SIAM J. COMPUT. © 1992 Society for Industrial and Applied Mathematics
Vol. 21, No. 1, pp. 54-58, February 1992 006

COMPUTING ALGEBRAIC FORMULAS USING A CONSTANT NUMBER OF REGISTERS*

MICHAEL BEN-OR† AND RICHARD CLEVE‡

Abstract. It is shown that, over an arbitrary ring, the functions computed by polynomial-size algebraic formulas are also computed by polynomial-length algebraic straight-line programs that use only three registers. This was previously known for Boolean formulas [D. A. Barrington, *J. Comput. System Sci.*, 38 (1989), pp. 150-164], which are equivalent to algebraic formulas over the ring $GF(2)$. For formulas over arbitrary rings, the result is an improvement over previous methods that require the number of registers to be logarithmic in the size of the formulas in order to obtain polynomial-length straight-line programs. Moreover, the straight-line programs that arise in these constructions have the property that they consist of statements whose actions on the registers are linear and bijective. A consequence of this is that the problem of determining the iterated product of n 3×3 matrices is complete (under P-projections) for algebraic NC^1. Also, when the ring is $GF(2)$, the programs that arise in the constructions are equivalent to bounded-width permutation branching programs.

Key words. algebraic computing, straight-line programs, complexity classes

AMS(MOS) subject classifications. 68Q15, 68Q40

1. Introduction.

The first investigation of the computational power of programs whose on-line storage capacity is limited to a constant number of data items was made by Borodin, Dolev, Fich, and Paul [5] and Chandra, Furst, and Lipton [8]. These authors considered bounded-width branching programs computing functions from $\{0, 1\}^n$ to $\{0, 1\}$. Such programs are equivalent to straight-line programs that employ a constant number of $\{0, 1\}$-valued read/write registers and have read-only access to their inputs (disregarding linear differences in the lengths of programs). An advantage of working with straight-line programs is that, by allowing the inputs and registers to take values from general algebraic structures, they extend naturally to a model of computation on more general types of data.

One way of assessing the computational power of these programs is to relate it to other models of computation, such as circuits or formulas. Circuits and formulas, like straight-line programs, extend naturally from settings where the data is $\{0, 1\}$-valued to settings where the data takes values from general algebraic structures. Brent [7] proved that, if the algebraic structure is a ring, any formula of size s can be "restructured" to have depth $O(\log s)$. (Actually, Brent's result, as it is stated, applies to commutative rings, but it is easily modified to apply to general rings.)

Barrington [2] showed how to compute Boolean formulas of size s (or, equivalently, depth $O(\log s)$) by bounded-width branching programs of length polynomial in s. One interesting consequence of this result is that the MAJORITY function from $\{0, 1\}^n$ to $\{0, 1\}$ is computed by a bounded-width branching program of length polynomial in n. It was previously thought (and conjectured in [5]) that the MAJORITY function is not computable by a polynomial-length bounded-width branching program. (The conjecture was supported by some results obtained under restricted conditions.)

* Received by the editors May 22, 1990; accepted for publication (in revised form) March 21, 1991. A preliminary version of this paper was presented at the 20th Annual ACM Symposium on Theory of Computing, Chicago, Illinois, 1988 [4].

† Institute of Mathematics and Computer Science, Hebrew University, Jerusalem, Israel.

‡ Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4. This research was conducted while this author was at the University of Toronto and partially supported by a Natural Sciences and Engineering Research Council of Canada postgraduate scholarship.

Our result is the following. Let $f(x_1, \cdots, x_n)$ be an algebraic formula of size s over an arbitrary ring $(\mathcal{R}, +, \cdot, 0, 1)$. We show how to construct a straight-line program of length polynomial in s that computes $f(x_1, \cdots, x_n)$ and uses only three registers. (Our result directly bounds the length of the straight-line program by 4^d, where d is the depth of the formula. Brent's result [7] allows us to assume that $d \in O(\log s)$.)

In the special case where the ring is $GF(2)$, our result is equivalent to Barrington's [2]. Over general rings, we obtain an improvement over the previously known (straightforward) method that involves an evaluation of each node in the formula in a "depth-first traversal" order. This latter method requires a number of registers that is equal to the depth of the formula, which, in general, is at least logarithmic in the size of the formula.

