Circuits, Matrices, and Nonassociative Computation*

Martin Beaudry†
Université de Sherbrooke

Pierre McKenzie**
Université de Montréal

Abstract

First, it is shown that the formula and circuit evaluation problems in the nonassociative context capture natural complexity classes up to $\text{NP}$, thus extending the known result that the word problem over a groupoid is $\text{LOGCFL}$-complete. Second, the problem of multiplying together matrices whose elements are taken from an algebraic structure more general than a semiring is defined and studied. It is shown that natural variants of this problem are complete for complexity classes such as $\text{NL}$, $\text{NC}^k$, $\text{AC}^k$, $\text{SC}^k$, and $\text{NP}$. In particular, the iterated multiplication problems involving $O(\log^k n)$ matrices over a structure $(S; +)$ in which $(S; +)$ is a monoid or an aperiodic monoid are complete for $\text{NC}^{k+1}$ and for $\text{AC}^k$ respectively, and an iterated multiplication problem variant involving matrices of size $O(\log^k n)$ is complete for $\text{SC}^k$.

1 Introduction

One usually thinks of straight-line programs, i.e. "algebraic 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 fan-in), the formula (fully or partially bracketed), 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 evaluating such circuits prescribed on input has been studied at length [1, 7, 8, 9, 11, 12, 16, 18, 21, 22, 23, 25, 26, 27].

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

- The boolean circuit value problem is complete for $P$ [23, 18].
- The boolean formula value problem is complete for $\text{NC}^1$ [11, 12].
- Word problems, i.e. "iterated multiplication problems", over appropriately represented associative structures, are complete for $L$ [16], $\text{NL}$ [21], $\text{DET}^*$ (see [15, 21]) and $\text{MOD}_L [10].$
- The word problem over the alternating group $A_5$ provides the key to the simulation of $\text{NC}^1$ by bounded-width polynomial-length branching programs [3].
- Mapping out the internal structure of $\text{NC}^1$, excluding its relationship to $\text{TC}^0$, amounts to determining the complexity of regular languages defined as word problems over appropriate monoids [8, 7, 4, 25].

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 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 evaluating such circuits prescribed on input has been studied at length [1, 7, 8, 9, 11, 12, 16, 18, 19, 21, 22, 23, 25, 26, 27].

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 $P$ [23, 18].
- The boolean formula value problem is complete for $\text{NC}^1$ [11, 12].
- Word problems, i.e. "iterated multiplication problems", over appropriately represented associative structures, are complete for $L$ [16], $\text{NL}$ [21], $\text{DET}^*$ (see [15, 21]) and $\text{MOD}_L [10].$
- The word problem over the alternating group $A_5$ provides the key to the simulation of $\text{NC}^1$ by bounded-width polynomial-length branching programs [3].
- Mapping out the internal structure of $\text{NC}^1$, excluding its relationship to $\text{TC}^0$, amounts to determining the complexity of regular languages defined as word problems over appropriate monoids [8, 7, 4, 25].

1.1 Results outline

Our first contribution is to observe that the groupoid generalization introduced by Bédard et al

*Extended abstract.
†Département de mathématiques et d'informatique, Université de Sherbrooke, Sherbrooke (Québec), J1K 2R1 Canada. Work supported by NSERC grant OGP0089786 and FCAR grants 92-NC-0608 and 91-ER-0642.
**Département d'informatique et recherche opérationnelle, Université de Montréal, C.P. 6128, succursale A, Montréal, (Québec), H3C 3J7 Canada. Work supported by NSERC grant OGPIN007 and by FCAR grant 91-ER-0642 and performed in part while on sabbatical leave at the University of British Columbia.

0-8186-2955-X/92 $03.00$ © 1992 IEEE
[9] 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 parenthesised 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", over an algebra specified on input by Cayley table, belongs to $LOGCFL$; over the fixed groupoid $G_{CFL}$ whose word problem is $LOGCFL$-complete [9], the problem is thus itself $LOGCFL$-complete (Proposition 3.4). For its part, the "nondeterministic circuit value problem" is clearly in $NP$; for circuits of logarithmic depth over groupoid $\langle \{0,1\}; NAND \rangle$, the problem is $NP$-complete (Proposition 3.5). Turning to "deterministic formulas", we observe that over any fixed algebra, the (usual) fully parenthesised formula value problem remains in $NC^1$ [12] provided that the infix representation be used (Proposition 3.8); when the formula is represented by means of its direct connection language, the problem, even over $\langle \{0,1\}; NAND \rangle$, is $L$-complete (Proposition 3.9). 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 $\langle \{0,1\}; NAND \rangle$ cannot compute functions which depend on all their inputs (Proposition 3.6).

