Interpolating Arithmetic Read-Once Formulas in Parallel

Nader H. Bshouty* **Richard Cleve***
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada T2N 1N4

Abstract

A formula is read-once if each variable appears at most once in it. An arithmetic read-once formula is one in which the operations are addition, subtraction, multiplication, and division (and constants are allowed). We present a randomized (Las Vegas) parallel algorithm for the exact interpolation of arithmetic read-once formulas over sufficiently large fields. More specifically, for n-variable read-once formulas, and fields of size at least $3(n^2+3n-2)$, our algorithm runs in $O(\log^2 n)$ parallel steps using $O(n^4)$ processors (where the field operations are charged unit cost). This complements other results which imply that other classes of read-once formulas *cannot* be interpolated—or even learned with membership and equivalence queries—in poly-logarithmic-time with polynomially many processors (even though they can be learned sequentially in polynomial-time). These classes include boolean read-once formulas and arithmetic read-once formulas over fields of size $o(n/\log n)$ (for n variable read-once formulas).

1 Introduction

The problem of *interpolating* a formula (from some class C) is the problem of exactly identifying the formula from queries to the assignment (membership) oracle. The interpolation algorithm queries the oracle with an assignment a and the oracle returns the value of the function at a.

There are a number of classes of arithmetic formulas that can be interpolated sequentially in polynomial-time as well as in parallel in poly-logarithmic-time (with polynomially many processors). These include sparse polynomials and sparse rational functions ([BT88,BT90,GKS90,GrKS88,RB89,M91]).

*Research supported in part by NSERC of Canada. Author's E-mail addresses: bshouty@cpsc.ucalgary.ca and cleve@cpsc.ucalgary.ca.

A formula over a variable set V is *read-once* if each variable appears at most once in it. An *arithmetic read-once* formula over a field $\mathcal{K}$ is a read-once formula over the basic operations of the field $\mathcal{K}$: addition, subtraction, multiplication, division, and constants are also permited in the formula.

Bshouty, Hancock and Hellerstein [BHH92] present a randomized sequential polynomial-time algorithm for interpolating arithmetic read-once formulas (AROFs) over sufficiently large fields. Moreover, they show that, for arbitrarily-sized fields, arithmetic read-once formulas can be *learned* using equivalence queries in addition to membership queries.

The question of whether arithmetic read-once formulas can be interpolated (or learned) quickly in parallel depends on the size of the underlying field. It is shown in [BHH92] that for arithmetic read-once formulas over fields with $o(n/\log n)$ elements there is *no* poly-logarithmic-time algorithm that uses polynomially many processors (for interpolating as well as learning). Also, a similar negative result holds for boolean read-once formulas.

We present a (Las Vegas) parallel algorithm for the exact interpolation of arithmetic read-once formulas over sufficiently large fields. For fields of size at least $3(n^2 + 3n - 2)$, the algorithm runs in $O(\log^2 n)$ parallel steps using $O(n^4)$ processors (where the field operations are charged unit cost).

If the "obvious" parallizations are made to the interpolating algorithm in [BHH92] (i.e., parallelizations of independent parts of the computation) one obtains a parallel running time that is $\Theta(d)$, where d is the depth of the target formula. Since, in general, d can be as large as $\Theta(n)$, this does not result in significant speedup. Our parallel algorithm uses some techniques from the sequential algorithm of [BHH92] as well as some new techniques that enable nonlocal features of the AROF to be determined in poly-logarithmic-time.

The parallel algorithm can be implemented on an oracle EREW PRAM that initially selects some random input values (uniformly and independently distributed) and then performs $O(n^3)$ membership queries (via its oracle).

2 Identification with queries

The learning criterion we consider is *exact identification*. There is a formula f called the *target formula*, which is a member of a class of formulas C defined over the variable set V. The goal of the learning algorithm is to halt and output a formula h from C that is equivalent to f.

In a *membership query*, the learning algorithm supplies values $(x_1^{(0)},\ldots,x_n^{(0)})$ for the variables in V As input to a *membership oracle*, and receives in return the value of $f(x_1^{(0)},\ldots,x_n^{(0)})$. Note that if f' is a projection of f, it is possible to simulate a membership oracle for f' using a membership oracle for f.

We say that the class C is learnable in polynomial time if there is an algorithm that uses the membership oracle and interpolates any $f \in C$ in polynomial time in the number of variables n and the size of f. We say that C is efficiently learnable in parallel if there is a parallel algorithm that uses the membership oracle and interpolates any $f \in C$ in polylogarithmic time with polynomial number of processors. In the parallel computation p processors can ask p membership queries in one step.

3 Preliminaries

A *formula* is a rooted tree whose leaves are labeled with variables or constants from some domain, and whose internal nodes, or *gates*, are labeled with elements from a set of *basis* functions over that domain. A *read-once formula* is a formula for which no variable appears on two different leaves. An *arithmetic read-once formula* over a field $\mathcal{K}$ is a read-once formula over the basis of addition, subtraction, multiplication, and division of field elements, whose leaves are labeled with variables or constants from $\mathcal{K}$.

In [BHH92] it is shown that a modified basis can be used to represent any arithmetic read-once formula. Let $\mathcal{K}$ be an arbitrary field. The modified basis for arithmetic read-once formulas over $\mathcal{K}$ includes only two non-unary functions, addition $(+)$ and multiplication $(\times)$. The unary functions in the basis are $(ax+b)/(cx+d)$ for every $a, b, c, d \in \mathcal{K}$ such that $ad - bc \neq 0$. This requirement is to prevent $ax+b$ and $cx+d$ being identically 0 or differing by just a constant factor. We can also assume that non-constant formulas over this modified basis do not contain constants in their leaves. We represent such a unary function as f_A, where

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

The restriction on a, b, c, and d is equivalent to saying the determinant of A (denoted $\det(A)$) is non-zero.

