Non-Commutative Arithmetic Circuits: Depth Reduction and Size Lower Bounds*

Eric Allender†

Department of Computer Science
Rutgers University
Hill Center, Busch Campus, P.O. Box 1179
Piscataway, NJ 08855-1179, USA
`allender@cs.rutgers.edu`

Jia Jiao‡

Department of Computer Science
Rutgers University
Hill Center, Busch Campus, P.O. Box 1179
Piscataway, NJ 08855-1179, USA
`jjiao@paul.rutgers.edu`

Meena Mahajan

The Institute of Mathematical Sciences,
C.I.T. Campus,
Madras 600 113, India
`meena@imsc.ernet.in`

V Vinay§

Department of Computer Science and Automation,
Indian Institute of Science,
Bangalore 560 012, India
`vinay@csa.iisc.ernet.in`

7 August 1995

*The results in this paper were originally announced in papers in Proc. 25th ACM Symposium on Theory of Computing and in Proc. 14th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science 880.

†Supported in part by National Science Foundation grants CCR-9204874 and CCR-9509603. Part of this work was done while on sabbatical leave at Princeton University.

‡Supported in part by National Science Foundation grant CCR-9204874.

§Work done while with the Institute of Mathematical Sciences, Madras 600 113, India.

Abstract

We investigate the phenomenon of depth-reduction in commutative and non-commutative arithmetic circuits. We prove that in the commutative setting, uniform semi-unbounded arithmetic circuits of logarithmic depth are as powerful as uniform arithmetic circuits of polynomial degree; earlier proofs did not work in the uniform setting. This also provides a unified proof of the circuit characterizations of LOGCFL and #LOGCFL.

We show that AC^1 has no more power than arithmetic circuits of polynomial size and degree $n^{O(\log\log n)}$ (improving the trivial bound of $n^{O(\log n)}$). Connections are drawn between TC^1 and arithmetic circuits of polynomial size and degree.

Then we consider non-commutative computation, and show that some depth reduction is possible over the algebra $(\Sigma^*, \max, \text{concat})$, thus establishing that OptLOGCFL is in AC^1. This is the first depth-reduction result for arithmetic circuits over a noncommutative semiring, and it complements the lower bounds of [Ni91, Ko90] showing that depth reduction cannot be done in the general noncommutative setting.

We define new notions called "short-left-paths" and "short-right-paths" and we show that these notions provide a characterization of the classes of arithmetic circuits for which optimal depth-reduction is possible. This class also can be characterized using the AuxPDA model.

Finally, we characterize the languages generated by efficient circuits over the (union, concat) semiring in terms of simple one-way machines, and we investigate and extend earlier lower bounds on non-commutative circuits.

1 Introduction

One of the most striking early results of arithmetic circuit complexity is the theorem of [VSBR83], showing that any arithmetic circuit of polynomial size and polynomial algebraic degree, with $+$ and $\times$ gates defined over a commutative semiring, is equivalent to an arithmetic circuit of polynomial size having depth $\log^2 n$. (In fact, if the $+$ gates are allowed to have unbounded fanin, the depth is logarithmic, as was observed in [Vi91].) Unfortunately, the construction in [VSBR83] is not uniform. The results of [VSBR83] were extended in [MRK88] by providing fast parallel algorithms for evaluating arithmetic circuits. However, these algorithms involve a component that is hard for NLOG, so no logspace uniform construction was known.

One of the first uniform depth reductions was shown in [Ve91] (improving [Ru81]) for the Boolean ring. Vinay [Vi91] showed a similar result for circuits over integers, by using LOGCFL machines (that is, AuxPDA's running in polynomial time and logarithmic space [Su78]) to achieve depth reduction. The proof crucially uses the fact that the given circuit may be simulated by a LOGCFL machine with small pushdown height.

In section 3, we build on these techniques, to give a direct uniform depth reduction result over any commutative semi-ring, while staying within the circuit model. We show that any polynomial size polynomial degree circuit has an equivalent polynomial size semi-unbounded logarithmic depth circuit. In Section 4, we include some related results concerning arithmetic circuits over commutative semirings, and connections to Boolean complexity classes. We show what can be viewed as essentially a degree-reduction result: every function in AC^1 can be reduced to a function computed by a polynomial size arithmetic circuit of degree $n^{O(\log\log n)}$ over the natural numbers, improving the trivial degree upper bound of $n^{O(\log n)}$.