Our second contribution concerns evaluation problems over input-dependent algebraic structures (or input-dependent subsets of an infinite algebraic structure). To our knowledge the only evaluation problems on input-dependent nonassociative structures studied so far are the word problems of [9] and [28]. In this paper, we consider evaluation problems involving matrices whose entries are drawn from a fixed algebra $(S; +, \cdot)$ which needs not be a semiring. In particular, $(S; +)$ and $(S; \cdot)$ need not be associative. When a single-valued inner vector product definition is given, such matrices form a groupoid $M(S; +, \cdot)$ 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 $O(\log^2 n)$ circuit whose inputs are elements of $M(S; +, \cdot)$ and whose gates call for multiplication in this groupoid belongs to $LOGCFL$; over the fixed groupoid $G_{CFL}$ whose word problem is $LOGCFL$-complete [9], the problem is thus itself $LOGCFL$-complete (Proposition 3.4). For its part, the "nondeterministic circuit value problem" is clearly in $NP$; for circuits of logarithmic depth over groupoid $\langle \{0,1\}; NAND \rangle$, the problem is $NP$-complete (Proposition 3.5). Turning to "deterministic formulas", we observe that over any fixed algebra, the (usual) fully parenthesised formula value problem remains in $NC^1$ [12] provided that the infix representation be used (Proposition 3.8); when the formula is represented by means of its direct connection language, the problem, even over $\langle \{0,1\}; NAND \rangle$, is $L$-complete (Proposition 3.9). 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 $\langle \{0,1\}; NAND \rangle$ cannot compute functions which depend on all their inputs (Proposition 3.6).

For our purposes an algebra is a "carrier set" $S$ equipped with a list $*_1, \ldots, *_k$ of unary or binary operators. Such an algebra is denoted $(S; *_1, \ldots, *_k)$. We assume throughout that $|S| \geq 2$. 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 $s \in S$, $e *_i s = s *_i e = s$.

2 Preliminaries and Definitions

For our purposes an algebra $A = (S; *_1, *_2, \ldots, *_k)$ is a directed acyclic graph with labelled nodes. Each indegree-zero node is labelled with an element of $S$, each indegree-one node with a unary operator of $A$, and each remaining node with a binary operator of $A$. 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-indegree-zero nodes at depth not exceeding that of $g$ which are predecessors of nodes deeper than $g$; the width of $C$ is the maximum over all nodes of the width at a node.

Definition. A formula over an algebra $A$ is a connected deterministic outdegree-one circuit over $A$. 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 $A = (S;{*}_1,*_2,\ldots,*_k)$ "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 [9]: a nondeterministic node first picks a bracketing of its input nodes 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 $(S;{*_1,*_2,\ldots,*_k})$ consists of determining whether $C$ can evaluate to $s$, where

- $C$ is a single-output circuit (resp. formula, word) over $(S;{*_1,*_2,\ldots,*_k})$, 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 $(S;+,-)$, where "-" and "*" are arbitrary binary operators. (We define the size of a matrix to be the number of entries in the matrix.) 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 $V,W \in S^*$ using the formula

$$\ldots(((v_1 \cdot w_1) + (v_2 \cdot w_2)) + (v_3 \cdot w_3)) + \ldots + (v_q \cdot w_q). \quad (1)$$

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 $M(S;+,-)$. (To take care of products which would be "undefined" because of dimension mismatch, an absorbing element $\bot$ can be created.)

Definition. Evaluation problems over $M(S;+,-)$ are those in which operands (e.g. circuit inputs) are matrices and in which the sole binary operation involved (e.g. the common label of all circuit gates) is the deterministic matrix multiplication defined implicitly by inner product (1).

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

$$(v_1 \cdot w_1) + (v_2 \cdot w_2) + (v_3 \cdot w_3) + \ldots + (v_q \cdot w_q). \quad (2)$$

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. 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 the sole binary operation involved (e.g. the common label of all circuit gates) is the nondeterministic matrix multiplication defined implicitly by inner product (2).

2.3 Complexity issues

We assume familiarity with the complexity classes

\[ AC^0 \subseteq ACC^0 \subseteq NC^1 \subseteq \mathcal{L} \subseteq \text{NL} \subseteq \text{LOGCFL} \subseteq AC^1 \subseteq NC^2 \subseteq P \subseteq NP \]

and with $NC^1$ and log space reducibilities (see [20, 15]). We adopt the definitions found in [12] for $AC^0$ and $AC^0$-reducibility (in their uniform settings). For $k \geq 1$ and $q \geq 2$, classes $AC^k, ACC^k(q)$ and $CC^k(q)$ are defined in terms of log space-uniform unbounded fanin boolean circuits of $O(\log^k n)$ depth with gate types \{AND, OR\}, \{AND, OR, MOD\} and \{MOD\} respectively, where a MOD gate outputs 1 iff the sum of its boolean inputs is a multiple of $q$. Class $ACC^k$ (resp. $CC^k$, also called "pure $ACC^k$") is the union
over all $q \geq 2$ of the classes $\text{ACC}^q(g)$ (resp. $\text{CC}^q(g)$).
Classes $\text{NC}^k$ (resp. $\text{NSC}^k$) are defined as simultaneous
deterministic (resp. nondeterministic) Turing machine
$\text{SPACE-TIME}[O(\log^k n), n^{O(1)}]$ [14, 6]. Some results
rely heavily on the recently developed simulations of
boolean circuits by programs over monoids such as
the alternating group $A_5$ [3, 5]. Recall that each instruc-
tion in such a program is specified by a function, from
the input alphabet to the monoid, together with an
integer position within the input.