The value of a read-once formula on an assignment to its variables is determined by evaluating the formula bottom up. This raises the issue of division by zero. In [BHH92] this problem is handled by defining basis functions over the extended domain $\mathcal{K} \cup \{\infty, \text{ERROR}\}$, where ∞ represents $1/0$ and ERROR represents $0/0$. For the special values we define our basis function as follows (assume $x \in \mathcal{K} - \{0\}$, $y \in \mathcal{K} \cup \{\infty, \text{ERROR}\}$, and A is as above).

$$y + \text{ERROR} = y \times \text{ERROR} = f_A(\text{ERROR}) = \text{ERROR}$$
$$x + \infty = x \times \infty = \infty$$
$$0 + \infty = \infty \times \infty = \infty$$
$$0 \times \infty = \infty + \infty = \text{ERROR}$$
$$f_A(\infty) = \begin{cases} \frac{a}{c} & c \neq 0 \\ \infty & c = 0 \end{cases} \quad \text{and} \quad f_A\left(\frac{-d}{c}\right) = \infty \quad \text{if} \quad c \neq 0.$$

It is shown in [BHH92] that these definitions are designed so that the output of the read-once formula is the same as it would be if the formula were first expanded and simplified to be in the form $p(x_1, \ldots, x_n)/q(x_1, \ldots, x_n)$ for some polynomials p and q where $\gcd(p, q) = 1$, and then evaluated.

We say that a formula f is *defined* on the variable set V if all variables appearing in f are members of V. Let $V = \{x_1, \ldots, x_n\}$. We say a formula f *depends* on variable x_i if there are values $x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}$ and $x_i^{(1)}$ in $\mathcal{K}$ for which

$$f(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}) \quad \neq \quad f(x_1^{(0)}, \ldots, x_{i-1}^{(0)}, x_i^{(1)}, x_{i+1}^{(0)}, \ldots, x_n^{(0)})$$

and for which both those values of f are not ERROR. We call such an input vector $v = (x_1^{(0)}, \ldots, x_n^{(0)})$ a *justifying assignment* for x_i.

Between any two gates or leaves α and β in an AROF, the relationships *ancestor*, *descendant*, *parent*, and *child* refer to their relative position in the rooted tree. Let $\alpha \leq \beta$ denote that α is a descendant of β (or, equivalently, that β is an ancestor of α). Let $\alpha < \beta$ denote that α is a *proper* descendant of β (i.e., $\alpha \leq \beta$ but $\alpha \neq \beta$). For any pair of variables x_i and x_j that appear in a read-once formula, there is a unique node farthest from the root that is an ancestor of both x_i and x_j, called their *lowest common ancestor*, which we write as $\mathrm{lca}(x_i, x_j)$. We shall refer to the *type* of $\mathrm{lca}(x_i, x_j)$ to mean the basis function computed at that gate. We say that a set W of variables has a common lca if there is a single node that is the lca of every pair of variables in W.

We define the *skeleton* of a formula f to be the tree obtained by deleting any unary gates in f (i.e. the skeleton describes the parenthesization of an expression with the binary operations, but not the actual unary operations or embedded constants).

We now list a basic property of unary functions f_A that is proved in [BHH92].

Property 1

1. *The function f_A is a bijection from $\mathcal{K} \cup \{\infty\}$ to $\mathcal{K} \cup \{\infty\}$ if and only if* $\det(A) \neq 0$. *Otherwise, f_A is either a constant value from $\mathcal{K} \cup \{\infty, ERROR\}$ or else is a constant value from $\mathcal{K} \cup \{\infty\}$, except on one input value on which it is ERROR.*

2. *The functions f_A and $f_{\lambda A}$ are equivalent for any $\lambda \neq 0$.*

3. *Given any three distinct points $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ and $p_3 = (x_3, y_3)$,*

   (a) *If p_1, p_2, p_3 are on a line then there exists a unique function f_A with $f_A(x) = ax + b$ that satisfies $f_A(x_1) = y_1$, $f_A(x_2) = y_2$, and $f_A(x_3) = y_3$.*

   (b) *If p_1, p_2, p_3 are not on a line then there exists a unique function f_A with* $\det(A) \neq 0$ *that satisfies $f_A(x_1) = y_1$, $f_A(x_2) = y_2$, and $f_A(x_3) = y_3$.*

4. *If functions f_A and f_B are equivalent and* $\det(A), \det(B) \neq 0$, *then there is a constant λ for which $\lambda A = B$.*

5. *The functions $(f_A \circ f_B)$ and f_{AB} are equivalent.*

6. *If* $\det(A) \neq 0$*, functions* f_A^{-1} *and* $f_{A^{-1}}$ *are equivalent.*

7. $f_A(\lambda x) = f_{A\begin{pmatrix} \lambda & 0 \\ 0 & 1 \end{pmatrix}}(x)$ *and* $f_A(\lambda + x) = f_{A\begin{pmatrix} 1 & \lambda \\ 0 & 1 \end{pmatrix}}(x)$.

   $\lambda f_A(x) = f_{\begin{pmatrix} \lambda & 0 \\ 0 & 1 \end{pmatrix} A}(x)$ *and* $\lambda + f_A(x) = f_{\begin{pmatrix} 1 & \lambda \\ 0 & 1 \end{pmatrix} A}(x)$.