[MRK88] also raised the question of whether analogous results could be proved in the presence of non-commutative multiplication. Motivated by this question, [Ko90] and [Ni91] showed that commutativity is crucial for the results of [VSBR83, MRK88]. Namely, it was shown in [Ko90, Theorem 1] that for the particular semiring of 2^{Σ^*} with $\times$ denoting concatenation and $+$ denoting union, there is a circuit with linear size and degree that is not equivalent to any circuit with sublinear depth. An essentially equivalent example is presented in [Ni91, Theorem 4]. Although fast parallel algorithms were shown for certain limited sorts of non-commutative algebras (e.g., finite semirings) in [MT87], no examples were known of non-commutative semirings where depth reduction can be accomplished within the arithmetic circuit model.

We show, in section 5, that for polynomial degree circuits, a limited sort of depth reduction can be carried out in the particular case of the algebra on Σ^* where $\times$ is concatenation and $+$ is lexicographic maximum. We do not have an interesting characterization of the class of algebras to which our techniques apply. However, this particular algebra is of interest for two reasons.

- Concatenation is in some sense the canonical example of a non-commutative multiplication operation.
- Natural classes of optimization problems (in particular the classes OptL [AJ93, AJ92] and OptLOGCFL [Vi91]) can be characterized in terms of arithmetic circuits over (max,concat).

It is important to note that we show only that over this algebra, arithmetic circuits of polynomial size and degree can be simulated by *unbounded* fanin circuits of logarithmic depth. An optimal result would do this for semi-unbounded fanin circuits.

We also give, in section 6.1, an augmented set of sufficient conditions (since polynomial degree alone is not known to suffice) for depth reduction in general non-commutative settings. We identify two structures called *short-left-paths* and *short-right-paths*. We prove that if a circuit is intertwined with these structures in a reasonable way, depth reduction is possible in a non-commutative setting. Interestingly, we use mirror reflections of circuits as a technique to handle non-commutative circuits, making some of the proofs very simple. These constructions syntactically characterize circuits with equivalent semi-unbounded logarithmic depth circuits, whereas the construction of section 5.2 yields only unbounded fanin circuits.

We introduce generalized LOGCFL machines (section 6.2), which can perform computations over any specified algebra. (For instance, over (max,concat), such machines define precisely OptLOGCFL.) We show that restricting the pushdown height in such machines precisely captures the short-left-paths property; thus pushdown-height-bounded generalized LOGCFL machines have equivalent semi-unbounded logarithmic depth arithmetic circuits. Note that in both the commutative cases of the Boolean ring and of integers, restricting the pushdown height does not result in any weakening of the class.

In section 7, we generalize some of the lower-bound results on the size of skew (union,concat) circuits. Skew circuits have been used to characterize NLOG [Ve92] as well as the complexity of the determinant [To92]. In [Ni91], Nisan shows lower bounds on the size of left-skew circuits generating certain languages.

(The lower bounds are proved for size of Algebraic Branching Programs; it is easy to see that these programs correspond exactly to left-skew circuits.) He further shows that if a set is hard to generate in left-skew circuits, then it must have high formula complexity and consequently high circuit depth complexity. Thus the resulting lower bound on depth is absolute. However, there is no corresponding lower bound on circuit size, or even on skew-circuit size.

In section 7.2, we extend this argument to show lower bounds on circuit size when the circuits are allowed to be more general than left-skew. In particular, we define a regular skewness pattern called clone skewness and show clone-skew-circuit size lower bounds for some problems. We also establish that the generalization is indeed proper; there are problems provably hard for left-skew-circuits (i.e. requiring exponentially sized left-skew circuits) which have small clone-skew circuits.

By way of proving these results, we establish, in section 7.1, formal connections between one-way language acceptors and (union,concat) circuits. This gives us some intuition in choosing candidate languages to exhibit the limitations and the power of clone-skew circuits.