Unless otherwise stated, our evaluation problems
over a fixed algebra are encoded on a fixed alphabet
using the direct connection language (see [12, Page
5]) of the circuits involved (recall that we view words
and formulas as special cases of unbounded fanin
circuits). 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
[12] concerns gate labels: we insist that the labels 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 in-
formation like the dimension of matrices and a specific
matrix entry in $\text{NC}^1$ will do the job.

Note that in log space it is possible to translate a
formula from its direct connection language representa-
tion to its infix representation. The reverse transfor-
mation can be done in $\text{NC}^1$.

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 $(S; *_1, *_2, \ldots, *_k)$ $\text{NC}^1$-reduces to the corre-
sponding evaluation problem over a groupoid $(S'; *)$, where $|S'| \leq (k+1)(|S|+2)$ 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 dis-
pose of all but one binary operator. For every binary
operator $*_i$, we add to $S$ a copy $S^{(i)}$ of $S$. Then each
node labelled with operation $*_i$, with inputs $s_1, \ldots, s_r$,
i.e. computing the word $s_1 s_2 \cdots s_r$, is replaced with
a subcircuit computing the partially parenthesized expres-
sion

$\prod_i (s_1 \cdots s_r)$

where for all $s, t, w \in S$, we define $s^{(i)} \cdots t^{(i)} = w^{(i)}$
if $s^{(i)} * t^{(i)} = w$, and where $1^{(i)}$ and $s_i^{(i)}$ are new unary
operators such that $(1^{(i)}, s^{(i)}) = s^{(i)}$ and $(1^{(i)}, t^{(i)}) = t^{(i)}$.

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 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 value $s \in S$ iff
the original circuit evaluated to $s$. To perform the reduc-
tion 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 $\text{NC}^1$. \hfill $\Box$

The word problem over any nonsolvable group $[3$
and the problem of evaluating a formula presented in
infix notation over $(\{0,1\}; \wedge, \vee, \sim)$ [11, 12] are $\text{NC}^1$-
complete. The proof by Buss, Cook, Gupta and Ram-
chandran generalizes to show:

Proposition 3.2 [11, 12] The problem of evaluating
a formula presented in infix notation, over any fixed
algebra, is in $\text{NC}^1$.

Proof sketch. Two modifications to the proof in
[12] are required. First, to accomodate noncommuta-
tive operators, the transformation to “Postfix Longer
Operand First” must replace an operator $*$ with a “re-
versed operator” $*'$ where $a * b = b *' a$, whenever the
operands of $*$ are interchanged. (This idea is used in
[12] in the context of algebraic formulas over non-
commutative rings.) Second, the pebbles used in the
two-person game now take values from $S$ (with appro-
priate modifications to the consistency requirements
determining the game outcome). Note that it seems
essential here that the size of $S$ be independent of the
input. \hfill $\Box$

We note that the infix representation is crucial to
the $\text{NC}^1$ upper bound in Proposition 3.2.

Proposition 3.3 The formula evaluation problem
over $(\{0,1\}; \wedge, \vee)$ is $L$-complete under $\text{NC}^1$ reduc-

ity. Computing the infix representation of a for-
mula over $(\{0,1\}; \wedge, \vee)$ from its “direct connection
language” representation is complete for $\text{FL} (= \text{the}
class of functions computable in log space [15])$ under
$\text{NC}^1$ reducibility.
Proof. The second statement follows from the first because evaluating the formula presented in infix can be done in NC$^1$ [12]. The NC$^1$ reduction proving the first statement is from the accessibility problem in a directed forest [16]. 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 $V$ 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 logarithmic space from its direct connection language representation. Then Lynch's algorithm solves the infix version of the problem [24].

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 LOGCFL [9]. We extend their upper bound to the case of nondeterministic outdegree-one circuits ("nondeterministic formulas", i.e. "formulas" which are only partially parenthesized).

**Proposition 3.4** The nondeterministic outdegree-one circuit problem over any fixed algebra belongs to LOGCFL.

Proof. By Proposition 3.1 we can take the fixed algebra to be a groupoid $(S;\ast)$. We reduce in log space the "nondeterministic formula" problem over $(S;\ast)$ to the word problem over a groupoid $(S';\circ)$, the latter word problem belonging to LOGCFL by [9]. The carrier $S'$ is built from $S$ by defining

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

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

left $\circ$ lowercase$(s) = $ uppercase$(s)$,
uppercase$(s) \circ$ right $ = $ 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 $\bot$, which is absorbing.