4 Collapsability of Operations

Whenever two non-unary gates of the same type in an AROF are separated by only a unary gate it may be possible to collapse them together to a single non-unary gate of the same type with higher arity. For $\star \in \{+, \times\}$, a unary operation f_A is called *$\star$-collapsible* if

$$f_A(x \star y) \star z \equiv f_B(x) \star f_C(y) \star z,$$

for some unary operations f_B and f_C. Intuitively, the above property means that if the f_A gate occurs between two non-unary $\star$ gates then the two $\star$ gates can be "collapsed" into a single $\star$ gate of higher arity, provided that new unary gates can be applied to the inputs.

In [BHH92] it is explained that a unary gate f_A is $\times$-collapsible if and only if A is of the form

$$\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 0 & a \\ b & 0 \end{pmatrix},$$

and $+$-collapsible if and only if A is of the form

$$\begin{pmatrix} a & b \\ 0 & c \end{pmatrix}.$$

The following are equivalent definitions of $\star$-collapsible that will be used in this paper.

Property 2 *The following are equivalent*

1. f_A *is* $+$*-collapsible.*

2. $f_A(x) = \alpha x + \beta$ *for some* $\alpha, \beta \in \mathcal{K}$ *and* $\alpha \neq 0$.

3. $f_A(\infty) = \infty$.

The following are equivalent

1. f_A *is* $\times$*-collapsible.*

2. $f_A(x) = \alpha x^\beta$ *for some* $\alpha \in \mathcal{K}$ *and* $\beta \in \{1, -1\}$.

3. $\{f_A(\infty), f_A(0)\} = \{0, \infty\}$.

Proof: We prove the property by showing that $1 \Leftrightarrow 2 \Leftrightarrow 3$. If f_A is $+$-collapsible then

$$A = \begin{pmatrix} a & b \\ 0 & c \end{pmatrix}$$

and therefore $f(x) = (a/c)x + (b/c)$. Since A is nonsingular $a \neq 0$ and $c \neq 0$ and $a/c \neq 0$. $2 \Rightarrow 1$ is obvious. If $f_A(x) = \alpha x + \beta$ for some $\alpha, \beta \in \mathcal{K}$ and $\alpha \neq 0$ then $f_A(\infty) = \alpha\infty + \beta = \infty$. If $f_A(\infty) = \infty$ then since $f_{\left(\begin{smallmatrix} a & b \\ d & c \end{smallmatrix}\right)}(\infty) = a/d = \infty$ we must have $d = 0$.

The result for $\times$-collapsible is left for the reader.□

In [BHH92], a *three-way justifying assignment* is defined as an assignment of constant values to all but three variables in an AROF such that the resulting formula depends on all of the three remaining variables. For the present results, we require assignments that meet additional requirements, which are defined below.

For any two gates, α and β, with $\alpha < \beta$, define the *α–β path* as the sequence of gate operations along the path in the tree from α to β. Define a *non-collapsing* three-way justifying assignment as a three-way justifying assignment with the following additional property. For the unassigned variables x, y, and z, if $\text{lca}(x, y) < \text{lca}(x, z)$ and all non-unary operations in the $\text{lca}(x, y)$–$\text{lca}(x, z)$ path are of the same type $\star$ (for some $\star \in \{+, \times\}$) then the function that results from the justifying assignment is of the form

$$f_E(f_C(f_A(x) \star f_B(y)) \star f_D(z)),$$

for some unary operations f_A, f_B, f_C, f_D and f_E, where f_C is not $\star$-collapsible. Intuitively, this means that, after the justifying assignment, the two gates, $\text{lca}(x, y)$ and $\text{lca}(x, z)$, cannot be collapsed—and thus the relationship $\text{lca}(x, y) < \text{lca}(x, z)$ can still be detected in the resulting function.

Now, define a *total* non-collapsing three-way justifying assignment as a *single* assignment of constant values to all variables in an AROF such that, *for any three variables*, if all but those three are assigned to their respective constants then the resulting assignment is non-collapsing three-way justifying.

5 Parallel Learning Algorithm

In this section, we present a parallel algorithm for learning AROFs. The algorithm has three principal components: finding a total non-collapsing three-way justifying assignment; determining the skeleton of the AROF; and, determining the unary gates of the AROF.

The basic idea is to first construct a graph (that will later be referred to as the LCAH graph) that contains information about the relative positions of the lcas of all pairs of variables. This cannot be obtained quickly in parallel from justifying assignments, because of the possibility that some of the important structure of an AROF "collapses" under any given justifying assignment. However, we shall see that any total non-collapsing justifying assignment is sufficient to determine the entire structure of the AROF at once (modulo some polylog processing).

Once the LCAH graph has been constructed, the skeleton of the AROF can be constructed by discarding some of the structure of the LCAH graph (a "garbage collection" step). This is accomplished using some simple graph algorithms, as well as a parallel prefix sum computation (which is NC^1 computable [LF80]).

Finally, once that skeleton is determined, the unary gates can be determined by a recursive tree contraction method (using results from [B74]).

5.1 Finding a Total Non-Collapsing Three-Way Justifying Assignment

In [BHH92], it is proven that, for any triple of variables x, y and z, by drawing random values (independently) from a sufficiently large field, and assigning them to the other variables in an AROF, a three-way justifying assignment for those variables is obtained with high probability. In the parallel algorithm, a three-way justifying assignment that is *total non-collapsing* is required. We show that, if the size of the field $\mathcal{K}$ is at least $O(n^2)$ then the same randomized procedure also yields a total non-collapsing three-way justifying assignment with probability at least $\frac{1}{2}$. Therefore in time $O(1)$ this step can be implemented.

We shall begin with some preliminary lemmas and then the precise statement that we require will appear in Corollary 4.

