JOURNAL OF COMPUTER AND SYSTEM SCIENCES 50, 441–455 (1995)

Circuits, Matrices, and Nonassociative Computation

MARTIN BEAUDRY*

Département de Mathématiques et d'Informatique, Université de Sherbrooke, Sherbrooke, Québec, J1K 2R1 Canada

AND

PIERRE MCKENZIE†

Département d'I.R.O., Université de Montréal, Montréal, Québec, H3C 3J7 Canada

Received February 3, 1994

We consider the complexity of various computational problems over nonassociative algebraic structures. Specifically, we look at the problem of evaluating circuits, formulas, and words, over both nonassociative structures themselves and over matrices with elements in these structures. Extending past work, we show that such problems can characterize a wide variety of complexity classes up to and including *NP*. As an example, the word (i.e., iterated multiplication) problems involving a sequence of $O(\log^k n)$ matrices over a structure $(S; +, \cdot)$ in which $(S; +)$ is a monoid or an aperiodic monoid are complete for NC^{k+1} and for AC^k, respectively, and a word problem variant involving matrices of size $O(\log^k n)$ is complete for SC^k.

1. INTRODUCTION

An algebraic circuit, or "straight-line program," is a circuit whose inputs are elements of an algebraic structure and whose gates perform an algebraic operation on the gate inputs. One usually thinks of such circuits as defined over algebraic structures with associative binary operators, for instance groups, semigroups, semirings, rings, or fields. The Boolean circuit (with bounded or unbounded fanin), the formula (i.e., outdegree-one circuit), and the simple "word" (i.e., sequence of elements with an implicit binary operator) are then natural special cases of the algebraic circuit. The complexity of circuit evaluation, in which a circuit and its inputs are given as data and the task is to determine the value computed at the output gate of the circuit, has been studied at length [1, 7, 8, 10, 12, 14, 18, 20, 21, 23–25, 27–29].

* Work supported by NSERC Grant OGP0089786 and FCAR Grants 92-NC-0608 and 91-ER-0642. E-mail: beaudry@dmi.usherb. ca.

† Work supported by NSERC Grant A9979 and by FCAR Grant 91-ER-0642 and performed in part while on sabbatical leave at the University of British Columbia. E-mail: mckenzie@iro.umontreal.ca.

The circuit evaluation problem involving associative structures is relevant to structural complexity theory because its various restrictions capture an extensive list of complexity classes. Here are examples:

- The Boolean circuit value problem is complete for P [25, 20].
- The Boolean formula value problem is complete for NC^1 [12, 14].
- Word problems, i.e., "iterated multiplication problems," over appropriately represented associative structures, are complete for L [18], NL[23], and DET^* (see [17, 23]).
- The word problem over the alternating group $\mathcal{A}_5$ provides the key to the simulation of NC^1 by bounded-width polynomial-length branching programs [3].
- Mapping out the internal structure of NC^1, excluding its relationship to TC^0, amounts to determining the complexity of regular languages defined as word problems over appropriate monoids [8, 7, 4, 27].

In this paper we examine the algebraic circuit evaluation problem in the *nonassociative* context. Consider a circuit gate g, labelled with a nonassociative binary operator, having inputs $g_1, \ldots, g_r$ in that order. We define the operation of gate g as follows. The gate first picks, nondeterministically, a binary tree with r leaves and assigns $g_1, \ldots, g_r$ to these leaves in the usual order. The binary operator that labels g is then assigned to each internal node of the tree. The value computed at gate g is defined to be the value computed in the obvious way at the root of this binary tree. This definition thus equates nonassociativity with nondeterminism and reduces to the usual gate operation in the case of an associative operation. A similar definition introduced in [10] led to a fixed *groupoid*, i.e., nonassociative binary algebra, over which the word problem is complete for *LOGCFL*.

0022-0000/95 $6.00

Our first contribution is to observe that the groupoid generalization introduced by Bédard *et al.* [10] in the setting of word problems naturally adapts to the settings of formulas and circuits. In other words, it makes sense to consider formulas which are only partially parenthesized and to consider circuits in which a node N labelled with a nonassociative binary operator may have indegree larger than two. We prove that the resulting "nondeterministic formula value problem," in the general case in which the problem input includes the multiplication table of the groupoid (together with the outdegree-one circuit and the inputs to the circuit), remains in *LOGFCL*; hence the problem is *LOGFCL*-complete by [10]. For its part, the "nondeterministic circuit value problem" is clearly in *NP*, regardless of whether the groupoid is specified as part of the input (using any representation which allows multiplying two groupoid elements in polynomial time); for circuits of logarithmic depth over the fixed groupoid $(\{0, 1\}; NAND)$, the problem is *NP*-complete.

Turning to "deterministic formulas," i.e., fully parenthesized formulas, we observe that over any fixed algebra, the problem remains in NC^1 [12, 14, 13], provided that the infix representation is used; when the formula is represented by means of its direct connection language, the problem (even over $(\{0, 1\}; NAND)$) is L-complete. Another aspect of the study of unbounded fanin constant-depth circuits emerges: how powerful are such circuits over various nonassociative bases? We make the simple observation that such circuits over $(\{0, 1\}; NAND)$ characterize NC^0.

Our second contribution concerns the above evaluation problems, this time with matrix operands. To our knowledge, only two papers [10, 30] have considered the complexity of evaluation problems over nonassociative structures which grow with the input length. Here we consider such problems involving matrices whose entries are drawn from a fixed algebra $(S; +, \cdot)$ which need not be a semiring. In particular, $(S; +)$ and $(S; \cdot)$ need not be associative. When a single-valued vector inner product definition is given, such matrices form a groupoid $\mathcal{M}(S; +, \cdot)$ under the usual matrix multiplication rules. This leads to evaluation problems over exponentially large groupoids whose elements can be described concisely as part of the input.

For any fixed algebra $(S; +, \cdot)$, the problem of evaluating a depth $O(\log^k n)$ deterministic circuit, whose inputs are elements of $\mathcal{M}(S; +, \cdot)$ and whose gates call for multiplication in this groupoid, belongs to NC^{k+1}. We prove further that when $(S; +)$ is an appropriately chosen monoid (resp. cyclic group, solvable monoid, aperiodic monoid), the iterated multiplication and the deterministic circuit problems over $\mathcal{M}(S; +, \cdot)$ are complete for NC^{k+1} (resp. $CC^k(q)$, $ACC^k(q)$, AC^k; see the next section for the definitions of these classes). This result is an upward translation of the Barrington and Thérien characterizations of NC^1 subclasses [3, 8] to the level of $O(\log^k n)$ depth circuits. We also characterize the classes SC^k, i.e., DSPACE-TIME$[O(\log^k n), n^{O(1)}]$: for an appropriately chosen $(S; +, \cdot)$, an iterated multiplication problem variant involving $z(n) \times z(n)$ matrices over $(S; +, \cdot)$ with $z^2(n) \in O(\log^k n)$ is SC^k-complete.

Finally we consider "nondeterminism" in the context of matrices. We prove that there is a fixed algebra $(S; +, \cdot)$ such that the problem of evaluating a nondeterministic formula over $\mathcal{M}(S; +, \cdot)$ is *NP*-complete. Then we investigate the effect of introducing nondeterminism in the very definition of the vector inner product, resulting in a multiple-valued matrix product. We observe that multiplying two (resp. three) matrices then becomes *LOGCFL*-complete (resp. *NP*-complete). Furthermore, we show that an iterated multiple-valued matrix multiplication problem variant involving $z(n) \times z(n)$ matrices with $z^2(n) \in O(\log^k n)$ captures the class $NSC^k = \text{NSPACE-TIME}[O(\log^k n), n^{O(1)}]$.

In summary, our results indicate a way in which much of *NP* can be described uniformly in more algebraic terms, with the role of nondeterminism played by nonassociativity.

Section 2 in this paper presents some background definitions. Section 3 deals with the evaluation problems over a fixed algebraic structure. Section 4 is concerned with the evaluation problems involving matrices. We conclude the paper in Section 5 with a brief discussion.

2. PRELIMINARIES AND DEFINITIONS

For our purposes an *algebra* is a "carrier set" S equipped with a list $*_1, ..., *_k$ of unary or binary operators. Such an algebra is denoted $(S; *_1, ..., *_k)$. We assume throughout that $|S| \geq 2$. Our evaluation problems either involve finite algebras which are independent of the problem input, or finite algebras which are directly or indirectly specified as part of the problem input. An element $a \in S$ is *absorbing for the binary operator* $*_i$ iff, for each $s \in S$, $a *_i s = s *_i a = a$. An element $e \in S$ is an *identity for the binary operator* $*_i$ iff, for each $s \in S$, $e *_i s = s *_i e = s$. An algebra $(S; *)$ is a *monoid* if $*$ is an associative binary operator and S contains an identity for $*$. The *exponent* of a finite monoid M is the least $q \geq 1$ such that for some $t \geq 0$ every element $a \in M$ satisfies the identity $a^{t+q} = a^t$.

2.1. *Words, Formulas, and Circuits*

DEFINITION. A *circuit* C over an algebra $\mathcal{A} = (S; *_1, *_2, ..., *_k)$ is a directed acyclic graph with labelled nodes. Each indegree-zero node is labelled with an element[1] of S, each indegree-one node is labelled with a unary

[1] We consider the elements of S labelling the input gates, i.e., the "inputs to the circuit," as forming part of the circuit. Hence, "given a circuit" means what some authors would write as "given a circuit with inputs to the circuit."

operator of $\mathscr{A}$, and each remaining node is labelled with a binary operator of $\mathscr{A}$. Nodes of outdegree zero are the *output* nodes. We say a node is *nondeterministic* if its indegree is greater than 2 (irrespective of whether the node is labelled with an associative operator). The *circuit* is *nondeterministic* if it includes a nondeterministic node; it is *deterministic* otherwise. The *depth of a node g* is the length of the longest path from g to an outdegree-zero node; the *depth of C* is the depth of its deepest node. The *width at node g* is the number of non-degree-zero nodes at depth at least that of g which are predecessors of nodes shallower than g; the *width of C* is the maximum over all nodes of the width at a node.

The "word problem" of [10] and the "formula value problem" of [14] will be natural restriction of the problem, defined precisely below, of evaluating a nondeterministic circuit. Indeed the word problem as defined in [10] consists in determining whether a sequence of elements from an algebra $(S; *)$ can be bracketed in such a way as to yield a target element when evaluated. This same problem arises as the evaluation of an outdegree-one nondeterministic depth-one circuit. Now observe that an outdegree-one nondeterministic circuit defines a partially bracketed word over the circuit inputs. Hence the Boolean formulas of [14] arise as deterministic outdegree-one circuits over the Boolean algebra.