Now write $F$ for the input "nondeterministic formula" over $(S;\ast)$. Let $w(F)$ be the word obtained from the infix representation of $F$ by replacing each occurrence of "(" with "left", each occurrence of ")" with "right", and by deleting each occurrence of "\ast". Then, for any $s \in S$, $F$ evaluates to $s$ iff $w(F)$ evaluates to 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 Cayley table as part of the input. Appealing to [9], 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 depth $O(\log n)$ bounded fanin nondeterministic circuit problem over $\langle\{0,1\};NAND\rangle$ 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 [13] produces a formula, with repeated boolean variables, which on given inputs can be evaluated by an $O(\log n)$ depth circuit. Now, writing $\ast$ for NAND, observe that

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

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, 30]) can be formulated. Naturally, 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** Any family $\{f_1,f_2,\ldots\}$ of boolean functions computed by a nondeterministic constant
of circuits involving a constant number of gates and
of the depth, at the cost of using large, although still
for the underlying structure
nondeterminism, and specifying algebraic properties
several proofs are written in terms of matrices of var-
ious shapes, which can be blown up to be square
polynomial-size, matrices. We first discuss the case
from now that the algebra always contains 0.

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 corre-
spondence 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 ob-
tained 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 \( R \) to the evalua-
tion 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, and 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 iden-
tical size. In order to ease understanding, however,
several proofs are written in terms of matrices of var-
ious 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 absor-
bining for ".". 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. 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 \( M(S; +, \cdot) \) belongs
to \( NC^3 \). There is an algebra \( (S'; +, \cdot) \) such that com-
puting the product of two matrices from \( M(S; +, \cdot) \) is
\( NC^3 \)-complete under \( AC^0 \) reducibility.

Proof sketch. In a circuit with a constant number
of gates, the expression for each entry of the result-
ing matrix has polynomial length; it can be computed
from the circuit description, and then be evaluated in
\( NC^3 \) using Proposition 3.2. For the hardness part,
define \( (S; +) \) as any nonsolvable group and appeal to the
\( NC^3 \)-hardness of the word problem over this group
\([3, 12]\). Build an instance \( MN = P, \) where \( M \) is a line
vector containing the instance of the word problem, \( N \)
is a column vector of matching dimension where each
entry is some element \( \lambda \) which is an identity for "\( \cdot \)"
and \( P \) is \( 1 \times 1 \), where the entry is the target value
of the instance of the word problem.

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

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

Proof. The case \( k = 0 \) is treated above. For \( k \geq 1 \),
we concentrate without loss of generality on depth
\( D(n) \in O(\log^k n) \) straight linear formulas involving
\( z(n) \times z(n) \) matrices. Using Proposition 4.1, we turn
such a formula into a boolean circuit by replacing each
indegree-two gate with a copy of a polynomial-size,
log-depth boolean circuit which computes the prod-
uct of any two \( z(n) \times z(n) \) matrices over \( (S; +, \cdot) \), and
replacing each indegree-zero node with a set of input
nodes which give the binary encoding of the corre-
sponding input matrix. This yields the \( n \)th member
of the family of boolean circuits we want to build: a
depth \( O(\log^{k+1} n) \) boolean circuit which takes as in-
put the description of an instance of the evaluation
problem for depth \( O(\log^k n) \) straight linear formulas
over \( M(S; +, \cdot) \) and outputs the correct answer.

We now describe a method for reductions in the
other direction. We simulate circuits over a fixed alge-
bra 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 cir-
cuit \( C \) over an algebra \( (S; \ast) \). Pad the circuit with
extra gates in such a way that for any $0 \leq d \leq D$, 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 an $N \times 1$ column vector $V_{d+1}$; then we build $R$ and $M_d$ such that $V_d = M_d(RV_{d+1})$: 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 line $i$ 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 depth $0$ gates. 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, this method can be applied to prove that the depth $O(\log^k n)$ straight linear formula problem over an appropriately chosen algebra $M(S;+)$ is hard for $NC^k$. Yet we now show how to adapt the method to obtain an $NC^{k+1}$-hardness result for this same problem.

**Theorem 4.3** Let $k \geq 1$. There is an algebra $(S; +)$ such that the depth $O(\log^k n)$ straight linear formula problem over $M(S; +)$ 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 $M(S; +)$, 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 $h_{i-1}$, $1 \leq i \leq D$; (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 $N_{i+1}$, i.e. 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.

We split this big circuit into “echelons” of depth $\log n$, where echelon number $k$ consists of those copies numbered $k \log n$ to $(k+1) \log n$, and treats the gates of $C_{(k+1)\log n}$ as if they were indegree-zero input gates. These echelons are identical, so we concentrate on echelon number $0$.

Gate $g_0$, together with those gates in copies $C_i$ to $C_{\log n}$ which contribute to its value, define a log-depth, single-output boolean circuit which can be translated into a program over the group $A_5$, 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 [3]; we deal here with a multiple-output circuit given by its direct connection language, in which 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 has 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 multiple of $N$. Then each instruction can be encoded as an ordered pair of elements of $A_5$.