2 Preliminaries

A *semiring* is an algebra over a set $\mathcal{S}$ with two operations $+, \times$ satisfying the usual ring axioms, but not necessarily having additive inverses. (For more formal definitions see [JS82].) There is an additive identity denoted $\perp$ and a multiplicative identity denoted λ (At times it will be more convenient to denote $\perp$ by 0, and to denote λ by 1. When we have an alphabet $\Sigma = \{\mathbf{0},\mathbf{1}\}$, we will try to avoid confusion by using boldface symbols to denote elements of Σ.) We will usually be interested mainly in finitely-generated semirings (meaning that there is a finite set $\mathcal{G} = \{g_1, \ldots, g_m\} \subseteq \mathcal{S}$ generating all of $\mathcal{S}$), although many of our results hold for circuits over semirings that are not finitely-generated (such as the real numbers).

An *arithmetic circuit* over this semiring consists of gates labeled with the operations $+$ and $\times$. The fanin of each gate may be bounded or unbounded, giving rise to three kinds of circuit classes: (1) bounded fanin circuits (unless otherwise stated, circuits are assumed to have bounded fanin gates), (2) unbounded fanin circuits, and (3) semi-unbounded circuits, where the $+$ gates have unbounded fanin but the $\times$ gates have bounded fanin. The inputs to the $\times$ gates are assumed to be ordered. Thus each $\times$ gate with fanin two has a left input and a right input. A circuit has one output node. A *circuit family* is a set of circuits $\{C_n : n = 1, 2, \ldots\}$, where C_n has n input variables.

Several different notions of algebraic circuit have been considered in the literature; often the most important difference between models concerns the type of inputs to the circuits that are considered. One popular model (for instance, the model studied in [VSBR83]) allows a circuit with n input variables to compute a function defined on $\mathcal{S}^n$; that is, an input variable can be assigned the value of *any* element from the semiring; and thus a single circuit may compute a function on an infinite domain. Although this is an interesting model, and although many of the proofs in this paper carry over to this model (particularly the proofs of Theorem 3.1 and Lemmas 6.1 and 6.4), our motivation in this paper comes primarily from machine-based complexity classes such as #L, #LOGCFL, OptL, and OptLOGCFL. These classes have appealing characterizations in terms of arithmetic circuits, where a circuit C_n with n input variables now has the restriction that variables can take on only values of "length" n, where there is some meaningful notion of the "length" of a semiring element. For the finitely-generated semirings that we find most interesting, elements of "length" n can be efficiently constructed from the generators, and thus it is no loss of generality to allow the inputs to a circuit to take on only values from the list of generators. (This also highlights the similarity between arithmetic circuits and Boolean circuits, where Boolean circuits take inputs only from the set $\{0, 1\}$.)

The discussion in the preceding paragraph motivates the following aspect of the arithmetic circuits considered in this paper. The leaf nodes of an arithmetic circuit are labeled either with some element of $\{\lambda, \perp\} \cup \mathcal{G}$, or with a predicate of the form $[x_i, a, b, c]$, where x_i is an input variable x_i, and $\{a, b, c\} \subseteq \{\lambda, \perp\} \cup \mathcal{G}$. ($\mathcal{G}$ is the set of generators.) If labeled by such a predicate, the leaf evaluates to b if $x_i = a$ and to c otherwise.

The convention that leaves are labeled by predicates of the form $[x_i, a, b, c]$ has not been used previously, and may require further justification. In particular, it might not be completely obvious to the reader that this convention allows the circuit to even pass on the value of the input x_i. To see that this is possible, note that this can be computed by a sub-circuit of the form

$$\sum_{a \in \mathcal{G}} \prod([x_i, a, \lambda, \perp] \times a)$$

We use this convention because it is quite essential in the special case of (Σ^*, max, concat), which is of particular interest to us in characterizing OptLogCFL. We cannot hope to characterize OptLogCFL without leaf predicates of this sort; in many semirings, denying this sort of function at the leaves of a circuit essentially forces the circuit to be monotone. Circuits where the leaves evaluate only to the value of the input variable x_i are of course a special case, as indicated above.