DEFINITION. A *nondeterministic formula* over an algebra $\mathscr{A}$ is a connected nondeterministic outdegree-one circuit over $\mathscr{A}$. A *formula* over $\mathscr{A}$ is a connected deterministic outdegree-one circuit over $\mathscr{A}$. A *linear formula* is a formula in which every indegree-two node has at least one leaf as an immediate predecessor. A *straight linear formula* is a linear formula in which every left child is a leaf. A *word* over an algebra $(S; *)$ is a nondeterministic outdegree-one depth-one circuit over $(S; *)$.

Each node in a deterministic circuit over $\mathscr{A} = (S; *_1, *_2, ..., *_k)$ "computes" an element of S in the usual way. In the nondeterministic case, we apply the "G-program" evaluation strategy crucial to the results of [10]: a nondeterministic node first picks a bracketing of its input nondeterministically and then evaluates accordingly. We stress that this nondeterministic choice occurs only once so that all nodes which have as immediate predecessor a common nondeterministic node g receive an identical value from node g.

DEFINITION. The *circuit* (resp. *formula, word*) *problem over a fixed algebra* $(S; *_1, *_2, ..., *_k)$ consists of determining, given C and s, whether C can evaluate to s, where

- C is a single-output *circuit* (resp. *formula, word*) over $(S; *_1, *_2, ..., *_k)$, and
- the "target value" s is an element of S.

2.2. *Matrices and Matrix Problems*

Each matrix evaluation problem discussed in this paper involves matrices whose entries are taken from a fixed (input-independent) finite algebra $(S; +, \cdot)$, where "$\cdot$" and "$+$" are arbitrary binary operators. We define the *size* of a matrix to be the number of entries in it. Each matrix problem involves a target matrix and a circuit with matrix inputs. In each case, the problem consists of determining whether the circuit (or formula or word) can evaluate to the target matrix. We define two basic types of such evaluation problems involving matrices.

The first type is obtained by defining an inner product of two vectors $V, W \in S^s$ using the formula

$$(\cdots(((v_1 \cdot w_1) + (v_2 \cdot w_2)) + (v_3 \cdot w_3)) + \cdots) + (v_s \cdot w_s). \quad (1)$$

Matrices $A \in S^{r \times s}$ and $B \in S^{s \times t}$ can be multiplied to yield a matrix $C \in S^{r \times t}$, using definition (1) to compute each entry in C. The set of matrices over S with matrix product thus forms a groupoid which we denote by $\mathscr{M}(S; +, \cdot)$. (To take care of products which would be "undefined" because of dimension mismatch, an absorbing element $\perp$ can be created.)

DEFINITION. *Evaluation problems over* $\mathscr{M}(S; +, \cdot)$ are those in which operands (e.g., circuit inputs) are matrices and in which the sole operation involved (e.g., the common label of all circuit gates) is the deterministic matrix multiplication defined implicitly by inner product (1). The problem consists of determining, given a circuit C and a target matrix M, whether C can evaluate to M.

The second type of matrix problem is obtained by embedding some nondeterminism into the inner product itself, as follows. Consider a vector inner product specified by

$$(v_1 \cdot w_1) + (v_2 \cdot w_2) + (v_3 \cdot w_3) + \cdots + (v_s \cdot w_s). \quad (2)$$

Since "$+$" is in general nonassociative, evaluating (2) is interpreted nondeterministically. This interpretation yields a matrix product which is not single-valued; an entry in the product of two matrices can take a number of values according to the nondeterministic choice of bracketing made in computing the expression (2) pertaining to this entry. Note that different nondeterministic bracketings may be chosen for different entries in the matrix product. However, we stress that when it is required to multiply more than two matrices, each nondeterministic choice required is made once and must be used throughout the computation; in other words, computing $(AB)C$ can be viewed as picking one of the nondeterministic values D for the product AB and then computing DC.

DEFINITION. *Multiple-valued product evaluation problems* are those in which operands (e.g., circuit inputs) are

matrices and in which sole binary operation involved (e.g., the common label of all circuit gates) is the nondeterministic matrix multiplication defined implicitly by inner product (2). The problem consists of determining, given a circuit C and a target matrix M, whether C can evaluate to M.

2.3. *Complexity Issues*

We assume familiarity with the complexity classes

$$NC^0 \subset AC^0 \subset ACC^0 \subseteq NC^1 \subseteq L \subseteq NL$$
$$\subseteq LOGCFL \subseteq AC^1 \subseteq NC^2 \subseteq P \subseteq NP$$

and with NC^1 and log space reducibilities (see [22, 17]). We adopt the definitions found in [14] for AC^0 and AC^0-reducibility (in their uniform settings). For $k \geqslant 1$ and $q \geqslant 2$, the classes AC^k and $ACC^k(q)$ are defined in terms of log space-uniform unbounded fanin Boolean circuits of $O(\log^k n)$ depth with gate types $\{AND, OR\}$ and $\{AND, OR, MOD_q\}$ respectively, where a MOD_q gate outputs 0 iff the sum of its Boolean inputs is a multiple of q. The classes $CC^k(q)$ are defined in terms of log space-uniform Boolean circuits of $O(\log^k n)$ depth with unbounded fanin MOD_q gates and bounded fanin gates of types $\{AND, OR\}$; note that for $q \geqslant 3$, this definition is equivalent to specifying that the circuit contains only unbounded fanin MOD_q gates (see [27, 32]). The class ACC^k (resp. CC^k) is the union over all $q \geqslant 2$ of the classes $ACC^k(q)$ (resp. $CC^k(q)$). The classes ACC^0 and CC^0 can also be defined as above, in terms of constant-depth circuits with the same (*DLOGTIME*) uniformity condition as used for AC^0 in [14]; ACC^0 and CC^0 are also known as "ACC" and "pure ACC," respectively [33]. The classes SC^k (resp. NSC^k) are defined as simultaneous deterministic (resp. nondeterministic) Turing machine SPACE-TIME$[O(\log^k n), n^{O(1)}]$ [16, 6].

Some results rely heavily on the recently developed simulations of Boolean circuits by programs over monoids such as the alternating group $\mathcal{A}_5$ [3, 5]. Recall that each instruction in such a program is specified by a function from the input alphabet to the monoid, together with an integer denoting one of the character positions within the input.

Unless otherwise stated, our evaluation problems over a fixed algebra are encoded over a fixed alphabet using the direct connection language of the circuits involved. More precisely (see [14, p. 759]), the direct connection language of an m-input circuit over a fixed algebra $(S; *_1, ..., *_k)$ is given by the set of 3-tuples $\langle \bar{m}, g, y \rangle$, where $\bar{m}$ is the binary encoding of m, $g \in \{0, 1\}^*$ is a gate number, and $y \in S \cup \{*_1, ..., *_k\} \cup \{0, 1\}^*$ is such that if $y \in \{0, 1\}^*$ then y is the number of a gate input to g; otherwise y is g's label. Recall that we view words and formulas as special cases of unbounded fanin circuits. The symbol n represents the length of the encoding of a problem instance. The only requirement which we add to the direct connection language as described in [14] concerns gate numbers; we insist that the numbers of gates input to a node be consistent with the order of evaluation at that node. We encode matrix evaluation problems in the same way, except that each input gate includes a pointer to the matrix operand it represents within a list of such matrices. We omit further details; any encoding which allows extracting basic information like the dimension of matrices or a specific matrix entry in NC^1 will do the job.

Note that in log space it is possible to translate a formula from its direct connection language representation to its infix representation. The reverse transformation can be done in NC^1.

3. FORMULAS AND CIRCUITS

In this section we consider evaluation problems over a finite algebra which is fixed in advance and remains the same for all inputs. (A *fixed* algebra refers to this situation.) We begin with the observation that in many cases no generality is lost by assuming that the fixed algebra involves only one operation.

PROPOSITION 3.1. *Any evaluation problem over an algebra $(S; *_1, *_2, ..., *_k)$ NC^1-reduces to the corresponding evaluation problem over a groupoid $(S'; *)$, where $|S'| \leqslant (k+1)(|S|+2)$ and where the depth, width, and size of the circuits involved are only affected by a constant multiplicative factor.*

Proof. The reduction goes in two steps. First, we dispose of all but one binary operator as follows. For every binary operator $*_i$ we add to S a copy $S^{(i)}$ of S. Then each node labelled with operation $*_i$, with inputs $s_1, ..., s_r$, that is, computing the word $s_1 *_i \cdots *_i s_r$, is replaced with a subcircuit computing the partially parenthesized expression

$$\downarrow_i((\uparrow_i s_1) * \cdots * (\uparrow_i s_r)),$$

where for all $s, t, w \in S$, we define $s^{(i)} * t^{(i)} = w^{(i)}$ iff $s *_i t = w$, and where $\uparrow_i$ and $\downarrow_i$ are new unary operators such that $(\uparrow_i s) = s^{(i)}$ and $(\downarrow_i s^{(i)}) = s$.

We are now left with a circuit involving only $*$ and unary operators, on a larger carrier set S'. For every unary operator u_i, add to S' a new element e_i, such that for every element s' defined so far, we have $e_i * s' = u_i(s')$. Thus each gate computing the expression $u_i(s')$ can be replaced with a subcircuit computing $e_i * s'$.

Now complete the definition of $(S'; *)$ arbitrarily. The resulting circuit evaluates to $s \in S$ iff the original circuit evaluated to s. To perform the reduction it suffices to be able to determine the operands of a node, and then to add or replace a circuit gate locally. Hence the reduction can be done in NC^1. ∎

The word problem over any nonsolvable group [3] and the problem of evaluating a formula presented in infix notation over $(\{0,1\};\ \wedge,\ \vee,\ \neg)$ are NC^1-complete [12, 14]. Buss [13] has recently presented a simplified NC^1 ($ALOGTIME$) algorithm to evaluate Boolean formulas. By allowing the pebble values in this algorithm to range over elements of a fixed algebra, we obtain the following. (Earlier proofs [12, 14] also extend to yield the next proposition but with a minor complication arising from the conversion to "Postfix Longer Operand First" representation in the event of a noncommutative algebra.)

PROPOSITION 3.2 [12, 14, 13]. *The problem of evaluating a formula presented in infix notation, over any fixed algebra, is in* NC^1.

We note that the infix representation is crucial to the NC^1 upper bound in Proposition 3.2.