We treat these programs as if they were fanin $P(N)$ nondeterministic gates, and we simulate their work with matrices. For this, we define the algebra $(S; +)$, as follows.

- $S = A_5 \cup (A_5 \times A_5) \cup \{ 0, \lambda, \text{zero, one, } (?) \}$;
- element $0$ is the identity of “+” and is absorbing for “·”, while (?) is absorbing for “+”;
- operation “+” works as the group operation inside $A_5$ and inside the direct product $A_5 \times A_5$;
- $\lambda + \text{one} = \lambda + \text{zero} = \text{one} + \text{one} = \text{one}$ and $\text{zero} + \text{zero} = \text{zero}$;
- in all other cases, “+” evaluates to $\bot$;
- with $(a, b) \in A_5 \times A_5$, let $(a, b) \cdot \text{zero} = a$ and $(a, b) \cdot \text{one} = b$;
- let $\epsilon$ denote the identity of $A_5$; define $(\epsilon, \epsilon) \cdot (?) = \epsilon$, and for all other $(a, b) \in A_5 \times A_5$, $(a, b) \cdot (?) = (?)$;
- $\lambda \cdot \text{zero} = \text{zero}$; $\lambda \cdot \text{one} = \text{one}$; $\lambda \cdot (?) = (?)$.
Element (?), represents an undefined gate output, and (ε, ε) is used as a dummy instruction for padding programs. This partial definition of \((S; +, \cdot)\) is all we need for the reduction. The reader can verify that \((S; +)\) is a monoid.

Number the programs in each echelon 1 to \(N\) in a manner consistent with the numbering of the corresponding gates. Let \(Δ = D/\log n\). For a given echelon \(k, 0 < k < Δ\), 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 \(k = Δ\), the outputs of the previous echelon otherwise. Each value is one of zero, one ("output defined" to boolean values ZERO or ONE, respectively), or (?) ("output undefined"). Define also a \(P(N) \times N\) matrix \(R\) as a stack of \(P(N)/N\) square matrices containing 1 on the diagonal and 0 elsewhere, an \(N \times P(N)\) matrix \(M\), where line \(g\) contains the \(P(N)\) instructions of the program for gate \(g, 1 \leq g \leq N\), as built above, and an \(N \times N\) matrix \(F\), with entries one on the diagonal and 0 everywhere else.

The formula \(V_k = F(M(R(V_{k+1}))\) simulates the work of echelon \(k\). Indeed, the product matrix \(RV_{k+1}\) is a \(P(N) \times 1\) column vector made of \(P(N)/N\) copies of \(V_{k+1}\), and \(M(RV_{k+1})\) is a \(N \times 1\) column vector containing the outputs of the \(N\) programs, in the form of either (?) (program has read an undefined input), or \(ε\) (program outputs the bit ZERO), or an other element of \(A_5\) (program outputs ONE). In order to encode the outputs of the echelon in the same format as its inputs, we multiply to the left with matrix \(F\). Thus we obtain the output of each gate of the original circuit \(C\) by evaluating the formula

\[ F(M(R(\cdots(F(M(R(V_1)))\cdots)))) \]

We are only interested in the value of gate number 1; therefore we add to the left of the formula an \(N \times N\) matrix \(G\), such that \(G_{11} = 1\) 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

\[ G(F(M(R(\cdots(F(M(R(V_1)))\cdots))))\]

involving \(3Δ + 1\) gates and 5 matrices of size at most \(N \times P(N)\). 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 to also belong to \(NC^{k+1}\). Thus we obtain a characterization of \(NC^{k+1}\) in terms of circuits over an appropriately chosen algebra \(M(S; +, \cdot)\).

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

**Proof.** The evaluation of an \(O(\log^k n)\) depth circuit \(C\) over \(M(S; +, \cdot)\) 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 \(M(T; 0, \circ)\). 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.

In the algebra \((S; +, \cdot)\) built in the completeness proof of Theorem 4.3, operation "\(\circ\)" is associative. This immediately suggests an investigation of cases defined by restricting the algebraic properties of the monoid \((S; +)\). We obtain a characterization for 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 \(M(S; +, \cdot)\) belongs to \(AC^k\). There is an algebra \((S; +, \cdot)\) for which the depth \(O(\log^k n)\) straight linear formula problem over \(M(S; +, \cdot)\) 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 \(M(S; +, \cdot)\) belongs to \(AC^k\). For every fixed integer \(q \geq 2\), there is an algebra \((S; +, \cdot)\) for which the depth \(O(\log^k n)\) straight linear formula problem over \(M(S; +, \cdot)\) 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 \(M(S; +, \cdot)\) belongs to \(CC^k(q)\). There is an algebra \((S; +, \cdot)\) for which the depth \(O(\log^k n)\) straight linear formula problem over \(M(S; +, \cdot)\) is complete for \(CC^k(q)\) under \(NC^1\) reducibility.**

**Proof sketch.** 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 $AC^0$[8], 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 $A_5$ of 4.3), and we verify that the resulting monoid $(S;+)\text{ is also aperiodic.}$ Notice 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].

