAM Journal on Computing,* **13** (1984), pp. 423-439.

[FrSt88] Frandsen, G. S. and Sturtivant, C., The Depth Efficacy of Unbounded Characteristic Finite Field Arithmetic. *Technical Report, DAIMI PB-240,* Aarhus University, 1988.

[FSS84] Furst, M., Saxe, J. B. and Sipser, M., Parity, Circuits and the Polynomial Time Hierarchy. *Mathematical Systems Theory,* **17** (1984), pp. 260-270.

[Imm87] Immerman, N., Languages that Capture Complexity Classes. *SIAM Journal on Computing,* **16** (1987), pp. 760-778.

[Imm89] Immerman, N., Expressibility and Parallel Complexity. *SIAM Journal on Computing,* **18** (1989), pp. 625-638.

[Lem75] Lempel,A., Matrix Factorization over GF(2) and Trace-Orthogonal Bases of GF(2^n). *SIAM Journal on Computing,* **4** (1975), pp. 175-186.

[LeWe88] Lempel, A. and Weinberger, M. J., Self-Complementary Normal Bases in Finite Fields. *SIAM Journal on Disrete Mathematics,* **1** (1988), pp. 193-198.

[LiNi83] Lidl, R. and Niederreiter, H., *Finite Fields.* Encyclopedia of Mathematics and its Applications **20**. Addison Wesley, 1983.

[Mur71] Muroga, S., *Threshold Logic.* Academic Press, New York, 1971.

[MTT60] Muroga, S., Toda, I. and Takasu, S., Theory of Majority Decision Elements. *J. Franklin Inst.,* **271** (May 1961), pp. 376-418. Originally in *J. Inst. Electron. Comm. Eng., Japan,* Vol **43**, nos 10 and 12 (Oct. and Dec 1960). (In Japanese).

[Ruz81] Ruzzo, W. L., On Uniform Circuit Complexity. *Journal of Computer and System Sciences,* **22** (1981), pp. 365-383.

[SeLe80] Seroussi, G. and Lempel, A., Factorization of Symmetric Matrices and Trace-Orthogonal Bases in Finite Fields. *SIAM Journal on Computing,* **9** (1980), pp. 758-767.

[Sho90] Shoup, V., New Algorithms for finding Irreducible Polynomials over Finite Fields. *Mathematics of Computation,* **54** (1990), pp. 435-447.

[SkVa85] Skyum, S. and Valiant, L. G., A Complexity Theory Based on Boolean Algebra. *Journal of the ACM,* **32** (1985), pp. 484-502.