Lemma 1: *If $g(y,z) = f_B(f_A(y) \star z)$, where f_A is not $\star$-collapsing then there exists at most one value $z^{(0)}$ for z such that $f_C(y) \equiv g(y, z^{(0)})$ is $\star$-collapsing.*

Proof: Let $\star = +$. If $f_B(f_A(y) + z_0)$ is $+$-collapsible then by property 2 we have

$$f_B(f_A(y) + z_0) = \alpha y + \beta,$$

where $\alpha \in \mathcal{K}\backslash\{0\}$ and $\beta \in \mathcal{K}$. We substitute $y = \infty$ and get

$$f_B(f_A(\infty) + z_0) = \infty.$$

Since f_A is not $+$-collapsible, by property 2, we have $f_A(\infty) = \gamma \neq \infty$. Solving the above system using property 1 we get

$$z_0 = f_{B^{-1}}(\infty) - \gamma.$$

This shows that there is at most one value of z that makes $f_B(f_A(y) + z)$ $+$-collapsible.

Let $\star = \times$. If $f_B(f_A(y)z_0)$ is $\times$-collapsible then by property 2 we have

$$f_B(f_A(y) + z_0) = \alpha y^{\beta},$$

where $\alpha \in \mathcal{K}\backslash\{0\}$ and $\beta \in \{+1, -1\}$. We substitute $y = 0, \infty$ and get

$$\{f_B(f_A(0)z_0), f_B(f_A(\infty)z_0)\} = \{0, \infty\}.$$

Since f_A is not $\times$-collapsible, by property 2, we have either $f_A(0)$ or $f_A(\infty)$ is not in $\{0,\infty\}$. Suppose $f_A(0) \notin \{0,\infty\}$ and suppose $f_B(f_A(0)z_0) = 0$ (the other cases are similar). Solving this gives

$$z_0 = f_{B^{-1}}(0)/f_A(0).$$

This shows that there is at most one value of z that makes $f_B(f_A(y)z)$ $\times$-collapsible.$\Box$

Lemma 2: *Let $F(x_1,\ldots,x_n)$ be an AROF with $lca(x_1,x_2) < lca(x_1,x_3)$ and suppose that all non-unary operations in the $lca(x_1,x_2) - lca(x_1,x_3)$ path are of the same type $\star \in \{+,\times\}$. Let $x_4^{(0)},\ldots,x_n^{(0)}$ be independently uniformly randomly chosen from $S \subseteq \mathcal{K}$, where $|S| = m$. Then the probability that $x_4^{(0)},\ldots,x_n^{(0)}$ is a non-collapsing three-way justifying assignment is at least $1 - \left(\frac{3n+1}{m}\right)$.*

Proof: Note that $x_4^{(0)},\ldots,x_n^{(0)}$ is *not* a non-collapsing three-way justifying assignment if and only if it is not a justifying assignment or there exists a path between the lcas of x_1, x_2 and x_3 all non-unary operations are of the same type and the path collapses under the assignment. From [BHH92], the probability of the former condition is at most $\frac{2n+4}{m}$. We need to bound the probability of the latter condition.

We have that $F(x_1,\ldots,x_n)$ is of the form

$$E(f_{H_k}(\cdots f_{H_1}(f_{H_0}(A(x_1)\star B(x_2))\star C_1)\cdots)\star C_k)\star D(x_3)),$$

where $A(x_1), B(x_2), C_1,\ldots,C_k, D(x_3), E(y)$ may depend on variables from $x_4,\ldots,x_n$ in addition to their marked arguments. Let $\bar{A}(x_1), \bar{B}(x_1), \bar{C}_1, \ldots, \bar{C}_k, \bar{D}(x_3), \bar{E}(y)$ denote the above formulas (respectively) with $x_4^{(0)},\ldots,x_n^{(0)}$ substituted for the variables $x_4,\ldots,x_n$. Also, let $d_1,\ldots,d_k$ denote the degrees of $C_1,\ldots,C_k$ (respectively), as functions of $x_4,\ldots,x_n$. By the assumption that F is in normal form, f_{H_0} is not $\star$-collapsing. Therefore, by Lemma 1, there exists at most one value of C_1 for which $f_{H_1}(f_{H_0}(y)\star C_1)$ is $\star$-collapsing. Since the degree of C_1 is d_1, the probability of this value occurring C_1 is at most d_1/m by Schwartz's result in [Sch80].

Similarly, if $f_{H_1}(f_{H_0}(y)\star C_1)$ is not $\star$-collapsing then Lemma 1 implies that there exists at most one value of C_2 for which $f_{H_2}(f_{H_1}(f_{H_0}(y)\star \bar{C}_1)\star C_2)$ is $\star$-collapsing, which occurs with probability at most d_2/m, and so on. It follows that the probability that

$$f_{H_k}(f_{H_{k-1}}(\cdots f_{H_1}(f_{H_0}(y)\star \bar{C}_1)\cdots)\star \bar{C}_k)$$

is $\star$-collapsing is at most $(d_1+\cdots+d_k)\frac{1}{m} \le \frac{n-3}{m}$.

The result now follows by summing the two bounds. $\Box$

Theorem 3: *Let $F(x_1,\ldots,x_n)$ be an AROF over $\mathcal{K}$, and $x_1^{(0)},\ldots,x_n^{(0)}$ be chosen uniformly from a set $S \subseteq \mathcal{K}$ with $|S| = m$. Then the probability that $x_1^{(0)},\ldots,x_n^{(0)}$ is a total non-collapsing three-way justifying assignment is at least $1 - \frac{6n^2}{m}$.*

Proof: First, note that, from Lemma 2, we can immediately infer that if $x_1^{(0)},\ldots,x_n^{(0)}$ are drawn independently uniformly randomly from $S \subseteq \mathcal{K}$, where

$|S| = m$ then the probability that $x_1^{(0)}, \ldots, x_n^{(0)}$ is a non-collapsing three-way justifying assignment is at most $\binom{n}{3}\left(\frac{n+1}{2m}\right) = O\left(\frac{n^4}{m}\right)$.