The proof of result c) is analogous, except for the simulation of $CC^k(q)$ circuits with formulas of matrices. There, 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 follow the method described in the proof of statement a), up to the description of a gate with a program over a monoid. Instead of doing this, we build an ad hoc simulation of the gate with an algebra with the prescribed properties. The details of this are deferred to the full paper.

Combining our simulation method with the $P$-completeness of the boolean circuit problem [23] leads immediately to the following.

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

### 4.2 Matrix size

In this subsection we obtain characterizations in terms of matrix formulas for classes defined by polynomial-size boolean circuits with restrictions on width, or alternately by Turing machines with simultaneous restrictions on worktime and workspace.

**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 $M(S;+,-)$ involving square matrices of size $O(Z(n)^2)$.

**Proof sketch.** 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 to the non-input gates, 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. \(\square\)

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

**Proof.** Let $z = \sqrt[3]{Z(n)}$. The product of two $z \times z$ matrices $M$ and $N$ over $(S;+,-)$ can be computed by a circuit over $(S;+,-)$ whose first level, of width $z^2$, computes the values $M_{ij} \cdot N_{jk}$ for all combinations of $i, j,$ and $k$, and is followed by $z^2 \times (z - 1)$ gates which compute in depth $z - 1$ the sums $((\cdot \cdot (M_{11} \cdot N_{12}) + (M_{22} \cdot N_{22})) + \cdot \cdot + (M_{14} \cdot N_{14})).$ Therefore, it suffices to replace in the input circuit over $M(S;+,-)$ every indegree-two node with a circuit over $(S;+,-)$ to compute the product of two matrices, to obtain a circuit with the appropriate properties. \(\square\)

The class $SC/poly$ is defined in terms of boolean circuits as $SIZE\cdot WIDTH[O(1), log^{O(1)} n]$, and in terms of non-uniform deterministic Turing machines as $TIME\cdot SPACE[O(1), log^{O(1)} n]/poly$ (see [29]). From the above proposition, we have the following.

**Proposition 4.9** For any algebra $(S;+,-)$, the linear formula problem over $M(S;+,-)$ involving matrices of size $O(1)$ $n$ belongs to $SC/poly$.

**Proof sketch.** In order to build a family of width $O(1)$ boolean circuits to evaluate the restriction to $z(n) \times z(n)$ matrices of the linear formula problem over $M(S;+,-)$, it suffices to transform every size $n$ instance of the problem into a standard shape; then by Proposition 4.7 we obtain the $n^{th}$ member of the circuit family we intend to build. We do so by replacing each indegree-two gate, which multiplies an input matrix with an intermediate result, in the form "(input)(result)" or "(result)(input)" , with a two-gate subcircuit computing "(input)((result)(input))", i.e. an expression of standard shape, where one "(input)"
is an identity for the product of matrices. We defer further details to the full paper.

We now characterize the classes \( SC^k, k \geq 1 \), by establishing a relationship between formulas of matrices and the computation of time and space bounded Turing machines (Theorem 4.11). This relationship is based on the following.

**Proposition 4.10** 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 \( M(S; +, -) \), involving matrices of size \( O(L(n)) \), where \((S; +, \cdot)\) is an appropriately chosen algebra.

**Proof sketch.** 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 doesn’t move beyond either end of its worktape, and that there is exactly one 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 [20 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 \( Z = [\sqrt{L(n)}] \). We encode a configuration of the machine as a \( Z \times Z \) matrix over an algebra \((S; +, \cdot)\), with lines and columns numbered \( 1 \) to \( Z \), in which the worktape is folded into \( Z \) segments of length \( Z \), starting at position \( 1,1 \), running back and forth through columns \( 1 \) to \( Z \), down to line \( Z \), as follows (the picture assumes that \( Z \) is even).

\[
\begin{array}{ccccccc}
\rightarrow & \cdots & \rightarrow & \\
\leftarrow & \cdots & \leftarrow & \\
\uparrow & \cdots & \uparrow & \\
\vdots & \ddots & \vdots \\
\downarrow & \cdots & \downarrow & \\
\leftarrow & \cdots & \leftarrow & \\
\end{array}
\]

The encoding is made within an algebra \((S; +, \cdot)\), which incidentally can be (carefully) worked out so that “+” is an associative operation.

Let \( Mt_{-1} \) encode the configuration of the machine after \( t-1 \) steps. Each entry of \( Mt_{-1} \) encodes a worktape cell with an element of \( S \). If the cell contains a tape character \( \sigma \), and the worktape head does not point to it, then the entry is \( \sigma \). Actually the entry contains some extra information pertaining to whether the matrix entry lies on an edge of the matrix, in which case the simulation is slightly different; we refer the reader to the full paper for further details.) If the worktape head points to the cell, then state \( q \) on a cell located in a segment running to the right in the matrix (i.e. on an odd-numbered line) and containing character \( \sigma \), is encoded as \((q \circ \sigma)\).