The *size* of an arithmetic circuit is the number of gates in it, and the *depth* is the length of the longest path from an input to the output. We will also need the notion of the (algebraic) *degree* of a node (which should not be confused with the fanin of a node). The degree is defined inductively: a leaf node has degree 1, a $+$ node has degree equal to the maximum of the degrees of its inputs, and a $\times$ node has degree equal to the sum of the degrees of its inputs. The degree of a circuit is the degree of its output gate. A circuit family is *uniform* if the function $1^n \mapsto C_n$ is logspace-computable. (Note that uniform circuit families have polynomial size.)

NC^k, SAC^k and AC^k refer to the classes of functions computed by uniform families of $O(\log^k n)$ depth circuits with bounded, semi-unbounded and unbounded fanin respectively, over the Boolean ring. #SAC^k and #AC^k refer to analogous classes over natural numbers.

A circuit is *skew* if each $\times$ gate has at most one non-leaf input. It is left-skew if the $\times$ gates have bonded fanin and the left input of each $\times$ gate is a leaf.

By a proof tree in a circuit we mean a sub-circuit represented as a tree (duplicating gates if required) such that

- The output gate of the circuit belongs to the subcircuit.
- Exactly one input of each $+$ gate in the sub-circuit is present in the sub-circuit.

- Both children of each $\times$ gate in the sub-circuit are present in the sub-circuit.

LOGCFL is the class of problems logspace reducible to context-free languages. It is characterized by the machine class of nondeterministic logspace-bounded AuxPDAs running in polynomial time [Su78], and also by the circuit class SAC^1 [Ve91]. Without loss of generality we assume that LOGCFL machines are in a normal form where they push or pop $O(\log n)$ symbols (one meta-symbol) at a time. (Such a conversion can be achieved by letting the LOGCFL machine store up to $\log n$ top-of-stack symbols on its tape.) In this normal form, we define "height" of the stack in terms of meta-symbols; the machine runs in stack height $h(n)$ if the number of symbols on the stack is at most $h(n)\log n$. This convention is not completely standard, but it simplifies the exposition and clarifies certain relationships. Essentially, we feel that this is the "right" way to view stack height in this model.

A surface configuration is a description of the AuxPDA's state, input tape head position, worktape contents, worktape head position, and top-of-stack meta-symbol. A pair of surface configurations (P, Q) forms a *realizable pair* if there is a computation of the AuxPDA which when started on P leads to Q, and the pushdown height at P and Q are identical, and the pushdown height at any intermediate step never goes below the pushdown height at P. For the original definitions of "surface configuration" and "realizable pair," see [Co71]. Further details (standard notation and definitions) about LOGCFL may be found in [Co71, Ve91, Vi91].

3 Depth Reduction in Commutative Rings

It is known [VSBR83, MRK88] that polynomial degree polynomial size circuits over any commutative semi-ring have equivalent logarithmic depth unbounded fanin polynomial size circuits. However, the argument provided in [VSBR83] requires that the degree of each gate be known *a priori*. Although this can be computed quickly in parallel [MRK88], it is easy to see that this is hard for NLOG, and thus cannot be assumed in the logspace-uniform circuit model. For the particular case of the Boolean ring, a uniform depth-reduction result is proved in [Ve91], and for the integers it is proved in [Vi91]. We present here a uniform depth reduction algorithm for the case of general commutative semirings.

Theorem 3.1 *Let R be any commutative semiring. The class of functions computed by uniform arithmetic circuits over R of polynomial degree is equal to the class of functions computed by uniform semi-unbounded arithmetic circuits over R of depth $O(\log n)$.*

Proof: The first step is to convert the given circuit to the normal form guaranteed by the following lemma:

Lemma 3.2 *For any uniform polynomial-degree circuit family there is an equivalent one with the property that each gate is labeled with its formal degree.*

Proof: Let C be an arithmetic circuit. Assume that there are no consecutive $+$ gates in C. (If necessary, insert "$\times 1$" gates between each two consecutive $+$ gates; this does not cause the degree to become non-polynomial.) Now let C' be the circuit constructed as follows: for each gate g in C, build gates $(g,1),(g,2),\ldots,(g,n^k)$ (where n^k is an upper bound on the degree of C). If g is a $+$ gate, then (g,i) is a $+$ gate with children $\{(h,i) : h \text{ is a child of } g\}$. If g is a $\times$ gate, then (g,i) is a $+$ gate with children that are $\times$ gates of the form $(h_1,j)\times(h_2,i-j)$, where h_1 and h_2 are the children of g, and $1 \leq j \leq i-1$. If h is *not* a leaf, then make $(h,1)$ a leaf with value 0. If h *is* a leaf and $i > 1$, then make (h,i) the root of a trivial subcircuit with degree i and value 0, and make $(h,1)$ a leaf connected to the same input variable as h is connected to. The output gate of C' is the sum of all gates (g,i), where g is the output gate of C.