PROPOSITION 3.3. *The formula evaluation problem over* $(\{0,1\};\ \wedge,\ \vee)$ *is L-complete under* NC^1 *reducibility. Computing the infix representation of a formula over* $(\{0,1\};\ \wedge,\ \vee)$ *from its "direct connection language" representation is complete for FL (= the class of functions computable in log space* [17]*) under* NC^1 *reducibility.*

Proof. The second statement follows from the first because evaluating the formula presented in infix can be done in NC^1 by Proposition 3.2. The NC^1 reduction proving the first statement is from the accessibility problem in a directed forest [18]. We can assume that the indegree of each node in this forest is either two or zero, that the source node u has indegree zero, and that the target node v has outdegree zero. Each tree in this forest can be seen as an outdegree one circuit by labelling the indegree zero nodes as inputs, the indegree two nodes as OR gates, and by considering the outdegree zero node as output. When setting the source node u to the Boolean value ONE and all other input nodes to ZERO, the target node v takes on the value ONE iff there was a path from u to v in the forest. To complete the reduction it is necessary to turn this multiple-output circuit into a single-output circuit. This is done by ANDing all but the target node with a ZERO input in a binary tree fashion and ORing the result with the target node.

To see that the problem is in L, note that the infix representation of the formula can be obtained in log space from its direct connection language representation. Then Lynch's algorithm [26] or Buss's simplified algorithm [13] solves the infix version of the problem. ∎

Bédard *et al.* showed that there is a groupoid over which the word problem is $LOGCFL$-complete, and they showed that the word problem over any fixed groupoid belongs to $LOGFCL$ [10]. We extend their upper bound to the case of nondeterministic formulas, i.e., "formulas" which are only partially parenthesized. A similar extension was obtain independently by Muscholl [30].

PROPOSITION 3.4. *The nondeterministic formula problem over any fixed algebra belongs to LOGCFL.*

Proof. By Proposition 3.1 we can take the fixed algebra to be a groupoid $(S; *)$. We reduce in log space the nondeterministic formula problem over $(S; *)$ to the word problem over a groupoid $(S'; \circ)$, the latter word problem belonging to $LOGCFL$ by [10]. The carrier set S' is built from S by defining

- three new elements, `left`, `right`, and $\perp$, and
- a "lower case" and a "upper case" copy of S.

The operation $\circ$ works inside the lower case copy of S exactly as $*$ does on S, while for each $s \in S$, we set

$$\texttt{left} \circ \mathit{lowercase}(s) = \mathit{uppercase}(s),$$

$$\mathit{uppercase}(s) \circ \texttt{right} = \mathit{lowercase}(s).$$

The definition of $\circ$ on S' is completed by setting any product not yet defined, in particular any product of two upper case elements, to $\perp$, which is absorbing. Now write F for the input nondeterministic formula over $(S; *)$. Let $w(F)$ be the word obtained from the infix representation of F by replacing each occurrence of "(" with "`left`" and each occurrence of ")" with "`right`," and by deleting each occurrence of "$*$." Then, for any $s \in S$, F evaluates to s iff $w(F)$ evaluates to $\mathit{lowercase}(s)$ when viewed as a word over $(S'; \circ)$. ∎

The proofs of Propositions 3.1 and 3.4 also apply to nondeterministic formulas over groupoids specified by their multiplication table as part of the input. Appealing to [10], this more general problem therefore belongs to $LOGCFL$ as well.

We now consider circuits. As is well known, polynomial size Boolean circuits with nondeterministic inputs characterize the class NP. Accordingly, our nondeterministic circuits are very powerful.

PROPOSITION 3.5. *The nondeterministic circuit problem over any fixed algebra belongs to NP; the balanced (i.e., depth* $O(\log n)$*) bounded fanin nondeterministic circuit problem over* $(\{0,1\}; NAND)$ *is NP-complete under log space reducibility.*

Proof. The circuit problem over any fixed algebra is solved in NP by guessing a bracketing at each circuit node and then evaluating the circuit deterministically. To prove hardness, note that a generic log space reduction to SAT [15] produces a formula, with repeated Boolean variables,

which on given inputs can be evaluated by an $O(\log n)$ depth circuit. Now, writing $*$ for *NAND*, observe that

$$0 = (0 * 1) * 1 \neq 0 * (1 * 1) = 1.$$

Thus nondeterministic *NAND* nodes with constant inputs 0, 1, and 1 can in effect produce nondeterministic bits to be used as values for the Boolean variables in the SAT formula. ∎

What is the power of constant depth polynomial size nondeterministic circuits over various groupoids? When the groupoid happens to be associative, this reduces to the study of the word problem (i.e., the circuit collapses to depth one with repeated inputs allowed). When the groupoid is not associative, questions related to the unproven proper containment of ACC^0 in NC^1 (see [8, 33]) can be formulated. Of course, for some nonassociative groupoids the situation is very simple. For instance, a *NAND* gate having four inputs has the property that on any setting of these inputs the output evaluates to zero or to one nondeterministically; this can be verified by brute force. By induction this property holds for *NAND* gates with five or more inputs as well. This leads to the following.

PROPOSITION 3.6. *Nondeterministic constant depth circuits over* $(\{0, 1\}; NAND)$ *compute all and only the* NC^0*-computable functions.*

4. MATRICES

We consider now the computational complexity of evaluation problems involving variable-size matrices over a fixed algebra. Our results show that formulas and circuits involving nonassociative matrix products provide a formalism which parallels that of Boolean circuits: the size, width, and depth parameters of the circuit correspond to description length, matrix size, and circuit/formula depth, respectively. This correspondence is easy to justify qualitatively. A matrix of size roughly equal to the width of the circuit can keep track of all gate values across a "circuit level," and values of the gates at the next level can be obtained by multiplication with an appropriate matrix. Hence a right to left product of $O(d)$ matrices (d being the depth of the circuit) will compute the output gate value.

Technically, we define a restriction of the evaluation problem for matrix circuits by doing one or more of:

- restricting the size, depth, and/or shape of the circuit,
- restricting the size of the matrices,
- forbidding nondeterminism,
- specifying algebraic properties for the underlying structure $(S; +, \cdot)$.

Our encodings for the instances of the evaluation problem are made in terms of square matrices of identical size. In order to ease understanding, however, several proofs are written in terms of matrices of various shapes, which can be blown up to be square through appropriate padding with an element, always denoted by 0, which is an identity for "$+$" and absorbing for "$\cdot$." If the algebra $(S; +, \cdot)$ does not already have such an element, then we can work instead in $(S'; +, \cdot)$, where $S' = S \cup \{0\}$. Therefore, we assume from now on that the algebra always contains 0.

4.1. *Circuit Depth*

This subsection deals exclusively with deterministic circuits. In order to capture complexity classes defined by Boolean circuits with restricted depth, we develop simulation techniques which enable us to keep control of the depth, at the cost of using large, although still polynomial-size, matrices.

In each of our completeness proofs, we observe that we can write our reductions with groupoids of matrices over algebras $(S; +, \cdot)$ where the operation "$+$" is associative. This observation leads immediately to the idea of adding restrictions on the properties of the monoid $(S; +)$; in doing so, we can apply known results on the subclasses of NC^1 to capture complexity classes defined by circuits with various types of gates (Theorem 4.5).

We first discuss the case of circuits involving a constant number of gates and matrices.

PROPOSITION 4.1. *For any algebra* $(S; +, \cdot)$, *the constant-size circuit problem over* $\mathcal{M}(S; +, \cdot)$ *belongs to* NC^1. *There is an algebra* $(S; +, \cdot)$ *such that computing the product of two matrices from* $\mathcal{M}(S; +, \cdot)$ *is* NC^1*-complete under* AC^0 *reducibility.*

Proof. In a circuit with a constant number of gates, the expression for each entry of the resulting matrix has polynomial length; it can be computed from the circuit description, and then be evaluated in NC^1 using Proposition 3.2. For the hardness part, define $(S; +)$ as a nonsolvable group (for instance $\mathcal{A}_5$) and appeal to the NC^1-hardness of the word problem over this group [3, 14]. We build from an instance z of the word problem over $\mathcal{A}_5$ an instance $MN = P$, where M is a row vector containing the word of the instance z, N is a column vector of matching dimension where each entry is some element λ which is an identity for "$\cdot$," and P is 1×1, where the entry is the target value of the instance z. Now the matrices are actually encoded in the form of a bit string enumerating the entries of three square matrices; we have to show that this string, which we denote by $f(z)$, can be built from z in *DLOGTIME* uniform AC^0. Consider the language $L = \{(i, z) \mid$ the ith bit in $f(z)$ has value ONE$\}$. A circuit to decide membership in L would, for instance, decide first which entry of which matrix the ith bit belongs to, through the computation of degree-2 polynomials in i and $|z|$, where $|z|$ denotes the length of z, and use the result to select the

value of the ith bit from the various possibilities (i.e., if the bit belongs to the encoding of a character in the instance of the word problem, to the encoding of constant λ, etc.). The problems of selection and of low-degree polynomial computation on small numbers both belong to *DLOGTIME* uniform AC^0. Now combining in parallel the AC^0 subcircuits which, for each $1 \leqslant i \leqslant |f(z)|$, with $|f(z)| \in O(|z|^2)$, determine whether $(i, z) \in L$, yields an AC^0 circuit which performs the desired reduction and which can also be made uniform. ▮

We use this proposition to obtain upper bounds for the special case of *straight linear formulas* over algebras of matrices. The case of arbitrary circuits of polylog depth is treated later, once the appropriate proof techniques have been developed (Theorem 4.4).

LEMMA 4.2. *For any algebra* $(S; +, \cdot)$ *and* $k \geqslant 0$, *the depth* $O(\log^k n)$ *straight linear formula problem over* $\mathcal{M}(S; +, \cdot)$ *belongs to* NC^{k+1}.

Proof. The case $k = 0$ is treated above. For $k \geqslant 1$ and for some constant c, consider a length-n instance I of the depth $c \log^k n$ straight linear formula problem over $\mathcal{M}(S; +, \cdot)$. By padding the matrices with $0 \in S$ and by inserting identity matrices, I can be transformed in log space (hence uniform NC^2) into another instance J (of the same problem) having depth exactly $\lceil c \log^k n \rceil$, involving matrices of size $n \times n$ and having the property that I is an accepting instance iff J is an accepting instance. Furthermore, formula gates in instance J can be numbered canonically in such a way that all size-n original instances map to the same J (except for variations in the contents of the matrices appearing in J; note that log space suffices—and seems to be required—for the task of determining which matrix corresponds to which indegree-zero gate in J.) Now consider the Boolean circuit defined from J as follows. Transform the formula over $\mathcal{M}(S; +, \cdot)$ in J into a Boolean circuit by replacing each indegree-two gate in the formula with a copy of a polynomial-size, log-depth Boolean circuit which computes the product of $n \times n$ matrices over $(S; +, \cdot)$, and replacing each indegree-zero node with a set of input nodes which give the binary encoding of the corresponding input matrix. Then add a small amount of circuitry in order to take care of testing whether the binary encoding of the matrix computed by the circuit equals the binary encoding of the target matrix in the instance J. This depth $O(\log^{k+1} n)$ Boolean circuit outputs 1 iff J is a positive instance iff I is a positive instance.