We simulate the \( t \)th step of the execution by multiplying \( Mt_{-1} \) to the right with a matrix \( B_t \), then to the left with matrix \( D_t \), so that \( Mt = D(Mt_{-1}B_t) \). We define \( B_t \) and \((S; +, \cdot)\) in such a way that the entries of \((Mt_{-1}B_t)\) are identical to those of \( Mt_{-1} \), except at the vicinity of the position of the read-write head. There, three things can occur. First, there can be a transition without a head motion; then if entry \( i,j \) of \( Mt_{-1} \) was \((q \circ \sigma)\) and the transition from state \( p \) reading \( \sigma \) leads to state \( q \) and to writing \( \tau \), then \((Mt_{-1}B_t)\) is identical to \( Mt_{-1} \) except at position \( i,j \) where the entry now is \((q \circ \tau)\). The subsequent product to the left with \( D_t \) then is not necessary to our purposes, hence we make sure that \( D(Mt_{-1}B_t) = Mt_{-1}B_t \).

A transition can also involve a head motion from a cell to its neighbour. Then there are two possibilities, depending on whether the two cell involved both lie on the same line in the matrix. If they do, then the encoding goes as follows. Let the transition be from state \( p \) reading \( \rho \), to state \( q \), writing \( \tau \), and doing a head motion. We use the property that the entry \( i,j \) of the product \((Mt_{-1}B_t)\) contains a contribution from every entry of line \( i \) of \( Mt_{-1} \), which means that we have all the necessary information to compute entry \( i,j \) of the product \((Mt_{-1}B_t)\) (which we expect to be \((q \circ \tau)\) ), and its appropriate neighbour (with an entry of the form \((q \circ \sigma)\)). Here too, we make sure that \( D(Mt_{-1}B_t) = Mt_{-1}B_t \).

The last case concerns a head motion which translates into a vertical motion in the matrix. This occurs only when the head starts the transition in the first or last column of \( Mt_{-1} \). Then we work in two steps. First the product with \( B_t \) leaves everything untouched, except that the entry containing the cell is changed to another element of \( S \) containing the same information, plus the indication of an upcoming vertical motion. Then, by an observation dual to the one used to simulate a horizontal motion, we use the product to the left with \( D_t \) to do the vertical motion.

Therefore the Turing machine accepts its input iff the formula

\[
D(\cdots(D(M(0 B_1))B_2 \cdots B_{T(n)}))
\]

evaluates to a matrix \( M_{\text{Accept}} \), which encodes the machine's accepting configuration.

**Theorem 4.11** Let \( k \geq 1 \). For any algebra \((S; +, \cdot)\), the linear formula problem over \( M(S; +, \cdot) \) involving matrices of size \( O(\log^k n) \) belongs to \( SC^k \). There is an
algebra \((S;+,\cdot)\) such that the linear formula problem over \(M(S;+,\cdot)\) involving matrices of size \(O(\log^k n)\) is complete for \(SC_k^1\) under \(NC^1\) reducibility.

Proof. For the first 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 \(TIME\cdot SPACE[O(n\log n),\log n]\) 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 4.10 is applied to this universal machine. Also, observe that if \(T(n) \in n^{O(1)}\) and \(L(n) \in \log n\), 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. \(\square\)

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 4.12 The multiple-valued product of two matrices can be done 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 do just one matrix product: we compute each matrix entry using Proposition 3.4. To prove the \(LOGCFL\)-hardness claim, define \((S;+\cdot)\) as the groupoid \(GCFL\), over which the word problem is \(LOGCFL\)-complete, and reduce as in the proof of Proposition 4.1. The \(NL\)-completeness statement concerns groupoid \(GNL\), over which the word problem is \(NL\)-complete. (See [9] for definitions of these groupoids and completeness proofs, given there with \(DLOGTIME\) reductions.) \(\square\)

Proposition 4.13 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 problem \(CNF\cdot SAT\) [17]. Consider a \(CNF\cdot SAT\) instance with \(m\) clauses and \(v\) variables. We define an \(m \times v\) matrix \(A\), a \(v \times 3\) matrix \(B\), and a \(3 \times 1\) matrix \(C\) over a structure \((\{0,\text{zero},\text{one},\text{true},\text{false},\perp\};+\cdot)\) having the property that \(A(BC)\) evaluates to the all-true vector iff the \(CNF\cdot SAT\) instance is satisfiable. As usual, element 0 acts as an identity for \(+\) and is absorbing for \(\cdot\). We define operation \(+\cdot\) on \(\{\text{zero, one}\}\) as a \(NAND\) and on \(\{\text{true, false}\}\) as \(OR\), otherwise \(+\cdot\) evaluates to \(\perp\). We define operation \(\cdot\) on the nonzero values with the following table.