It is easy to prove by induction on i that each gate (h,i) in C' has as its value the sum of all monomials of degree i in the formal polynomial corresponding to gate h in circuit C. Thus C' is equivalent to C. ▌

The following technical definition will be useful later in the argument. Say that g and h are $+$-*adjacent* if there is a path from g to h where all intermediate nodes are $+$ gates. Note that the circuit C' constructed in Lemma 3.2 has the property that if g and h are $+$-adjacent, then the path of $+$ edges connecting

g and h must be unique and have length at most 3, and there is not any path from g to h through a $\times$ gate. Among other things, this guarantees that it is easy to check in $O(\log n)$ space if g and h are $+$-adjacent.

Also, note that it is easy to re-write C' so that the first child of any $\times$ gate has degree no more than the degree of the second child (since R is commutative).

An *exploration* of gate g is a depth-first search of a subcircuit rooted at g, with the property that

- For each $+$ node, a child is chosen nondeterministically to explore.
- For each $\times$ node the second child is put on the stack and the first child is explored.
- When a leaf is encountered, the stack is popped and the node on top of the stack is explored (unless the stack is empty, in which case the exploration stops). The leaf that is the last node visited is the *terminal node* of the exploration.

Note that, by our guarantee that the degree of the first child of a $\times$ node is no more than half the degree of its parent, it follows that the degree of the node being explored decreases by $1/2$ each time the stack height increases. Thus the stack height is logarithmic on any exploration. Let the *exploration height* of a node be the maximum stack height of any exploration of the node.

Clearly the value of g is the sum (over all explorations e of g) of the product of all leaves encountered on e. (This can be verified by an easy induction starting at the leaves.)

Given gate g and leaf l, define $[g,l]$ to be the sum over all explorations e of g having terminal leaf l of the product of all leaves encountered on e. (To clarify: in the case that *no* exploration of g has terminal leaf l, $[g,l]=0$. Thus the value of g is the sum over all leaves l of $[g,l]$.

More generally, for any two gates g and h where h is *not* a leaf, let $[g,h]$ denote the value determined by the definition in the preceding paragraph, where gate h is replaced by a leaf with value 1. We will show how to build a circuit computing the values $[g,h]$ for *all* h (leaf and non-leaf).

If g is a leaf, then $[g,g]$ is a leaf returning the value of g, and for all other h, $[g,h]$ is a leaf with value 0.

If g is a $+$ gate, then $[g,h]$ is simply the sum of all $[g',h]$, where g' is a child of g.

If g is a $\times$ gate and g and h are $+$-adjacent and h *is* a leaf, then $[g,h]$ should simply return $h \times$ (the value of g_1), where g_1 is the first child of g. (This is because there is a one-to-one correspondence between explorations of g and explorations of g_1, because of the uniqueness of the $+$ path connecting g and h). Thus if g is a $\times$ gate and g and h are $+$-adjacent and h is a leaf, then $[g,h]$ is $h \times$ (the sum over all leaves l of $[g_1,l]$).

Similarly, if g is a $\times$ gate and g and h are $+$-adjacent and h is *not* a leaf, then $[g,h]$ is the sum over all leaves l of $[g_1,l]$. (This is because h is treated as a leaf with value 1.)