We claim that this proves membership in NC^{k+1}. Indeed observe that the "wires" in the Boolean circuit obtained from J above depend only on the length of J (and the length of J depends only on c, k, n). Hence the uniformity log space machine constructs its nth circuit as a two-stage circuit: the first stage transforms I to J, and the second stage *is* the circuit defined from J above. Since the first stage is an NC^2 circuit, the two stages together still form an NC^{k+1} circuit. Now constructing the first stage can be done in log space by the fact that the NC^2 circuit which implements the transformation from I to J is uniform. The second stage is very regular and can thus be constructed in log space as well. ▮

We now describe a method for reduction in the other direction. We simulate *circuits* over a fixed algebra with matrix *straight linear formulas*, by encoding the description of the circuit into the matrices, which we then multiply in a trivial right-to-left order.

Method. Consider a depth D single-output (deterministic) circuit C over an algebra $(S; *)$. Pad the circuit with extra gates in such a way that for any $0 \leqslant d \leqslant D$, there are exactly N gates at depth d, numbered 1 to N, and each takes its inputs from gates at depth $d+1$, except for those at level D which are input gates. The outputs of the gates at level $d+1$ can be represented as a $N \times 1$ column vector V_{d+1}; then we build R and M_d such that $V_d = M_d(RV_{d+1})$. The matrix R is designed such that RV_{d+1} consists of two copies of V_{d+1} stacked one above the other, and M_d is an $N \times 2N$ matrix whose row i encodes the definition of gate i of level d, i.e., where to fetch its first input in the top half of RV_{d+1} and the second in the bottom half. Iterating this process from V_D upwards, we obtain in V_0 the outputs of the gates at depth 0, which coincide with the outputs of the gates in the original circuit. Only one of these outputs is actually of interest to us; we cancel the others by using an extra $N \times N$ matrix G. Hence the evaluation of circuit C over $(S; *)$ is encoded as the straight linear formula $G(M_0(R(\cdots M_{D-1}(RV_D))\cdots)))$ of matrices over an appropriately defined algebra.

Such a level-by-level and gate-by-gate simulation does not exploit the full power of a matrix algebra, however. Indeed, a direct application of this method shows that the depth $O(\log^k n)$ straight linear formula problem over an appropriately chosen $\mathcal{M}(S; +, \cdot)$ is hard for NC^k. However, a refined version of the method, where we represent the circuit with indegree $n^{O(1)}$ gates instead of indegree-two Boolean gates, yields an NC^{k+1}-hardness result for this same problem.

THEOREM 4.3. *Let* $k \geqslant 1$. *There is an algebra* $(S; +, \cdot)$ *such that the depth* $O(\log^k n)$ *straight linear formula problem over* $\mathcal{M}(S; +, \cdot)$ *is* NC^{k+1}*-complete under log space reducibility.*

Proof. We know from Lemma 4.2 that the problem belongs to NC^{k+1}. For the completeness proof, we build in log space a family of straight linear formulas over an appropriately defined algebra $\mathcal{M}(S; +, \cdot)$, which solves the evaluation problem for deterministic Boolean circuits of size $N(n)$ and depth $D(n)$; the formulas will have depth $D(n)/\log n$ and involve matrices of size polynomial in $N(n)$.

We consider a Boolean circuit C, of depth at most D, containing N gates numbered 1 to N, where in particular gate

N outputs the Boolean constant ZERO, and gate 1 gives the output value of C. Build $D+1$ copies of C, labelled C_0 to C_D, and connect the gates according to the rules: (i) if gate g is input to gate h in the original circuit, then gate g_i in copy C_i is input to h_{i-1}, $1 \leqslant i \leqslant D$; (ii) if g is an indegree-zero gate in C and outputs a value v, then g_D is of indegree zero with output v, and for all $i < D$, gate g_i is an indegree two OR with inputs g_{i+1} and N_{i+1}, so that it outputs value v; (iii) if g is an indegree-two gate, then g_D is an indegree-zero gate which outputs an arbitrary constant value.

The circuit thus built is now split into "echelons" of depth $\log n$, where echelon number k consists of those copies numbered $k \log n$ to $(k+1)\log n$ of C, and treats the gates of $C_{(k+1)\log n}$ as if they were its input gates. An echelon can thus be thought of as the NC^1 computation of a function from the echelon's inputs to the echelon's outputs. Such a computation can be broken up into many NC^1 computations of an individual output bit of the echelon. The idea of applying the Barrington construction individually to each output bit of an NC^1-computable function was first used in [10] in the context of many-one NC^1-reductions. We adapt this idea to our context below.

Since all echelons are identical, we consider echelon number 0. Any gate g in echelon 0, together with those gates in copies C_1 to $C_{\log n}$ which contribute to its value, defines a log-depth, single-output Boolean circuit which can be translated into a program over the group $\mathscr{A}_5$, using a method invented by Barrington [3]. (This method is an AC^0 reduction in the case of a single-output, log-depth Boolean circuit which is appropriately presented as a well-balanced formula [5]; here a multiple-output circuit is given by its direct connection language and problems such as deciding which gates participate in the evaluation of which outputs must be solved; in a log-depth circuit, this is feasible in deterministic log space.) Each such program has at most N inputs and length polynomial in N. Through appropriate padding, the programs can be made to do repeated scans of all N possible inputs, performing at most one "real" instruction per scan, and to have the same length $P(N)$, a polynomial in N. In this way all the programs access their N inputs in the same order. Hence a matrix can be defined as having each program in a separate row, in such a way that the column position of an instruction indicates which input is accessed. (This step of making the programs oblivious was not required in [10].) Since an instruction no longer has to specify the input it accesses, the instruction can be encoded simply as an ordered pair of elements of $\mathscr{A}_5$.

We treat these programs as if they were fanin $P(N)$ nondeterministic gates and apply on them our (adapted) simulation method. For this, we define the algebra $(S; +, \cdot)$, as follows:

- $S = \mathscr{A}_5 \cup (\mathscr{A}_5 \times \mathscr{A}_5) \cup \{0, \lambda, \texttt{zero}, \texttt{one}, \langle ? \rangle\}$;
- element 0 is the identity of "$+$" and is absorbing for "$\cdot$," while $\langle ? \rangle$ is absorbing for "$+$";
- operation "$+$" works as the group operation inside $\mathscr{A}_5$ and inside the direct product $\mathscr{A}_5 \times \mathscr{A}_5$;
- `zero+one=one+zero=one+one=one` and `zero+zero=zero`;
- in all other case, "$+$" evaluates to $\langle ? \rangle$;
- with $(a, b) \in \mathscr{A}_5 \times \mathscr{A}_5$, let $(a, b) \cdot \texttt{zero} = a$ and $(a, b) \cdot \texttt{one} = b$;
- let ε denote the identity of $\mathscr{A}_5$; define $(\varepsilon, \varepsilon) \cdot \langle ? \rangle = \varepsilon$, and for all other $(a, b) \in \mathscr{A}_5 \times \mathscr{A}_5$, $(a, b) \cdot \langle ? \rangle = \langle ? \rangle$;
- $\lambda \cdot \texttt{zero}=\texttt{zero}$; $\lambda \cdot \texttt{one}=\texttt{one}$; $\lambda \cdot \langle ? \rangle = \langle ? \rangle$;
- $\texttt{one} \cdot \varepsilon = \texttt{zero}$; for all other elements $\sigma \in \mathscr{A}_5$, we define $\texttt{one} \cdot \sigma = \texttt{one}$.

Element $\langle ? \rangle$ represents an undefined gate output, and $(\varepsilon, \varepsilon)$ is used as a dummy instruction for padding programs. This partial definition of $(S; +, \cdot)$ is all we need for the reduction. It is straightforward to verify that the operation "$+$" is associative.

In each echelon, number the programs 1 to N in a manner consistent with the numbering of the corresponding output gates. Let $\Delta = D/\log n$, which we assume to be an integer. For a given echelon k, $0 \leqslant k < \Delta$, let the $N \times 1$ column vector V_{k+1} contain the inputs to this echelon, i.e., the value of each gate of C at the beginning of the evaluation of the circuit if $k = \Delta$ and the outputs of the previous echelon otherwise. Each value is one of `zero`, `one` ("output defined" as the Boolean values ZERO or ONE, respectively), or $\langle ? \rangle$ ("output undefined"). Define also a $P(N) \times N$ matrix R by piling up $P(N)/N$ square matrices containing λ on the diagonal and 0 elsewhere; an $N \times P(N)$ matrix M, where row g contains the $P(N)$ instructions of the program for gate g, $1 \leqslant g \leqslant N$, as built above; and an $N \times N$ matrix F, with entries `one` on the diagonal and 0 everywhere else. We claim that the formula $V_k = F(M(R(V_{k+1})))$, depicted below, simulates the work of echelon k:

$$\begin{bmatrix} \texttt{one} & & 0 \\ & \ddots & \\ 0 & & \texttt{one} \end{bmatrix} \left(\begin{bmatrix} \vdots & & \vdots \\ M[g,1] \leftarrow \text{program for gate } g & \cdots & \rightarrow M[g,P(N)] \\ \vdots & & \vdots \end{bmatrix} \left(\begin{bmatrix} \lambda & & 0 \\ & \ddots & \\ 0 & & \lambda \\ \vdots & \vdots & \vdots \\ \lambda & & 0 \\ & \ddots & \\ 0 & & \lambda \end{bmatrix} \left(\begin{bmatrix} V_{k+1}[1] \\ \vdots \\ V_{k+1}[N] \end{bmatrix} \right) \right) \right)$$

Indeed recall that V_{k+1} contains the N "inputs" to echelon k. The first product in the evaluation of $F(M(R(V_{k+1})))$ yields a matrix RV_{k+1} which is a $P(N) \times 1$ column vector made up of $P(N)/N$ copies of V_{k+1}. Then $M(RV_{k+1})$ is a $N \times 1$ column vector containing the outputs of the N programs, in the form of either $\langle ? \rangle$ (program has read an undefined input), or ε (program outputs the bit ZERO), or another element of $\mathscr{A}_5$ (program outputs ONE). In order to encode the outputs of the echelon in the same format as its inputs, we multiply on the left with matrix F. Thus we obtain the output of each gate of the original circuit C by evaluating the formula $F(M(R(\cdots(F(M(R(V_\Delta)))\cdots))))$. Actually, we are only interested in the value of gate number 1; we therefore append to the left of the formula an $N \times N$ matrix G, such that $G_{11} = \lambda$ and all other entries are 0. Hence we have reduced in log space the computation of a circuit of depth D with N gates to a formula

$$G(F(M(R(\cdots(F(M(R(V_\Delta))))\cdots)))),$$

involving $3\Delta + 1$ gates and 5 matrices with at most $P(N)$ rows and $P(N)$ columns. With $D \in O(\log^k n)$ and $P(N) \in n^{O(1)}$, we obtain our result. ∎

The depth $O(\log^k n)$ straight linear formula problem is a very restricted subcase of the depth $O(\log^k n)$ deterministic circuit problem, which we now show also belongs to NC^{k+1}. Thus we obtain a characterization of NC^{k+1} in terms of circuits over an appropriately chosen algebra $\mathscr{M}(S; +, \cdot)$.