<table>
<thead>
<tr>
<th></th>
<th>zero</th>
<th>one</th>
<th>false</th>
<th>true</th>
<th>(\perp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>false</td>
<td>false</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
</tr>
<tr>
<td>one</td>
<td>zero</td>
<td>one</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
</tr>
<tr>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>(\perp)</td>
</tr>
</tbody>
</table>

Set \(C = \begin{pmatrix} \text{zero} \\ \text{one} \end{pmatrix}\) and fill \(B\) everywhere with the value one. Given that \((\text{zero} + \text{one}) + \text{one} = \text{zero}\) and \(\text{zero} + (\text{one} + \text{one}) = \text{one}\), the product \(BC\) nondeterministically produces a \(v \times 1\) column vector filled with \(\text{zeros}\) and \(\text{ones}\). This vector provides the guesses for the values of the boolean variables satisfying the \(CNF\cdot SAT\) formula. Now define each entry \(a_{ij}\) in \(A\), \(1 \leq i \leq m\), \(1 \leq j \leq v\), as follows:

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

Now, assuming without loss of generality that no variable appears twice in any given clause, the \(i^{th}\) entry in the \(m\)-vector \(A(BC)\) evaluates to \(true\) whenever the boolean values given by \(BC\) satisfy clause \(i\). \(\square\)

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 4.13 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 4.10. The proof consists in modifying technical details in the proof of 4.10, and is deferred to the full paper.

**Proposition 4.14** 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 \(\mathcal{M}(S;+,\cdot)\) is an appropriately chosen algebra. \(\Box\)

**Theorem 4.15** Let \(k \geq 1\). For any algebra \(\mathcal{M}(S;+,\cdot)\), the linear formula problem, involving matrices over \(\mathcal{M}(S;+,\cdot)\) of size \(O(\log^k n)\) and multiple-valued products, belongs to \(\text{NSC}^k\). There is an algebra \(\mathcal{M}(S;+,\cdot)\) over which this problem is complete for \(\text{NSC}^k\) under \(\text{NC}^1\) reducibility.

**Proof.** Every entry of a multiple-valued matrix product can be evaluated in turn by first guessing a parenthesization for its expression, using \(O(\log n)\) characters, and then evaluating deterministically. The rest of the proof follows Theorem 4.11. \(\Box\)

We now state a result on circuits with nondeterministic gates, a counterpart to Proposition 4.13. In this statement, we call a nondeterministic formula a circuit in which the gates have outdegree one, while the in-degree does not have to be at most two.

**Proposition 4.16** The problem of evaluating a nondeterministic matrix circuit belongs to \(\text{NP}\). There is an algebra \(\mathcal{M}(S;+,\cdot)\) over which the nondeterministic linear formula problem is \(\text{NP}\)-complete under \(\log\) space reducibility.

**Proof sketch.** We reduce from an instance of problem CNF-SAT involving \(m\) clauses and \(v\) variables to the evaluation of the partially parenthesized formula \(A(F((ID_1 I) \cdots (ID_v I)))\), where each subformula \((ID_i I), 1 \leq i \leq v\), is used to guess the value of the \(i\)th variable, matrix \(F\) forms their product into a \(v \times 1\) column vector, and matrix \(A\) verifies whether the clauses are satisfied, and works in the same way as the matrix \(A\) of Proposition 4.13. We define \(I\) and \(D_i\), \(1 \leq i \leq v\), as \(v \times v\) matrices, where \(I\) contains a value \(\Delta\) on the diagonal and 0 everywhere else, and \(D_i\) is identical, except at position \(i, i\) where the entry is \(\gamma\). With \(\Delta \cdot \gamma = \text{true}\) and \(\gamma \cdot \Delta = \text{false}\), we obtain different settings for the \(i\)th boolean variable, depending on whether the block \(ID_i I\) was evaluated as \((ID_i I)\), or \((ID_i I)\). Further details are deferred to the full paper. \(\Box\)

Applying the argument of Proposition 3.4 to algebras of matrices, we obtain from this an \(\text{NP}\)-completeness statement for the word problem over such algebras. An \(\text{NP}\)-completeness result for the word problem on exponential-size groupoids (with a different representation) has been obtained independently by Muscholl [28].

5 Conclusion

We have generalized the circuit evaluation problem to include the evaluation of unbounded fan-in 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 \(\mathcal{M}(S;+,\cdot)\) or from the nonassociative groupoid \(\mathcal{M}(S;+,\cdot)\) of matrices with entries from \(\mathcal{M}(S;+,\cdot)\) (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 \(\text{NP}\), in particular the various "parallel complexity" classes within \(\text{NC}\) (albeit with the need for an occurrence of the bound \(O(\log^k n)\) as part of the problem definition in some cases).

Although our results are descriptive in the sense that they characterize known complexity classes without suggesting separation arguments, 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 \(\mathcal{M}(S;+,\cdot)\) and its resulting matrix groupoid \(\mathcal{M}(S;+,\cdot)\)?

References