If g is a $\times$ gate and g and h are *not* $+$-adjacent, then the definitions imply that $[g,h]$ is equal to 0 unless there is an exploration of g with h as a terminal node (where h is treated as a leaf). For each such exploration there is a *unique* sequence of gates $g = g_0, g_1, \ldots, g_m = h$ such that the *second* child of g_i is $+$-adjacent to g_{i+1}, and each g_i is a $\times$ gate (except possibly $g_m = h$). (That is, being the terminal node of an exploration is equivalent to being reachable via a path using only $+$ gates and the *second* edges out of $\times$ gates.) In such a sequence there is *exactly* one g_i such that[1] degree(g_i) $\geq$ (degree(g) $+$ degree(h))/2 $>$ degree of the second child of g_i. The product of the leaves encountered along this exploration is the product of the leaves encountered before g_i and those encountered after g_i. It follows that $[g,h]$ is the sum over $\{g_i$ such that degree(g_i) $\geq$ (degree(g) $+$ degree(h))/2 $>$ degree of the second child of $g_i\}$ of $([g,g_i] \times [g_i,h])$.

Clearly the resulting circuit is of polynomial size, and is semi-unbounded. To analyze the depth of the circuit, observe that the subcircuit evaluating $[g,h]$ when g and h are $+$-adjacent depends on subcircuits of the form $[g',h']$, where the exploration height of g' is one less than the exploration height of g. Also note that if g and h are not $+$-adjacent, then the subcircuit evaluating $[g,h]$ depends on subcircuits of the form $[g',h']$ where degree(g') $-$ degree(h') is no

[1]In this expression, "degree(g)" refers to the degree of g in circuit C', not its degree in the subcircuit being explored (where h is a leaf). Similarly, "degree(h)" is the degree of h in C'. Note that by the construction of C', we can assume that these degrees are explicitly encoded in the names of g and h.

more than half of degree(g) − degree(h). It follows that the depth is $O(\log n)$.
▮

4 Relating Arithmetic and Boolean Complexity Classes

Computing the determinant of integer matrices is known to be hard for NLOG, and it can be done in TC^1 (TC^1 denotes the class of functions computable by threshold circuits (equivalently, MAJORITY circuits) of polynomial size and depth $O(\log n)$). However, no relationship is known between SAC^1 or AC^1 and the determinant. In this section we review some known results about arithmetic circuits that bear on these questions, and present some new inclusions and characterizations.

Definition 4.1 *#L is the class of functions of the form $\#acc_M(x)$, where M is an NLOG machine. ($\#acc_M(x)$ counts the number of accepting computations of M on input x.)*

#LOGCFL is the class of functions of the form $\#acc_M(x)$, where M is a polynomial-time bounded nondeterministic AuxPDA.

Vinay has shown (see [Vi91]) that #LOGCFL is precisely the class $\#\mathrm{SAC}^1$, and is also precisely the class of functions computed by uniform poly-degree arithmetic circuits over the natural numbers.

It is known that the complexity of the determinant is roughly determined by #L. More specifically, f is logspace many-one reducible to the determinant[2] iff it is the difference of two #L functions (see [Vi91a, Da91, To91a]; an essentially equivalent result is also proved in [Va92, Theorem 2]). Also, this class of functions is precisely the class computed by polynomial-size skew arithmetic circuits over the integers [To92]. For additional related results, see [AO94].

The question of the relationship between #L and #LOGCFL is thus exactly the question asked in [Va79], concerning the relationship between the determinant and circuits of polynomial size and degree.

It is worth mentioning that Immerman and Landau [IL95] have conjectured that TC^1 is exactly the class of sets reducible to $\#\mathrm{SAC}^1$; in fact they make the

[2]That is, there is a logspace-computable g such that $f(x) =$ determinant($g(x)$).

stronger conjecture that computing the determinant is hard for TC^1. Here, we point out that there is a tantalizing connection between TC^1 and $\#\mathrm{SAC}^1$.

Theorem 4.2 *A function is computed by TC^1 circuits iff it is computed by arithmetic circuits over the natural numbers, with depth $O(\log n)$, polynomial size, with unbounded fanin $+$ gates, and fanin two $\times$ and $\div$ gates.*

Here, $\div$ is integer division, with the remainder discarded.

Proof: Since unbounded fanin $+$, and $\times$ and $\div$ can be computed by TC^0 circuits [RT92], inclusion from right-to-left is straightforward. (This is true even for logspace-uniformity, since it follows from [BCH86] and [RT92] that division can be done in uniform TC^0 if the number N is given, where N is the product of the first n^2 primes. But N can be computed in TC^1.)