THEOREM 4.4. *For any algebra $(S; +, \cdot)$ and $k \geq 1$, the depth $O(\log^k n)$ deterministic circuit problem over $\mathscr{M}(S; +, \cdot)$ belongs to NC^{k+1}.*

Proof. The evaluation of an $O(\log^k n)$ depth circuit C over $\mathscr{M}(S; +, \cdot)$ is done in two steps. First, we replace locally each gate of C with a log-depth Boolean circuit computing the product of two matrices, which yields a polynomial-size Boolean circuit of depth $O(\log^{k+1} n)$. Next, using Theorem 4.3, we encode in log space this Boolean circuit as a depth $O(\log^k n)$ straight linear formula F over some algebra $\mathscr{M}(T; \oplus, \odot)$. Through a sequence of two log space reductions, we thus have reduced the evaluation of C to the evaluation of F, a problem which we have shown to be in NC^{k+1}. Since log space reducibility is transitive, we obtain our result. ∎

We now apply our techniques to cases defined by restricting the algebraic properties of the monoid $(S; +)$. This leads to a characterization of complexity classes defined by circuits with various types of gates, namely AC^k, $ACC^k(q)$, ACC^k, and $CC^k(q)$, where $k \geq 1$ and $q \geq 2$ are fixed integers. Definitions for these classes are given in Section 2.

THEOREM 4.5. *Let $k \geq 1$.*

(a) *For any algebra $(S; +, \cdot)$ in which $(S; +)$ is an aperiodic monoid, the depth $O(\log^k n)$ deterministic circuit problem over $\mathscr{M}(S; +, \cdot)$ belongs to AC^k. There is an algebra $(S; +, \cdot)$, where $(S; +)$ is an aperiodic monoid, for which the depth $O(\log^k n)$ straight linear formula problem over $\mathscr{M}(S; +, \cdot)$ is AC^k-complete under NC^1 reducibility.*

(b) *For any algebra $(S; +, \cdot)$ in which $(S; +)$ is a solvable monoid, the depth $O(\log^k n)$ deterministic circuit problem over $\mathscr{M}(S; +, \cdot)$ belongs to ACC^k. For every fixed integer $q \geq 2$, there is an algebra $(S; +, \cdot)$, where $(S; +)$ is a solvable monoid of exponent q, for which the depth $O(\log^k n)$ straight linear formula problem over $\mathscr{M}(S; +, \cdot)$ is complete for $ACC^k(q)$ under NC^1 reducibility.*

(c) *For any algebra $(S; +, \cdot)$ in which $(S; +)$ is a cyclic group of order q for a fixed $q \geq 3$, the depth $O(\log^k n)$ deterministic circuit problem over $\mathscr{M}(S; +, \cdot)$ belongs to $CC^k(q)$. There is an algebra $(S; +, \cdot)$, where $(S; +)$ is a cyclic group of order q, $q \geq 3$, for which the depth $O(\log^k n)$ straight linear formula problem over $\mathscr{M}(S; +, \cdot)$ is complete for $CC^k(q)$ under NC^1 reducibility.*

Proof. We reason in the same way as above, working first on straight linear formulas of matrices and then extending to circuits of arbitrary shape.

We begin with result (a). We first use the characterization of (non-uniform) AC^0 in terms of families of polynomial-length programs over aperiodic monoids [8] and the fact that the word problem in aperiodic monoids belongs to uniform AC^0 [5], to obtain results similar to Proposition 4.1 and Lemma 4.2, i.e., the first statement of (a) restricted to straight linear formulas. Then we prove a result analogous to Theorem 4.3, using the same proof with the following adaptations: we use echelons of depth one, each consisting of N Boolean gates plus N input gates; then we translate the echelon into N programs over an appropriately defined aperiodic monoid (instead of the $\mathscr{A}_5$ of 4.3), and we verify that the resulting monoid $(S; +)$ is also aperiodic. Note that with echelons of depth one, there is no need to do a search for ancestors, as in 4.3, so that the reduction is NC^1. We then apply the argument given in Theorem 4.4 to complete the proof.

Result (b) is proved in a similar manner, using the characterization of ACC^0 in terms of programs over solvable monoids [8, 5].

For the first statement of result (c), observe that computing the product of two matrices M and N consists of first computing the values $M_{ij} \cdot N_{jk}$ for all combinations of i, j, and k, which is feasible in constant depth via table look-up, since we work in a fixed-size structure, and then in computing additions modulo q, one for each entry of MN. The additions are performed by turning the input values into

unary and feeding them in parallel to q MOD_q gates (each of which determines whether the sum modulo q takes one of q possible values), followed by a constant-depth circuit which merges the results and translates them into the format required for the next matrix product. Now applying the method used to prove Lemma 4.2 and noting that log space (apparently required by this method to translate the original problem instances into canonical form) is included in $CC^1(q)$, we obtain a result that is analogous to 4.2 for straight linear formulas of restricted depth.

In the other direction, we have to justify that the algebra of statement (c), i.e. a set of size q with addition modulo q and another operation "$\cdot$," is powerful enough to characterize $CC^k(q)$. We adapt for this the method described informally before Theorem 4.3, and as discussed in Section 2.3 we assume with no loss of generality that $CC^k(q)$ circuits for $q \geqslant 3$ consist exclusively of MOD_q gates. We work on a circuit C of depth D with N gates, where gate number 0 is the output gate. We build a straight linear formula $F(M(\cdots F(MV_D)\cdots))$, where V_D contains the values of the gates of copy C_D, and each subformula "$F(M(\cdots))$" simulates the work of a copy of C. Let $S = \{0, 1, ..., q-1\}$, where $q \geqslant 3$. The operation "$+$" acts on S as the addition modulo q. For our purposes, it suffices to specify "$\cdot$" in such a way that 0 is absorbing and that $x \cdot y = x$ for all x and all $y \neq 0$.

A bit with value ZERO (resp. ONE) is represented by the element 0 (resp. 1). Let F be an $N \times N$ matrix containing value 1 on the diagonal and 0 everywhere else, and let M be an $N \times N$ matrix, filled with values from S in such a way that if gate g uses the output of gate h x times, then the entry g, h of M has value x. Then with the outputs of copy C_i encoded in V_i, the product MV_i computes the arithmetic sums of the inputs to the gates of C_{i-1}, and the product on the left with F translates the results into 0's and 1's. ∎

Note that statement (c) is written with $q \geqslant 3$. Regarding the $q = 2$ case, we first recall that the last part of the proof of (c) is written in terms of circuits consisting exclusively of MOD_q gates; but the problem of evaluating a circuit consisting exclusively of *XOR* gates, of arbitrary depth, is complete for complexity class $\oplus L$ [9], so that the insertion of constant fanin *AND*s and *OR*s between the MOD_q gates does seem to make a difference when $q = 2$. Meanwhile, the constraints stated at the beginning of this section, that the element 0 be the identity for "$+$" and absorbing for "$\cdot$," imply that only two possible algebras $(S; +, \cdot)$ can be built when $q = 2$, namely $(\{0, 1\}; XOR, AND)$ and $(\{0, 1\}; XOR, \circ)$, where the operation "$\circ$" always yields value 0. In both cases, the related matrix algebras are associative; the word problem for the latter is trivial, while for the former it is complete for $\oplus L$ [11]. Finally, defining subcases for the $q = 2$ case, by restricting circuit depth or the number of matrices, no longer seems to yield a correspondence with a natural chain of complexity classes.

Combining our simulation method with the *P*-completeness of the Boolean circuit problem [25] leads immediately to the following.

PROPOSITION 4.6. *For any algebra $(S; +, \cdot)$, the deterministic circuit problem over $\mathcal{M}(S; +, \cdot)$ belongs to P. There is an algebra $\mathcal{M}(S; +, \cdot)$ over which the straight linear formula problem is P-complete.*

4.2. *Matrix Size*

In this subsection we look at the evaluation of deterministic circuits where restrictions have been put on the size of the matrices. We obtain characterizations in terms of matrix formulas for classes defined by polynomial-size Boolean circuits with restriction on width, or alternately by Turing machines with simulations restrictions on worktime and workspace. More precisely, we characterize the class *SC*, defined in terms of uniform Boolean circuits as SIZE-WIDTH$[n^{O(1)}, \log^{O(1)} n]$, and in terms of deterministic Turing machines as TIME-SPACE$[n^{O(1)}, \log^{O(1)} n]$ (see [31]).

Interestingly, the correspondence seems much tighter between time and space bounded Turing machines and formulas of matrices with bounded size (Proposition 4.9), than between width-restricted Boolean circuits and formulas of matrices with bounded size. The next two propositions illustrate what we could do with the latter.

PROPOSITION 4.7. *For any algebra $(S; *)$ and $D(n) \in \Omega(\log n)$, the depth $D(n)$, width $Z(n)$ deterministic circuit problem over $(S; *)$ reduces in space $D(n)$ to the evaluation of a depth $O(D(n))$ linear formula over $\mathcal{M}(S; +, \cdot)$, involving square matrices of size $4Z(n)^2$.*

Proof. We follow almost exactly the method described before Theorem 4.3. Since we are not interested in transporting the input values from a depth level to another, we apply the method only on the non-input gates, at most $Z(n)$ of them per depth level, which enables us to construct matrices of the appropriate size. The input values used by gates at level i are read during the reduction and encoded into the matrix M_i as if they were constants. ∎

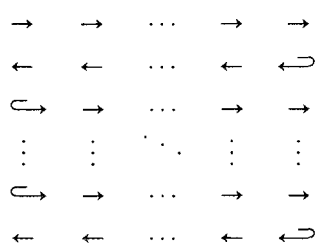
PROPOSITION 4.8. *For any algebra $(S; +, \cdot)$, the depth $D(n)$ linear formula problem over $\mathcal{M}(S; +, \cdot)$, involving $D(n)$ gates and square matrices of size $Z(n)$, reduces in log space to the depth $O(D(n)\sqrt{Z(n)})$ and width $O(Z(n)^{3/2})$ deterministic circuit problem over $(S; +, \cdot)$.*

Proof. Simply replace each gate in the circuit over $\mathcal{M}(S; +, \cdot)$ with a circuit over $(S; +, \cdot)$ which computes the product of two matrices. ∎

PROPOSITION 4.9. *Let $L(n) \in \Omega(\log n)$ and $T(n) \in \Omega(n)$ be space constructible functions. The computation of a* TIME-SPACE$[T(n), L(n)]$ *Turing machine can be* $\log(T(n)\,L(n))$ *space reduced to the evaluation of a depth*

$O(nT(n))$ linear formula over $\mathcal{M}(S; +, \cdot)$, involving matrices of size $O(L(n))$, where $(S; +, \cdot)$ is an appropriately chosen algebra.