Also, the straight-line programs that arise in our constructions have a special form: they consist of statements that apply special invertible linear operations to the registers. More precisely, if one regards each possible configuration of values of the three registers as a vector in $\mathcal{R}^3$, then the effect of executing a statement of these programs is equivalent to multiplying this vector by a 3×3 matrix with determinant 1 (and one entry of this matrix is an input or its negation, and the other entries are constants). Thus, the statements that constitute these programs can be viewed as elements of $SL_3(\mathcal{R})$, the *special linear group*, consisting of 3×3 matrices with determinant 1. In Barrington's constructions [2], the statements of the programs can be viewed as elements of the group S_5, of permutations on a five-element set. To further compare our results, we note that, when the ring is $GF(2)$, our programs are (in the language of Barrington) "width-7 permutation branching programs" and, when $\mathcal{R}$ is finite, our programs are "$SL_3(\mathcal{R})$-permutation branching programs."

The main motivation for this research is to investigate alternate characterizations of the complexity class "algebraic NC^1" (functions computed by logarithmic-depth algebraic circuits). In addition, Kilian [10] has shown that the fact that the programs that arise in Barrington's constructions [2] are permutation programs has applications in the design of cryptographic protocols. By expressing Boolean formulas as bounded-width permutation branching programs, Kilian shows how to construct cryptographic protocols that perform "oblivious function evaluations" of Boolean formulas with a constant number of rounds of interaction (whereas previously proposed constructions require $\Omega(\log s)$ rounds, for general formulas of size s). Using our results, this can be extended to formulas over other rings, such as the integers (see Bar-Ilan and Beaver [1] for a particular construction).

The proof of our main result is partly motivated by the constructions used by Coppersmith and Grossman [9] and Barrington [2].

2. Models of computation.

2. Models of computation. Let $(\mathcal{R}, +, \cdot, 0, 1)$ be an arbitrary ring. In this section, we define formulas and straight-line programs over $\mathcal{R}$. Our definition of formulas is very standard (as circuits that are trees). Our straight-line program model is similar to conventional models (in which there is a set of registers that contain single ring elements and statements perform single ring operations) except that our statements each perform *two* ring operations. Due to the particular form of the operations in these straight-line programs, we call them "linear bijection straight-line programs." Any statement in such a program can be easily simulated by two statements in a more conventional straight-line program if one additional register is available.

DEFINITION 1. A *formula* over $(\mathcal{R}, +, \cdot, 0, 1)$ of depth d is defined as follows. A depth 0 formula is either c, for some $c \in \mathcal{R}$ (a *constant*), or x_u, for some $u \in \{1, 2, \cdots\}$ (an *input*). For $d > 0$, a depth d formula is either $(f + g)$ or $(f \cdot g)$, where f and g are formulas of depth d_f and d_g (respectively) and $d = \max(d_f, d_g)+1$. The *size* of a formula is defined as follows. A depth 0 formula has size 1, and if f and g have sizes s_f and s_g (respectively), then the sizes of $(f+g)$ and $(f\cdot g)$ are both s_f+s_g+1. A formula computes a function from $\mathcal{R}^n$ to $\mathcal{R}$ in a natural way (where n is the number of distinct inputs occurring in the formula).

DEFINITION 2. A *linear bijection straight-line program* (LBS *program*) over $(\mathcal{R}, +, \cdot, 0, 1)$ is a sequence of assignment statements of the form

$$\begin{aligned} &R_j \leftarrow R_j + (R_i \cdot c); \quad \text{or}\\ &R_j \leftarrow R_j - (R_i \cdot c); \quad \text{or}\\ &R_j \leftarrow R_j + (R_i \cdot x_u); \quad \text{or}\\ &R_j \leftarrow R_j - (R_i \cdot x_u), \end{aligned}$$

where $i, j \in \{1, \cdots, m\}$, $i \neq j$, $c \in \mathcal{R}$, and $u \in \{1, \cdots, n\}$. $R_1, \cdots, R_m$ are *registers* and $x_1, \cdots, x_n$ are *inputs* (m is the number of registers and n is the number of inputs). The *length* of an LBS program is the number of statements it contains. LBS programs compute functions from $\mathcal{R}^n$ to $\mathcal{R}$ in a natural way, provided that we have some fixed convention about the initial values of registers, and about which register's final value is taken to be the output of the computation (we will specify these later).

For each specific value of the inputs $x_1, \cdots, x_n$, each statement in an LBS program induces a transformation on the row vector consisting of the values of the registers $(R_1, \cdots, R_m)$. This transformation can be represented by the matrix whose diagonal entries are 1, whose ijth entry is $\pm c$ or $\pm x_u$ (depending on which of the four basic forms the statement takes), and whose other entries are 0. Executing a statement is equivalent to multiplying $(R_1, \cdots, R_m)$ on the right by the corresponding matrix. For example, the statement $R_1 \leftarrow R_1 + (R_2 \cdot x_1)$ corresponds to the matrix (when $m = 3$)

$$\begin{pmatrix} 1 & 0 & 0\\ x_1 & 1 & 0\\ 0 & 0 & 1 \end{pmatrix}.$$

In this manner, the statements in an LBS program correspond to elements of $SL_m(\mathcal{R})$, the special linear group consisting of $m\times m$ matrices with determinant 1. In particular, in the language of Valiant [11], an LBS program of length l that uses m registers can be viewed as a "P-projection" of an iterated product of l $m\times m$ matrices.