To obtain a better bound, consider each subformula C_i that is an input to some non-unary gate in the AROF. By results in [BHH92], there are at most two possible values of C_i that will result in some triple of variables with respect to which the the assignment is not three-way justifying (the values are 0 and ∞). Thus, as in the proof of Lemma 2, the probability of one of these values arising for C_i is at most $\frac{2d}{m}$, where d is the degree of C_i. Also, from Lemma 2, there is at most one value of C_i that will result in a collapsing assignment, and the probability of this arising is at most $\frac{d}{m}$. Thus, the probability of one of the two events above arising is at most $\frac{3d}{m}$, and, since $d \leq n$, this is at most $\frac{3n}{m}$.

Since there are at most $2n$ such subformulas C_i, the probability of any one of them attaining one of the above values is at most $\frac{6n^2}{m}$. $\square$

The constant in the proof of theorem 3 can be improved to obtain probability of

$$1 - \frac{\frac{3}{2}(n^2 + 3n - 2)}{m}.$$

by using the following observation. Notice that we upper bounded the degree of each subtree by n. In fact we can upper bound the degree of the leaves (there are n leaves) by degree 1 since they are variables. Then we have another $n-1$ internal subformulas of degrees $d_1 < d_2 < \cdots < d_{n-1}$. It is easy to show that $d_i \leq i+1$ (simple induction on the number of nodes). Taking all this into account we obtain the above bound.

By setting $m \geq 3(n^2 + 3n - 2)$, we obtain the following.

Corollary 4: *Let $F(x_1, \ldots, x_n)$ be an AROF over $\mathcal{K}$, and $x_1^{(0)}, \ldots, x_n^{(0)}$ be chosen uniformly from a set $S \subseteq \mathcal{K}$ with $|S| = 3(n^2 + 3n - 2)$. Then the probability that $x_1^{(0)}, \ldots, x_n^{(0)}$ is a total non-collapsing three-way justifying assignment is least $\frac{1}{2}$.*

This Corollary implies that the time complexity of finding an total non-collapsing three-way justifying assignment is $O(1)$.

5.2 Determining the Skeleton of a Read-Once Formula in Parallel

In this section, we assume that a total non-collapsing three-way justifying assignment is given and show how to construct the skeleton with $O(n^3)$ membership queries in one parallel step followed by an $O(\log n)$ steps of computation.

Firstly, suppose that, for a triple of variables x, y, and z, we wish to test whether or not $\mathrm{lca}(x, y) < \mathrm{lca}(x, z)$. If $\mathrm{op}(x, y) \neq \mathrm{op}(x, z)$ then this can be accomplished by a direct application of the techniques in [BHH92], using the fact that we have an assignment that is justifying with respect to variables x, y, and z. On the other hand, if $\mathrm{op}(x, y) = \mathrm{op}(x, z)$ then $\mathrm{lca}(x, y) < \mathrm{lca}(x, z)$ could be difficult to detect with a mere justifying assignment because the justifying assignment might collapse the relative structure between these three variables. If all the non-unary operations in the $\mathrm{lca}(x, y)$–$\mathrm{lca}(x, z)$ path are identical then, due to the fact that we have a non-collapsing justifying assignment, we are guaranteed that the substructure between the three variables does not collapse, and we can determine that $\mathrm{lca}(x, y) < \mathrm{lca}(x, z)$ in $O(1)$ time (again by directly applying techniques in [BHH92]). The leaves the case where $\mathrm{op}(x, y) = \mathrm{op}(x, z)$ but the non-unary operations in the $\mathrm{lca}(x, y)$–$\mathrm{lca}(x, z)$ path are *not* all of the same type. In this case, the techniques of [BHH92] might fail to determine that $\mathrm{lca}(x, y) < \mathrm{lca}(x, z)$ and report them as equal. We shall overcome this problem at a later stage in our learning algorithm, by making inferences based on hierarcical relationships with other variables. For the time being, we can, in time $O(1)$ with one processor, compute the following.

$$\mathbf{DESCENDANT}(x,y,z) = \begin{cases} \text{YES} & \text{if } \mathrm{lca}(x,y) < \mathrm{lca}(x,z) \text{ and } \mathrm{op}(x,y) \neq \mathrm{op}(x,z); \\ \text{YES} & \text{if } \mathrm{lca}(x,y) < \mathrm{lca}(x,z) \text{ and all non-unary operations} \\ & \text{in the } \mathrm{lca}(x,y)\text{–}\mathrm{lca}(x,z) \text{ path are of the same type;} \\ \text{YES or MAYBE} & \text{if } \mathrm{lca}(x,y) < \mathrm{lca}(x,z) \text{ and } \mathrm{op}(x,y) = \mathrm{op}(x,z) \text{ but } \textit{not} \\ & \text{all non-unary operations in the } \mathrm{lca}(x,y)\text{–}\mathrm{lca}(x,z) \\ & \text{path are of the same type;} \\ \text{MAYBE} & \text{otherwise.} \end{cases}$$

Note that if $\mathbf{DESCENDANT}(x, y, z) = \text{YES}$ then it must be that $\mathrm{lca}(x, y) < \mathrm{lca}(x, z)$; however, if $\mathbf{DESCENDANT}(x, y, z) = \text{MAYBE}$ then it is possible that $\mathrm{lca}(x, y) < \mathrm{lca}(x, z)$, but $\mathrm{op}(x, y) = \mathrm{op}(x, z)$ and the non-unary operations on the $\mathrm{lca}(x, y)$–$\mathrm{lca}(x, z)$ are not of the same type, or that $\mathrm{lca}(x, y) \not< \mathrm{lca}(x, z)$.