Proof. We consider a Turing machine with time bound $T(n)$ and a single worktape of length $L(n)$. We assume that the machine is oblivious, that it can recognize both ends of its worktape, and that it has a unique accepting configuration. Such a machine can be built from an arbitrary machine at the cost of an $O(n)$ increase in running time, and the addition of extra tapes of length $O(\log n)$, which can then be merged with the original worktape (see [2 or 22] for details); the lower bounds on $L(n)$ and $T(n)$, and the "$O(nT(n))$" in the statement of the proposition come from this. Let $Z = \lceil \sqrt{L(n)} \rceil$. We encode a configuration of the machine as $Z \times Z$ matrix over an algebra $(S; +, \cdot)$, with rows and columns numbered 1 to Z, in which the worktape is folded starting at position 1, 1 (the upper left corner), running back and forth through columns 1 to Z, down to row Z, as follows (the picture assumes that Z is even):



The encoding is made with an algebra $(S; +, \cdot)$, where the operations "$+$" and "$\cdot$" are partially defined by "rules" which we introduce as we proceed to describe how the machine's computation is simulated. We build "$+$" in such a way that it is partially associative, in that all the expressions we deal with can be written without parentheses; we leave aside the tedium of verifying that "$+$" is indeed an associative operation. Meanwhile, let $l \in \{T, B, O\}$, $c \in \{L, R, O\}$, $\circ \in \{\triangleright, \triangleleft\}$ be indices. The set S consists of:

- for each combination of tape character σ and indices l and c, elements $\langle \sigma_{lc} \rangle$, $\langle \sigma_{lc}]$, $[\sigma_{lc}\rangle$, and $[\sigma_{lc}]$;
- for each combination of machine state q and index $\circ$, elements $\langle q \circ]$ and $[q \circ \rangle$;
- for each combination of $q, \sigma, l, c, \circ$, elements $\langle q \circ \sigma_{lc} \rangle$, $\langle q \circ \sigma_{lc} \Downarrow \rangle$, and $\langle q \circ \sigma_{lc} \Uparrow \rangle$;
- for each input character a, elements $\mathtt{zero}_a$ and $\mathtt{one}_a$;
- elements $\langle 0 \rangle$, η, ω, $\perp$, and 0.

Element 0 is absorbing for "$\cdot$" and an identity for "$+$," while $\perp$ is absorbing for "$+$." (Note that we could do away with a few elements of S at the expense of more complicated arguments to take care of special cases.)

Let M_{t-1} encode the configuration of the machine after $t-1$ steps. Each entry of M_{t-1} encode a worktape cell with an element of S. If the cell contains a tape character σ, and the worktape head does not point to it, then the entry is $\langle \sigma_{lc} \rangle$, where the index $l \in \{T, B, O\}$ indicates whether the cell belongs to the tape segment encoded in the top row of the matrix (resp. the bottom row, any other row), and $c \in \{L, R, O\}$ indicates whether the entry lies on the leftmost column of the matrix (resp. the rightmost column, any other column). In the case where the worktape head points to the cell, then an element of the form $\langle q \circ \sigma_{lc} \rangle$ is used, where q is the state of the machine, σ is the content of the cell, indices l and c are used as above, and $\circ \in \{\triangleright, \triangleleft\}$ indicates whether the cell is located on an odd-numbered row (if $\circ = \triangleright$) or on an even-numbered row ($\circ = \triangleleft$; the triangle actually points to the right-direction of the tape).

We simulate the tth step of the execution by computing the matrix $M_t = D(M_{t-1} B_t)$. First, we design B_t and $(S; +, \cdot)$ in such a way that the entries of $(M_{t-1} B_t)$ are identical to those of M_{t-1}, except at the vicinity of the position of the read–write head. Our method is based on the property that M_{t-1} contributes to entry i, j of the product $(M_{t-1} B_t)$ only through its row i. Let a be the input accessed at the tth step (a is known in advance since the machine is oblivious); then define B_t to contain element $\mathtt{one}_a$ on the diagonal, and everywhere else element $\mathtt{zero}_a$.

First, we are interested in conserving intact the content of the tape at all positions away from the read–write head. For this, we define the rules

$$\langle \sigma_{lc} \rangle \cdot \mathtt{zero}_a = \langle 0 \rangle$$
$$\langle \sigma_{lO} \rangle \cdot \mathtt{one}_a = [\sigma_{lO}],$$
$$\langle \sigma_{lR} \rangle \cdot \mathtt{one}_a = [\sigma_{lR}\rangle, \quad \langle \sigma_{lL} \rangle \cdot \mathtt{one}_a = \langle \sigma_{lL}].$$

(Observe that the rules are not concerned with a transition of the machine, and thus they are independent of the input character a.) In the case when row i of M_{t-1} does not contain the read–write head, the entries on row i of $(M_{t-1} B_t)$ look as follows (here "$\cdots$" represents zero or more occurrences of $\langle 0 \rangle$):

Column	Expression for the entry
1	$\langle \sigma_{lL}] + \cdots + \langle 0 \rangle$
2 to $Z-1$	$\langle 0 \rangle + \cdots + [\sigma_{lO}] + \cdots + \langle 0 \rangle$
Z	$\langle 0 \rangle + \cdots + [\sigma_{lR}\rangle$

We then define rules which allow these expressions to correctly evaluate to $\langle \sigma_{lc} \rangle$ for all $c \in \{L, R, O\}$. The reader can also verify that we are justified in writing the above expressions without parentheses:

$$\langle 0 \rangle + \langle 0 \rangle = \langle 0 \rangle$$
$$\langle 0 \rangle + [\sigma_{lc}] = \langle 0 \rangle + \langle \sigma_{lc}] = \langle \sigma_{lc}]$$
$$[\sigma_{lc}] + \langle 0 \rangle = [\sigma_{lc}\rangle + \langle 0 \rangle = [\sigma_{lc}\rangle$$
$$\langle 0 \rangle + \langle \sigma_{lc} \rangle = \langle 0 \rangle + [\sigma_{lc}\rangle = \langle \sigma_{lc}\rangle + \langle 0 \rangle$$
$$= \langle \sigma_{lc}] + \langle 0 \rangle = \langle \sigma_{lc} \rangle.$$

Consider now the row where the read–write head is located. At a given transition, either the head stays immobile, or it moves one position. In the former case, we deal with a transition reading σ while on state p, and changing the state q while writing τ, as follows:

$$\langle p \circ \sigma_{lc} \rangle \cdot \mathtt{zero}_a = \langle 0 \rangle, \quad \langle p \circ \sigma_{lc} \rangle \cdot \mathtt{one}_a = \langle q \circ \tau_{lc} \rangle$$
$$\langle q \circ \tau_{lc} \rangle + \langle 0 \rangle = \langle 0 \rangle + \langle q \circ \tau_{lc} \rangle = \langle q \circ \tau_{lc} \rangle.$$

Combining this with the above rules, we verify that row i of $(M_{t-1}B_t)$ encodes correctly the ith segment of the tape, after the transition.

If a transition requires a move of the read–write head, this is one position to the right or to the left on the tape, and is represented as a move to the right or to the left in the matrix, depending on the row in the matrix where the cell is encoded. We describe how to deal with a move one position to the right on the tape from a cell lying on an odd-numbered row of the matrix, ie., where the right-direction of the tape coincides with the right-direction in the matrix; all other possibilities can be treated dually. There are three cases, depending on whether the cell is in the rightmost column, the leftmost column, or elsewhere. We begin with this last case: the head moves from position i, j to position i, $(j+1)$, where $1 < j < Z$. We define the following rules for a transition reading ρ while on state p, writing τ and changing to state q while moving one cell to the right:

$$\langle p \rhd \rho_{lO} \rangle \cdot \mathtt{one}_a = \langle \tau_{lO} \rangle, \quad \langle p \rhd \rho_{lO} \rangle \cdot \mathtt{zero}_a = \langle q \rhd].$$

The entries on row i of $(M_{t-1}B_t)$ then look as follows:

Column	Expression for the entry
1	$\langle \sigma_{lL}] + \cdots + \langle q \rhd] + \cdots + \langle 0 \rangle$
2 to $j-1$	$\langle 0 \rangle + \cdots + [\sigma_{lO}] + \cdots + \langle q \rhd] + \cdots + \langle 0 \rangle$
j	$\langle 0 \rangle + \cdots + \langle \tau_{lO} \rangle + \cdots + \langle 0 \rangle$
$j+1$ to $Z-1$	$\langle 0 \rangle + \cdots + \langle q \rhd] + \cdots + [\sigma_{lO}] + \cdots + \langle 0 \rangle$
Z	$\langle 0 \rangle + \cdots + \langle q \rhd] + \cdots + [\sigma_{lR}\rangle$

We can already observe that, by the rules defined above, the entry at column j correctly evaluates to $\langle \tau_{lO} \rangle$. For the other entries, we define some more rules:

$$\langle q \rhd] + \langle 0 \rangle = \langle 0 \rangle, \quad \langle 0 \rangle + \langle q \rhd] = \langle q \rhd]$$
$$\langle q \rhd] + \langle \sigma_{lc} \rangle = \langle q \rhd] + \langle \sigma_{lc}] = \langle \sigma_{lc} \rangle$$
$$\langle \sigma_{lc}] + \langle q \rhd] = \langle \sigma_{lc} \rangle + \langle q \rhd] = [\sigma_{lc} \rangle + \langle q \rhd]$$
$$= \langle \sigma_{lc} \rangle$$
$$\langle q \rhd] + [\sigma_{lc} \rangle = \langle q \rhd] + [\sigma_{lc}] = \langle q \rhd \sigma_{lc} \rangle$$
$$\langle q \rhd \sigma_{lc} \rangle + \langle 0 \rangle = \langle 0 \rangle + \langle q \rhd \sigma_{lc} \rangle = \langle q \rhd \sigma_{lc} \rangle.$$