Also, in the language of Barrington [2], when the underlying ring is $GF(2)$, each LBS program that uses m registers is equivalent to a "permutation branching program" of width 2^m-1 (where the "states" are the nonzero elements of $\{0,1\}^m$). Barrington also defines "G-permutation branching programs" for any finite group G. When $\mathcal{R}$ is finite, our LBS programs correspond to $SL_m(\mathcal{R})$-permutation branching programs.

3. Main result.

DEFINITION 3. Let $f(x_1, \cdots, x_n)$ be an arbitrary formula over $\mathcal{R}$. For distinct $i, j \in \{1, \cdots, m\}$, we say that an LBS program *offsets* R_j *by* $+R_i \cdot f(x_1, \cdots, x_n)$ if it transforms the values of the registers as follows. R_j is incremented by the value of R_i times $f(x_1, \cdots, x_n)$ and (importantly) all other registers incur no net change (i.e., for all $k \neq j$, R_k has the same final value as its initial value). For example, the single statement $R_1 \leftarrow R_1 + (R_2 \cdot x_1)$ offsets R_1 by $+R_2 \cdot x_1$. Similarly, we say that an LBS program *offsets* R_j *by* $-R_i \cdot f(x_1, \cdots, x_n)$ if it *decrements* R_j by the value of R_i times $f(x_1, \cdots, x_n)$ and causes no net change in the values of all other registers. For example, the single statement $R_1 \leftarrow R_1 - (R_2 \cdot x_1)$ offsets R_1 by $-R_2 \cdot x_1$.

Note that to compute $f(x_1, \cdots, x_n)$ it is sufficient to construct an LBS program that offsets R_j by $+R_i \cdot f(x_1, \cdots, x_n)$ if one adopts an initialization convention where R_i is initially 1 and R_j is initially 0, and one adopts the convention that the value of R_j is the output of the computation.

For convenience, we say that we have LBS programs that offset R_j by $\pm R_i \cdot f(x_1, \cdots, x_n)$ whenever we have an LBS program that offsets R_j by $+R_i \cdot f(x_1, \cdots, x_n)$ as well as an LBS program that offsets R_j by $-R_i \cdot f(x_1, \cdots, x_n)$.

THEOREM 1. *Over an arbitrary ring* $(\mathcal{R}, +, \cdot, 0, 1)$, *any formula* $f(x_1, \cdots, x_n)$ *of depth* d *is computed by an* LBS *program that uses three registers and has length at most* $(2^d)^2$.

Proof. Recursively on d, the depth of $f(x_1, \cdots, x_n)$, we construct LBS programs that use the three registers and offset R_j by $\pm R_i \cdot f(x_1, \cdots, x_n)$ (for distinct $i, j \in \{1, 2, 3\}$).

The construction is trivial when $d = 0$: the appropriate single statement offsets R_j by $\pm R_i \cdot c$, or by $\pm R_i \cdot x_k$.

Suppose that we have LBS programs that offset R_j by $\pm R_i \cdot f(x_1, \cdots, x_n)$, and that offset R_j by $\pm R_i \cdot g(x_1, \cdots, x_n)$. Then we can offset R_j by $+R_i \cdot (f+g)(x_1, \cdots, x_n)$ by the LBS program

```
offset $R_j$ by $+R_i \cdot f(x_1, \cdots, x_n)$
offset $R_j$ by $+R_i \cdot g(x_1, \cdots, x_n)$,
```

and we can construct a similar program that offsets R_j by $-R_i \cdot (f+g)(x_1, \cdots, x_n)$. The interesting part of our construction is that we can offset R_k by

$$+R_i \cdot (f \cdot g)(x_1, \cdots, x_n)$$

by the LBS program

```
offset $R_k$ by $-R_j \cdot g(x_1, \cdots, x_n)$
offset $R_j$ by $+R_i \cdot f(x_1, \cdots, x_n)$
offset $R_k$ by $+R_j \cdot g(x_1, \cdots, x_n)$
offset $R_j$ by $-R_i \cdot f(x_1, \cdots, x_n)$.
```