To construct the extended skeleton of an AROF, we first construct its least common ancestor hierarchy (LCAH) graph, which is defined as follows.

Definition: The *least common ancestor hierarchy (LCAH) graph* of an AROF with n variables consists of $\binom{n}{2}$ vertices, one corresponding to each (unordered) pair of variables. For the distinct variables, x and y, denote the corresponding vertex by xy or, equivalently, yx. Then, for distinct vertices xy and zw, the directed edge $xy \to zw$ is present in the LCAH graph if and only if $\mathrm{lca}(x, y) \leq \mathrm{lca}(z, w)$.

We shall prove that the following algorithm constructs the LCAH graph of an AROF.

Algorithm CONSTRUCT-LCAH-GRAPH

1. **in parallel for all distinct variables** x, y, z **do**
   if DESCENDANT$(x, y, z) = \text{YES}$ **then**
   insert edges $xy \to xz$ and $xy \to yz$ and $xz \to yz$ and $yz \to xz$
2. **in parallel for all distinct variables** x, y, z, w **do**
   if edges $xy \to xw \to xz$ are present **then**
   insert edge $xy \to xz$

3. **in parallel for all distinct variables** x, y, z **do**
   if no edges between any of xy, xz, yz are present **then**
   insert edges in each direction between every pair of xy, xz, yz
4. **in parallel for all distinct variables** x, y, z, w **do**
   if edges $xy \rightarrow xw \rightarrow zw$ present **or** edges $xy \rightarrow yw \rightarrow zw$ present **then**
   insert edge $xy \rightarrow zw$

Theorem 5: *Algorithm* **CONSTRUCT-LCAH-GRAPH** *constructs the LCAH graph of an AROF.*

Proof: The proof follows from the following sequence of observations:

(i) For all distinct variables x, y and z for which $\text{lca}(x, y) < \text{lca}(x, z) = \text{lca}(y, z)$, after executing steps 1 and 2 of the algorithm, the appropriate edges pertaining to vertices xy, xz and yz (namely, $xy \rightarrow xz$, $xy \rightarrow yz$, $xz \rightarrow yz$ and $yz \rightarrow xz$) are present.

(ii) For all distinct variables x, y and z for which $\text{lca}(x, y) = \text{lca}(x, z) = \text{lca}(y, z)$, after executing step 3 of the algorithm, the appropriate edges pertaining to vertices xy, xz and yz (namely, edges in both directions between every pair) are present.

(iii) For all distinct variables x, y, z and w, after executing step 4 of the algorithm, the edge $xy \rightarrow zw$ is present if and only if $\text{lca}(x, y) \leq \text{lca}(z, w)$.□

It is straightforward to verify that algorithm **CONSTRUCT-LCAH-GRAPH** can be implemented to run in $O(\log n)$ time on an EREW PRAM with $O(n^4)$ processors. Moreover, the $O(n^3)$ membership queries can be made initially in one parallel step.

In an AROF, each non-unary gate corresponds to a biconnected component (which is a clique) of its LCAH graph. Thus, to transform the LCAH graph into the extended skeleton of the AROF, we simply "compress" each of its connected components into a single vertex and then extract the underlying tree structure of this graph (where the underlying tree structure of a graph is the tree whose transitive closure is the graph[1]).

This is accomplished using standard graph algorithm techniques, including a parallel prefix sum computation ([LF80]). The details follow.

We first designate a "leader" vertex for each component. We then record the individual variables that are descendants of each non-unary gate, and then discard the other nodes in each connected component.

The algorithm below selects a leader from each connected component in an LCAH graph. We assume that there is a total ordering $\prec$ on the vertices of the LCAH graph (for example, the lexicographic ordering on the pair of indices of the two variables corresponding to each vertex).

Algorithm LEADER

in parallel for all vertices $xy \prec zw$ **do**
  if edges $xy \rightarrow zw$ **and** $zw \rightarrow xy$ are present **then**

[1]All edges are directed towards the root.

mark xy with X

It is easy to prove the following.

Lemma 6: *After executing algorithm* **LEADER**, *there is precisely one unmarked node (namely, the largest in the $\prec$ ordering) in each connected component of the LCAH graph.*

After selecting a leader from each component of the LCAH graph, we add n new nodes to this graph that correspond to the n variables. The edge $x \to yz$ is inserted if and only if the variable x is a descendant of $\mathrm{lca}(y,z)$. This is accomplished by the following algorithm.

Algorithm LEAVES

in parallel for all distinct variables x, y, z, w **do**
  insert edge $x \to xy$
  if edge $xy \to zw$ is present **then**
    insert edge $x \to zw$

Lemma 7: *After executing algorithm* **LEAVES**, *the edge $x \to yz$ in present if and only if variable x is a descendant of lca(y,z).*

Both algorithms **LEADER** and **LEAVES** can be implemented in $O(1)$ time with $O(n^4)$ processors.