This ensures that the term $\langle q \rhd]$ is "eliminated" everywhere but on column $j+1$, where we correctly obtain an entry of the form $\langle q \rhd \sigma_{lc} \rangle$. In order to deal with the $j = 1$ case, it suffices to add to the above the following rules:

$$\langle p \rhd \rho_{lL} \rangle \cdot \mathtt{one}_a = \langle \tau_{lL} \rangle, \quad \langle p \rhd \rho_{lL} \rangle \cdot \mathtt{zero}_a = \langle q \rhd].$$

The last case we treat is a motion to the right from position i, Z. Note that, on the tape, the right neighbour of the cell encoded at this position is located at position $(i+1), Z$; we cannot transfer the read–write head to this position with the product $(M_{t-1}B_t)$, since the entry $(i+1), Z$ of this matrix is independent of the content of row i of M_{t-1}. Instead, we use the following rules to ensure that $(M_{t-1}B_t)_{i,Z} = \langle q \rhd \tau_{lR} \Downarrow \rangle$, an element of S which encodes that the tape content has been correctly updated to τ, and that we still have to move the head one row down in the matrix:

$$\langle p \rhd \rho_{lR} \rangle \cdot \mathtt{one}_a = \langle q \rhd \tau_{lR} \Downarrow \rangle,$$
$$\langle p \rhd \rho_{lR} \rangle \cdot \mathtt{zero}_a = \langle 0 \rangle$$
$$\langle 0 \rangle + \langle q \rhd \tau_{lR} \Downarrow \rangle = \langle q \rhd \tau_{lR} \Downarrow \rangle.$$

The rest of the work, a head motion without modification of the tape, is simulated with a product to the left with the matrix D, which contains value η on the diagonal and ω everywhere else. By an observation dual to the one on which the mechanisms described up to now are based, namely that a matrix V contributes to entry i, j of the product UV only through its column j, we see that we can encode the vertical motion of the head with the product $D(M_{t-1}B_t)$ by further extending the definition of $(S; +, \cdot)$ with rules analogous to the above (in which the index $l \in \{T, B, O\}$ comes into play); these rules ensure that, whenever necessary, the head moves up or down one row, while the rest of the encoding is unchanged. Therefore the Turing machine accepts its input iff the formula

$$D(\cdots D((D(M_0\, B_1))\, B_2) \cdots B_{T(n)})$$

evaluates to a matrix M_{accept}, which encodes the machine's accepting configuration. ∎

THEOREM 4.10. *Let $k \geq 1$. For any algebra $(S; +, \cdot)$, the linear formula problem over $\mathcal{M}(S; +, \cdot)$ involving matrices of size $O(\log^k n)$ belongs to SC^k. There is an algebra $(S; +, \cdot)$ over which this problem is complete for SC^k under NC^1 reducibility.*

Proof. For this statement, observe that, since the formula is linear, only one matrix has to be kept in memory as an intermediate result. For the second statement, observe that the input for a decision problem in SC^k can, through

an NC^1 reduction, be reencoded in binary and appended to the binary description of a TIME-SPACE$[n^{O(1)}, \log^k n]$ Turing machine M which solves this problem. This transformed input is then fed to a universal Turing machine which simulates M on its input, using $O(\log^k n)$ workspace (this is done at the cost of a polynomial increase in the running time). Then Proposition 4.9 is applied to this universal machine. Also, observe that if $T(n) \in n^{O(1)}$ and $L(n) \in \log^{O(1)} n$, then the $\log(T(n)\, L(n)) \in O(\log n)$ space reduction described above can be rewritten as an NC^1 reduction, a detail relevant for the $k = 1$ case. ∎

4.3. *Nondeterministic Matrix Circuits*

In this subsection, we consider nondeterministic restrictions to the evaluation problem. Recall the discussion in Section 2: nondeterminism can appear when two matrices are multiplied (we speak of a multiple-valued product), or when a gate in the circuit has fanin three or more. We begin with the multiple-valued matrix product.

PROPOSITION 4.11. *The multiple-valued product of two matrices can be computed in LOGCFL. There is an algebra $(S; +, \cdot)$ for which the problem is complete for LOGCFL under AC^0 reducibility. There is also an algebra for which the problem is complete for NL under AC^0 reducibility.*

Proof. The *LOGCFL* upper bound is clear since we deal with just one matrix product; computing each matrix entry reduces to solving an instance of the word problem over the groupoid $(S; +)$. To prove the hardness claims, we reduce as in the proof of Proposition 4.1, that is, we format an instance of a word problem over a fixed algebra into an instance of the form $MN = P$. The arguments concerning uniformity are transported here as well. For the *LOGCFL*-completeness part we define $(S; +)$ as the groupoid G_{CFL}, over which the word problem *LOGCFL*-complete. The *NL*-completeness statement concerns groupoid G_{NL}, over which the word problem is *NL*-complete. See [10] for definitions of these groupoids and completeness proofs. ∎

PROPOSITION 4.12. *Assuming the multiple-valued matrix product, the evaluation problem for a circuit with matrix inputs belongs to NP. There is an algebra $(S; +, \cdot)$ such that the evaluation problem for the fixed formula $A(BC)$, where A, B, and C are matrices over $(S; +, \cdot)$, is NP-complete under log space reducibility.*

Proof. The *NP* upper bound is obtained by guessing a bracketing for each entry in each matrix product and evaluating the resulting circuit deterministically. To prove *NP*-hardness, we reduce from the problem CNF-SAT [19]. Consider a CNF-SAT instance with m clauses and v variables. We define an $m \times v$ matrix A, a $v \times 3$ matrix B, and a 3×1 matrix C over a structure $(\{\texttt{zero}, \texttt{one}, \texttt{true}, \texttt{false}, \perp\}; +, \cdot)$ having the property that $A(BC)$ evaluates to the all-`true` vector iff the CNF-SAT instance is satisfiable. As usual, the element 0 acts as an identity for "+" and is absorbing for "·." We define the operation "+" on {`zero`, `one`} as a *NAND* and on {`true`, `false`} as *OR*; otherwise "+" evaluates to $\perp$. We define the operation "·" on the nonzero values with the following table.

	Zero	One	False	True	$\perp$
Zero	False	False	$\perp$	$\perp$	$\perp$
One	Zero	One	$\perp$	$\perp$	$\perp$
False	True	False	$\perp$	$\perp$	$\perp$
True	False	True	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Set

$$C = \begin{pmatrix} \texttt{zero} \\ \texttt{one} \\ \texttt{one} \end{pmatrix}$$

and fill B everywhere will the value `one`. Given that `(zero + one) + one = zero` and `zero + (one + one) = one`, the product BC nondeterministically produces a $v \times 1$ column vector filled with `zeros` and `ones`. This vector provides the guesses for the values of the Boolean variables satisfying the CNF-SAT formula. Now define each entry a_{ij} in A, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant v$, as

$$a_{ij} = \begin{cases} \texttt{true} & \text{if } j \text{ appears unnegated in clause } i \\ \texttt{false} & \text{if } j \text{ appears negated in clause } i \\ \texttt{zero} & \text{if } j \text{ does not appear in clause } i. \end{cases}$$

Now, assuming without loss of generality that no variable appears twice in any given clause, the ith entry in the m-vector $A(BC)$ evaluates to `true` whenever the Boolean values given by BC satisfy clause i. ∎

Note that we can relax the condition that the expression ABC be parenthesized as $A(BC)$; a minor addition to the definition of $(S; +, \cdot)$ in the proof of Proposition 4.12 could prevent the product $(AB)\, C$ from evaluating to anything but the all-$\perp$ vector. Hence we have shown that the *word* problem involving three matrices, assuming the multiple-valued matrix product, is also *NP*-complete.

Interestingly, if we place restrictions on the size of the matrices, we can use the multiple-valued variant of the formula evaluation problem to characterize another chain of significant complexity classes. We do this with a nondeterministic counterpart to Proposition 4.9.

PROPOSITION 4.13. *Let $L(n) \in \Omega(\log n)$ and $T(n) \in \Omega(n)$ be space constructible functions. The computation of a* NTIME-SPACE$[T(n), L(n)]$ *Turing machine can be* $\log(T(n)\, L(n))$ *space reduced to the evaluation of a depth*

$O(nT(n))$ multiple-valued linear formula over $\mathcal{M}(S; +, \cdot)$, involving matrices of size $O(L(n))$, where $(S; +, \cdot)$ is an appropriately chosen algebra.

Proof. Let the Turing machine be as in the proof of Proposition 4.9; we use the exact same technique to simulate its computation with matrices. Without loss of generality, we can restrict nondeterminism to transitions which do not involve head motions and pick one of two possible actions. The algebra $(S; +, \cdot)$ of 4.9 can then be modified as follows. For each transition which, upon reading character σ in state p, either writes τ and enters state q, or writes τ' and enters q', and for each combination of indices l, c, and $\circ \in \{\rhd, \lhd\}$, define an element $\langle q \circ \tau \,?\, q' \circ \tau' \rangle$, and the rules

$$\langle p \circ \sigma_{lc} \rangle \cdot \mathtt{zero}_a = \langle 0 \rangle,$$
$$\langle p \circ \sigma_{lc} \rangle \cdot \mathtt{one}_a = \langle q \circ \tau_{lc} \,?\, q' \circ \tau'_{lc} \rangle$$
$$\langle 0 \rangle + \langle q \circ \tau_{lc} \,?\, q' \circ \tau'_{lc} \rangle = \langle q \circ \tau_{lc} \rangle$$
$$\langle q \circ \tau_{lc} \,?\, q' \circ \tau'_{lc} \rangle + \langle 0 \rangle = \langle q' \circ \tau'_{lc} \rangle.$$

Note that this scheme does not work in cells located on the left- or rightmost columns of the matrix. To deal with this problem, we modify the Turing machine prior to its simulation, by padding the worktape with cells containing a special character on which the machine just skips without making nondeterministic choices, in such a way that the left- and rightmost columns now contain this type of cell. Observe that in the simulation, only one entry of the product $(M_{t-1}B_t)$ is evaluated nondeterministically. ∎

THEOREM 4.14. *Let $k \geq 1$. For any algebra $(S; +, \cdot)$, the linear formula problem, involving matrices over $(S; +, \cdot)$ of size $O(\log^k n)$ and multiple-valued products, belongs to NSC^k. There is an algebra $(S; +, \cdot)$ over which this problem is complete for NSC^k under NC^1 reducibility.*

Proof. Every entry in a multiple-valued matrix product can be evaluated in turn by first guessing a parenthesization for its expression, using $O(\log^k n)$ characters, and then evaluating deterministically. The rest of the proof follows Theorem 4.10. ∎

We now state a result on circuits with nondeterministic gates, a counterpart to Proposition 4.12.