To verify that this program has the required properties, let r_i, r_j, r_k be the values of R_i, R_j, R_k (respectively) before executing the above program. Note that register R_i is not modified by any of the four stages in the program. Register R_j is incremented by $r_i \cdot f(x_1, \cdots, x_n)$ (in the second stage) and then decremented by $r_i \cdot f(x_1, \cdots, x_n)$ (in the fourth stage), and thus register R_j incurs no net change from the execution of the program. Finally, register R_k is decremented by $r_j \cdot g(x_1, \cdots, x_n)$ (in the first stage) and then incremented by

$$(r_j + r_i \cdot f(x_1, \cdots, x_n)) \cdot g(x_1, \cdots, x_n)$$

(in the third stage). Thus, the net effect of the above program is to increment the contents of register R_k by $r_i \cdot (f \cdot g)(x_1, \cdots, x_n)$, as claimed.

Also, there is a similar construction that offsets R_k by $-R_i \cdot (f \cdot g)(x_1, \cdots, x_n)$ (simply switch the "+" and "−" in the second and fourth stages).

Since the maximum recursive factor per level of depth in this construction is four, if the depth of $f(x_1, \cdots, x_n)$ is d, the resulting LBS program has length at most $4^d = (2^d)^2$. □

Combining Theorem 1 with Brent's restructuring result [7], we obtain the following corollary.

COROLLARY 2. *Over an arbitrary ring* $(\mathcal{R}, +, \cdot, 0, 1)$, *any formula* $f(x_1, \cdots, x_n)$ *of size* s *is computed by an* LBS *program that uses three registers and has length polynomial in* s.

By recalling (from § 2) the close relationship between LBS programs and iterated products of matrices, and noting that iterated products of 3×3 matrices are easily expressed as polynomial-size formulas, we obtain the following from Corollary 2.

COROLLARY 3. *Over an arbitrary ring* $(\mathcal{R}, +, \cdot, 0, 1)$, *the problem of computing the product of* n 3×3 *matrices is complete under P-projections* (*as defined by Valiant* [11]) *for algebraic* NC^1 (*the class of functions computed by polynomial-size formulas*). *Moreover, the problem remains complete if the matrices are restricted to those that have determinant* 1.

Finally, we mention that the uniform versions of Barrington's result [2], [3] also carry over to our results in a straightforward manner. For example, Corollary 3 also holds for log-time uniform NC^1 and with respect to log-time uniform P-projections.

Acknowledgments. We would like to thank an anonymous referee for making thoughtful and constructive comments. Also, the second author thanks Charles Rackoff for raising some interesting questions concerning the cryptographic security of certain block ciphers that helped inspire this work.

REFERENCES

[1] J. BAR-ILAN AND D. BEAVER, *Non-cryptographic fault-tolerant computing in constant number of rounds of interaction*, in Proc. 8th Annual ACM Symposium on Principles of Distributed Computing, 1989, pp. 201-209.

[2] D. A. BARRINGTON, *Bounded-width polynomial-size branching programs recognize exactly those languages in* NC^1, J. Comput. System Sci., 38 (1989), pp. 150-164.

[3] D. A. BARRINGTON, N. IMMERMAN, AND H. STRAUBING, *On uniformity within* NC^1, in Proc. 3rd Annual IEEE Conference on Structure in Complexity Theory, 1988, pp. 47-59.

[4] M. BEN-OR AND R. CLEVE, *Computing algebraic formulas using a constant number of registers*, in Proc. 20th Annual ACM Symposium on Theory of Computing, 1988, pp. 254-257.

[5] A. BORODIN, D. DOLEV, F. FICH, AND W. PAUL, *Bounds for width two branching programs*, SIAM J. Comput., 15 (1986), pp. 549-560.

[6] A. BORODIN AND I. MUNRO, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.

[7] R. P. BRENT, *The parallel evaluation of general arithmetic expressions*, J. Assoc. Comput. Mach., 21 (1974), pp. 201-206.

[8] A. CHANDRA, M. FURST, AND R. J. LIPTON, *Multiparty protocols*, in Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 94-99.

[9] D. COPPERSMITH AND E. GROSSMAN, *Generators for certain alternating groups with applications to cryptography*, SIAM J. Appl. Math., 29 (1975), pp. 624-627.

[10] J. KILIAN, *Founding cryptography on oblivious transfer*, in Proc. 20th Annual ACM Symposium on Theory of Computing, 1988, pp. 20-31.

[11] L. G. VALIANT, *Completeness classes in algebra*, in Proc. 11th Annual ACM Symposium on Theory of Computing, 1979, pp. 249-261.