After these steps, the marked nodes are discarded from the augmented LCAH graph (that contains $\binom{n}{2} + n$ vertices), resulting in a graph with at most $2n-1$ vertices that is isomorphic to the extended skeleton of the AROF. This discarding is accomplished by a standard technique involving the computation of prefix sums. We first adopt the convention that the order $\prec$ extends to the augmented LCAH graph as $x_1 \prec \cdots \prec x_n$ and $x \prec yz$ for any variables x, y and z. Then, for each node v, set

$$\varphi(v) = \begin{cases} 1 & \text{if } v \text{ is unmarked} \\ 0 & \text{if } v \text{ is marked,} \end{cases}$$

and compute the prefix sums

$$\sigma(v) = \sum_{u \preceq v} \varphi(u).$$

With algorithms for parallel prefix sum computation ([LF80]) this can be accomplished in $O(\log(\binom{n}{2} + n)) = O(\log n)$ time with $O(\binom{n}{2} + n) = O(n^2)$ processors.

The function σ is a bijection between the unmarked nodes of the augmented LCAH graph and some $S \subseteq \{1, 2, \ldots, 2n-1\}$, and $\sigma(x_i) = i$ when $i \in \{1, \ldots, n\}$. The following algorithm uses the values of this function to produce the extended skeleton of the AROF.

Algorithm COMPRESS-AND-PRUNE

in parallel for all distinct vertices u, v **do**
if vertices u, v are both unmarked
and edge $u \to v$ is in augmented LCAH graph **then**
insert edge $\sigma(u) \to \sigma(v)$ in skeleton graph
in parallel for all distinct $i, j, k \in S$ **do**
if edges $i \to j \to k$ **and** $i \to k$ are in skeleton graph **then**
remove edge $i \to k$ from skeleton graph

The following is straightforward to prove.

Lemma 5: *The "skeleton" graph that* **COMPRESS-AND-PRUNE** *produces is isomorphic to the extended skeleton of the AROF, where the inputs $x_1, \ldots, x_n$ correspond to the vertices $1, \ldots, n$ (respectively) of the graph.*

5.3 Determining a Read-Once Formula from its Skeleton

Once the skeleton of an AROF is determined, what remains is to determine the constants in its unary gates (note that the non-unary operations are easy to determine using the techniques in [BHH92]). We show how to do this in $O(\log^2 n)$ steps with $O(n \log n)$ processors. The main idea is to find a node that partitions the skeleton into three parts whose sizes are all bounded by half of the size of the skeleton. Then the unary gates are determined on each of the parts (in a recursive manner), and the unary gates required to "assemble" the parts are computed.

The following lemma is an immediate consequence from a result in [B74].

Lemma 9 [B74]: *For any formula $F(x_1, \ldots, x_n)$, there exists a non-unary gate of type $\star$ that "evenly" partitions it in the following sense. With a possible relabelling of the indices of the variables,*

$$F(x_1, \ldots, x_n) \equiv G(f_A(f_B(H(x_1, \ldots, x_k)) \star f_C(I(x_{k+1}, \ldots, x_l))), x_{l+1}, \ldots, x_n),$$

and the number of variables in $G(y, x_{l+1}, \ldots, x_n)$, $H(x_1, \ldots, x_k)$ and $I(x_{k+1}, \ldots, x_l)$ are all bounded above by $\lceil \frac{n}{2} \rceil$.

A minor technicality in the above lemma is that, since the skeleton is not necessarily a binary tree, it may be necessary to "split" a non-binary gate into two smaller gates.

It is straightforward to obtain the above decomposition of a skeleton in NC^1. Once this decomposition is obtained, the recursive algorithm for computing the unary gates of the ROF follows from the following lemma.

Lemma 10: *Let $x_1^{(0)}, \ldots, x_n^{(0)}$ be a total non-collapsing justifying assignment for the AROF $F(x_1, \ldots, x_n)$. If*

$$F(x_1, \ldots, x_n) \equiv G(f_A(f_B(H(x_1, \ldots, x_k))\star$$

$$f_C(I(x_{k+1}, \ldots, x_l))), x_{l+1}, \ldots, x_n)$$

then:
(i) Given the skeleton of $F(x_1, \ldots, x_n)$ and the subformulas $G(y, x_{l+1}, \ldots, x_n)$, $H(x_1, \ldots, x_k)$ and $I(x_{k+1}, \ldots, x_l)$, it is possible to determine A, B and C, and, thus, the entire structure of $F(x_1, \ldots, x_n)$ in $O(\log n)$ steps with $O(n \log n)$ processors.
(ii) Given the skeleton of $F(x_1, \ldots, x_n)$, the problem of determining $G(y, x_{l+1}, \ldots, x_n)$, $H(x_1, \ldots, x_k)$ and $I(x_{k+1}, \ldots, x_l)$ is reducible to the problem of determining a ROF given its skeleton.

Proof: For part (i), assume that the subformulas $G(y, x_{l+1}, \ldots, x_n)$, $H(x_1, \ldots, x_k)$ and $I(x_{k+1}, \ldots, x_l)$ are given. Since $x_1^{(0)}, \ldots, x_n^{(0)}$ is a justifying assignment, $G(y, x_{l+1}^{(0)}, \ldots, x_n^{(0)})$, $H(x_1, x_2^{(0)}, \ldots, x_k^{(0)})$, $I(x_{k+1}, x_{k+2}^{(0)}, \ldots, x_l^{(0)})$ are all nonconstant unary functions, so there exist nonsingular matrices A', B', C' (which are easy to determine in $O(\log n)$ parallel steps) such that

$$\begin{aligned} f_{A'}(y) &\equiv G(y, x_{l+1}^{(0)}, \ldots, x_n^{(0)}) \\ f_{B'}(x_1) &\equiv H(x_1, x_2^{(0)}, \ldots, x_k^{(0)}) \\ f_{C'}(x_{k+1}) &\equiv I(x_{k+1}, x_{k+2}^{(0)}, \ldots, x_l^{(0)}). \end{aligned}$$

Also,

$$F(x_1, x_2^{(0)}, \ldots, x_k^{(0)}, x_{k+1}, x_{k+2}^{(0)}, \ldots, x_n^{(0)}) \equiv f_{A'}(f_A(f_B(f_{B'}(x_1) \star f_C(f_{C'}(x_{k+1})),$$

so the matrices $A' \cdot A$, $B \cdot B'$, $C \cdot C'$ can be determined in $O(1)$ steps. From this, the matrices A, B, C can be determined.