PROPOSITION 4.15. *The problem of evaluating a nondeterministic matrix circuit belongs to NP. There is an algebra $\mathcal{M}(S; +, \cdot)$ over which the nondeterministic formula problem is NP-complete under log space reducibility.*

Proof. The *NP* upper bound is clear. To prove *NP*-hardness, we reduce from an instance of problem CNF-SAT involving m clauses and v Boolean variables. Let $S = \{0, \mathtt{true}, \mathtt{false}, \langle ? \rangle, \Delta, \bot\}$. The operation "+" is defined as *OR* on $\{\mathtt{true}, \mathtt{false}\}$ and evaluates to $\bot$ with any other two nonzero arguments. Meanwhile, we define an operation "$\cdot$" as follows:

	$\langle ? \rangle$	Δ	false	true	$\bot$
$\langle ? \rangle$	$\bot$	true	$\bot$	$\bot$	$\bot$
Δ	false	Δ	false	true	$\bot$
false	$\bot$	false	true	false	$\bot$
true	$\bot$	true	false	true	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

The problem CNF-SAT is simulated by asking whether $T = AV$, where T is a $m \times 1$ all-true column vector, where V is a $v \times 1$ vector which will be made to contain values guessed from $\{\mathtt{true}, \mathtt{false}\}$ for each variable, and where matrix entry a_{ij} in A, $1 \leq i \leq m$, $1 \leq j \leq v$, is defined as

$$a_{ij} = \begin{cases} \mathtt{true} & \text{if variable } j \text{ appears unnegated in clause } i \\ \mathtt{false} & \text{if variable } j \text{ appears negated in clause } i \\ 0 & \text{if variable } j \text{ does not appear in clause } i. \end{cases}$$

Matrix V is computed by the nondeterministic formula $V = ((ID_1 I) \cdots (ID_v I))\, F$, where I is a $v \times v$ matrix containing Δ on the diagonal and 0 everywhere else, and D_i is identical, except at position i, i, where the entry is $\langle ? \rangle$. Note that evaluating $ID_i I$ yields a diagonal matrix with Δ everywhere on the diagonal, except at position i, i, whose content is true or false, according to whether $ID_i I$ is evaluated as $I(D_i I)$ or as $(ID_i) I$. The subexpressions $ID_i I$, $1 \leq i \leq v$, commute between themselves, so that they can be evaluated in any order. Their product is a $v \times v$ matrix with, on the diagonal, the values assigned to the Boolean variables; in order to obtain the $v \times 1$ column vector V, we multiply this matrix on the right with a $v \times 1$ column vector F whose entries are all Δ. The reader can verify that the nondeterministic formula $A(((ID_1 I) \cdots (ID_v I))\, F)$ can be built in log space. ∎

We note that the nondeterministic formula involved in this proof has constant depth. Applying the argument of Proposition 3.4 to algebras of matrices, we can adapt the proof to obtain an *NP*-completeness statement for the *word* problem over such algebras, An *NP*-completeness result for the word problem over exponential-size groupoids (with a different representation) has been obtained independently by Muscholl [30].

5. CONCLUSION

We have generalized the circuit evaluation problem to include the evaluation of unbounded fanin circuits with nonassociative gates. Such gates in effect provide nondeterminism, and several variants of the evaluation problem arise. In most cases the operands in our evaluation problems are groupoid elements, either drawn from a fixed

algebraic structure $(S; +, \cdot)$ or from the nonassociative groupoid $\mathcal{M}(S: +, \cdot)$ of matrices with entries from $(S; +, \cdot)$ (with the consequence that length-n inputs could encode groupoid elements drawn from a groupoid of size exponential in n).

Our results extend the known complexity results concerning the evaluation of words, formulas, and circuits over various algebraic structures. It now follows that suitable restrictions to one and the same general evaluation problem are complete for (and thus capture) the fundamental subclasses of NP, in particular the various "parallel complexity" classes within NC (albeit with the need for an occurrence of the bound $O(\log^k n)$ as part of the problem definition in some cases).

Our results are *descriptive* in the sense that they characterize known complexity classes without suggesting separation arguments. However, could manageable new lower bound techniques perhaps be developed to investigate the complexity of our evaluation problems in the context of a severely restricted fixed algebra $(S; +, \cdot)$ and its resulting matrix groupoid $\mathcal{M}(S; +, \cdot)$?

ACKNOWLEDGMENTS

We are very grateful to both anonymous referees for their numerous comments on style and English usage, for their keen suggestions to expand certain arguments, and for the obvious care with which they read through this paper.

REFERENCES

1. K. Abrahamson, N. Dadoun, D. Kirkpatrick, and T. Przytycka, A simple parallel tree contraction algorithm, *J. Algorithms* **10** (1989), 287–302.
2. J. L. Balcázar, J. Díaz, and J. Gabarro, "Structural Complexity I," Springer-Verlag, New York/Berlin, 1988; "Structural Complexity II," Springer-Verlag, New York/Berlin, 1990.
3. D. A. Barrington, Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1, *J. Comput. System Sci.* **38** (1989), 150–164.
4. D. Mix Barrington, K. Compton, H. Straubing, and D. Thérien, Regular languages in NC^1, *J. Comput. System Sci.* **44**, No. 3 (1992), 478–499.
5. D. A. Mix Barrington, N. Immerman, and H. Straubing, On uniformity within NC^1, *J. Comput. System Sci.* **41** (1990), 274–306.
6. D. A. Mix Barrington and P. McKenzie, Oracle branching programs and logspace versus P, *Inform. and Comput.* **95** (1991), 96–115.
7. D. A. Barrington, H. Straubing, and D. Thérien, Non-uniform automata over groups, *Inform. and Comput.* **89**, No. 2 (1990), 109–132.
8. D. A. Barrington and D. Thérien, Finite monoids and the fine structure of NC^1, *J. Assoc. Comput. Mach.* **35** (1988), 941–952.
9. M. Beaudry, P. McKenzie, and P. Péladeau, Circuits with monoidal gates, *in* "Proceedings, 10th Symp. on Theoretical Aspects of Computer Science," Lecture Notes in Comput. Sci., Vol. 665, pp. 555–565, Springer-Verlag, New York/Berlin, 1993.
10. F. Bédard, F. Lemieux, and P. McKenzie, Extensions to Barrington's M-program model, *Theoret. Comput. Sci.* **107**, No. 1 (1993), 31–61.
11. G. Bruntrock, C. Damm, U. Hertrampf, and C. Meinel, Structure and importance of logspace-MOD-classes, *Math. Systems Theory* **25** (1992), 223–237.
12. S. R. Buss, The Boolean formula value problem is in ALOGTIME, *in* "Proceedings 19th Symp. on the theory of Computing, 1987," pp. 123–131.
13. S. R. Buss, Algorithms for Boolean formula evaluation and tree contraction, *in* "Arithmetic, Proof Theory and Computational Complexity" (P. Clote and J. Krajíček, Eds.), Oxford Univ. Press, Oxford, 1993.
14. S. R. Buss, S. Cook, A. Gupta, and V. Ramachandran, An optimal parallel algorithm for formula evaluation, *SIAM J. Comput.* **21**, No. 4 (1992), 755–780.
15. S. A. Cook, The complexity of theorem-proving procedures, *in* "Proceedings, 3rd ACM Symp. on the Theory of Computing, 1971," pp. 151–158.
16. S. A. Cook, Towards a complexity theory of synchronous parallel computation, *Enseign. Math. (2)* **27** (1981), 1–2.
17. S. A. Cook, A taxonomy of problems with fast parallel solutions, *Inform. and Comput.* **64** (1985), 2–22.
18. S. A. Cook and P. McKenzie, Problems complete for deterministic logarithmetic space, *J. Algorithms* **8** (1987), 385–394.
19. M. Garey and D. Johnson, "Computers and Intractability, A Guide to the Theory of NP-Completeness," Freeman, San Francisco, 1979.
20. L. M. Goldschlager, The monotone and planar circuit value problems are log space complete for P, *SIGACT News* **9**, No. 2 (1977), 25–29.
21. X. He, Efficient parallel algorithms for solving some tree problems, *in* "Proceedings, 24th Allerton Conference on Communication, Control and Computing, 1986," pp. 777–786.
22. J. E. Hopcroft and J. D. Ullman, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, MA, 1979.
23. N. Immerman and S. Landau, The complexity of iterated multiplication, *in* "Proceedings, 3rd Structure in Complexity Conference, 1989," pp. 104–111, IEEE Comput. Soc. Press, Washington, DC.
24. S. Kosaraju and A. Delcher, A tree-partitioning technique with applications to expression evaluation and term matching, *in* "Proceedings, 31st IEEE Symp. on the Foundations of Computer Science, 1990," pp. 163–172.
25. R. E. Ladner, The circuit value problem is log-space complete for P, *ACM SIGACT Newletter* **7** (1975), 18–20.
26. N. Lynch, Log space recognition and translation of parenthesis languages, *J. Assoc. Comput. Mach.* **24** (1977), 583–590.
27. P. McKenzie, P. Péladeau, and D. Thérien, NC^1: The automata-theoretic viewpoint, *Comput. Complexity* **1** (1991), 330–359.
28. G. Miller and J. Reif, Parallel tree contraction, Part 1: Fundamentals, *in* "Randomness and Computation," Vol. 5 (S. Micali, Ed.), JAI Press, Greenwich, 1989; Parallel tree contraction, Part 2: Further applications, *SIAM J. Comput.* **20** (1991), 1128–1147.
29. D. Muller and F. Preparata, "Parallel Restructuring and Evaluation of Expressions," Tech. Rep. UILU-ENG-88-2253 ACT-101, Coordinated Science Lab., Univ. of Illinois at Urbana-Champaign, October 1988.
30. A. Muscholl, Characterizations of LOG, LOGDCFL, and NP based on groupoid programs, extended abstract, 1992.
31. N. Pippenger, On simultaneous resource bounds, *in* "Proceedings 20th IEEE Symp. on the Foundations of Computer Science, 1979," pp. 307–311.
32. H. Straubing, Constant-depth periodic circuits, *Internat. J. Algebra Comput.* **1** (1991), 49–88.
33. A. Yao, On ACC and threshold circuits, *in* "Proceedings 31st IEEE Symp. on the Foundations of Computer Science, 1990," pp. 619–627.