For part (ii), consider the problem of determining $G(y, x_{l+1}, \ldots, x_n)$. Note that

$$F(y, x_2^{(0)}, \ldots, x_l^{(0)}, x_{l+1}, \ldots, x_n) \equiv G(f_{A''}(y), x_{l+1}, \ldots, x_n),$$

for some nonsingular A''. Therefore, if we fix $x_2, \ldots, x_l$ to $x_2^{(0)}, \ldots, x_l^{(0)}$ then we have a reduction from the problem of determining $G(f_{A''}(y), x_{l+1}, \ldots, x_n)$.

Similarly, we have reductions from the problem of determining $f_{B''}(H(x_1, \ldots, x_k))$ and $f_{C''}(I(x_{k+1}, \ldots, x_l))$ for nonsingular matrices B'' and C''. Since the matrices A'', B'', C'' can be absorbed into the processing of part (i) this is sufficient.□

By recursively applying Lemmas 9 and 10, we obtain a parallel algorithm to determine an AROF given its skeleton and a total noncollapsing three-way justifying assignment in $O(\log^2 n)$ steps. The processor count for this can be bounded by $O(n \log n)$.

References

[A87] D. Angluin. Queries and concept learning. *Machine Learning*, 2, pages 319-342, 1987.

[AHK89] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. Technical report, Report No. UCB/CSD 89/528, Computer Science Division, University of California Berkeley, 1989. To appear, *J. ACM.*

[AK91] D. Angluin and M. Kharitonov. When won't membership queries help? In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 444–454, 1991.

[BOT88] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the 20th Annual Symposium on the Theory of Computing*, pages 301–309, 1988.

[B74] R. P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, 1974.

[BHH92] N. H. Bshouty, T. R. Hancock, and L. Hellerstein. Learning arithmetic read-once formulas. In *Proceedings of the 24th Annual Symposium on the Theory of Computing*, 1992.

[BC92] N. H. Bshouty, R. Cleve. On the exact learning of formulas in parallel, Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, 24–27, 1992), 513–522, 1992.

[B2H92] N. H. Bshouty, T. R. Hancock, and L. Hellerstein. Learning boolean read-once formulas with arbitrary symmetric and constant fan-in gates. In *The 1992 Workshop on Computational Learning Theory*, 1992.

[BGHM93] N. H. Bshouty, S. Goldman, T. Hancock and S. Matar. Asking Questions to Minimize Errors. In *The 1993 Workshop on Computational Learning Theory*,1993.

[BHHK91] N. H. Bshouty, T. R. Hancock, L. Hellerstein, and M. Karpinski. Read-once threshold formulas, justifying assignments, and transformations. Unpublished Manuscript.

[BT88] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 301–309, 1988.

[BT90] A. Borodin and P. Tiwari. On the decidability of sparse univariate polynomials. In *Proceedings of the 22nd Symposium on the Theory of Computing*, 1990.

[GKS90a] S. A. Goldman, M. J. Kearns, and R. E. Schapire. Exact identification of circuits using fixed points of amplification functions. In *Proceedings of the 31st Symposium on Foundations of Computer Science*, 1990.

[GKS88] D.Y. Grigoriev, M. Karpinski, and M.F. Singer. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. Technical report, Research Report No. 8523-C5, University of Bonn (1988), 1988. To appear in *SIAM J. Comp.*

[GKS90b] D.Y. Grigoriev, M. Karpinski, and M.F Singer. Interpolation of sparse rational functions without knowing bounds on the exponent. In *Proceedings of the 31st Symposium on Foundations of Computer Science*, 1990.

[Han90] T. Hancock. Identifying μ-formula decision trees with queries. In *The 1990 Workshop on Computational Learning Theory*, pages 23–37, 1990.

[HH91] T. Hancock and L. Hellerstein. Learning read-once formulas over fields and extended bases. In *The 1991 Workshop on Computational Learning Theory*, 1991.

[HS80] J. Heintz and C. P. Schnorr. Testing polynomials that are easy to compute. In *Proceedings of the 12th Annual Symposium on the Theory of Computing*, pages 262–272, 1980.

[LF80] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *J. ACM* 27(4):831–838, 1980.

[L88] N. Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear Threshold Algorithm, *Machine Learning*, pp. 285–318, v. 2, n. 4, 1988.

[M91] Y. Mansour. Randomized approximation and interpolation of sparse polynomials. To appear in *SIAM Journal on Computing*.

[MT90] W. Maass and G. Turán. On the complexity of learning from counterexamples and membership queries. In *Proceedings of the 31st Symposium on Foundations of Computer Science*, 1990.

[RB89] M. R. Roth and G. M. Benedek. Interpolation and approximation of sparse multivariate polynomials over gf(2). Manuscript (to appear in *SIAM J. Computing*), 1989.

[1] R. E. Schapire and L. M. Sellie. Learning sparse multivariate polynomials over a field with queries and counterexamples. In *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 17-26, 1993.

[Sch80] J. T. Schwartz. Fast polynomial algorithms for verification of polynomial identities. *Journal of the Association for Computing Machinery*, 27(4):701–707, 1980.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[VL] J. S. Vitter and J. Lin, Learning in parallel, *The 1988 Workshop on Computational Learning Theory*, pages 106-124, 1988.