To see the other direction, note that the MAJORITY of $x_1, \ldots, x_n$ is equal to $(\sum_{i=1}^{n} x_i) \div 2^{\lfloor \log n \rfloor}$. The subcircuits computing powers of 2 can be re-used; thus $O(\log n)$ layers of MAJORITY gates can be simulated with $O(\log n)$ levels of arithmetic gates. ∎

We note that other (less trivial) connections between TC^0 and classes of arithmetic circuits over finite fields are also known [RT92, BFS92].

In spite of Theorem 4.2, it is not known if TC^1 or even AC^1 can be reduced to arithmetic circuits of polynomial size and degree ($\#\mathrm{SAC}^1$).[3] It is a trivial observation that AC^1 can be reduced to arithmetic circuits over the integers of polynomial size and degree $n^{O(\log n)}$.[4] The following result improves this trivial bound to $n^{O(\log\log n)}$. (Note that arithmetic circuits of nonpolynomial degree

[3]In this context, when we say that a complexity class can be "reduced to" a class of functions, we mean that for every language A in the complexity class, there is a function f in the class of functions, and a logspace-bounded oracle Turing machine that, on input x, can determine if $x \in A$ by computing a query y, and receiving from the oracle the result $f(y)$, and then using this information to decide whether to accept or reject. In fact, our results deal equally well with the setting where only one bit of $f(y)$ is requested from the oracle. Note in particular that SAC^1 can be reduced to $\#\mathrm{SAC}^1$ in this way, by the same observation showing that NLOG is contained in probabilistic logspace (and thus the high-order bit of a #LOGCFL function can determine if a LOGCFL machine accepts).

[4]The circuit can first be made "unambiguous" by replacing each OR of gates $g_1, \ldots, g_m$ by an OR of m gates testing, for each i, if the condition "gate $g_i = 1$ but for all $j < i, g_j = 0$". Now replace each AND by $\times$ and each OR by $+$; the low-order bit of the answer is the answer we seek. The degree bound is easily seen to hold.

can produce output of more than polynomial length. The following proof does not make use of this capability; only the information in the low-order $O(\log n)$ bits is used.)

Theorem 4.3 [5] *Every language in AC^1 is efficiently reducible to a function computed by polynomial-size, degree $n^{O(\log\log n)}$ arithmetic circuits over the natural numbers.*

Proof: First we need the following lemma, which follows directly from the results of [CRS93] and [IZ89]. (This improves earlier constructions in, for example, [VV86, To91, AH93, KVVY93], which also showed how to simulate AND and OR by parity gates and ANDs of small fan-in. It would also be possible to use constructions by [NRS94] or [Gu95], instead of that of [CRS93].)

Lemma 4.4 *For each $l \in \mathbf{N}$, there is a family of constant-depth, polynomial size, probabilistic circuits consisting of unbounded-fan-in PARITY gates, AND gates of fan-in $O(\log n)$, and $O(\log n)$ probabilistic bits, computing the OR of n bits, with error probability $< 1/n^l$.*

Proof: (To see why the claim is true, first observe that the construction in [CRS93] gives a depth 5 probabilistic circuit that computes the NOR correctly with probability at least $\frac{1}{2}$ and uses $O(\log n)$ random bits. More precisely, using the terminology of [CRS93], let $m = \lceil \log n \rceil$, let $S = \{1, \ldots, m\}$, and let $\mathcal{F}$ be the collection of subsets of S, such that $A \in \mathcal{F}$ iff the bit string k of length $\log n$ representing the characteristic sequence of A corresponds to a binary number $k \leq n$ such that the k-th bit of the input sequence $x_1, \ldots, x_n$ has value 1. That is, the OR of $x_1, \ldots, x_n$ evaluates to 1 iff $\mathcal{F}$ is not empty. The strategy of [CRS93] is to use probabilistic bits to define a way of assigning a "weight" to each set $A \in \mathcal{F}$ so that if $\mathcal{F}$ is not empty, then with high probability there is a unique element of $\mathcal{F}$ having minimum weight. The next paragraph explains how this is done.

[5]This improves a theorem of [AJ93a], where a similar result for nonuniform circuits was proved.

Let $c = \lceil \log m \rceil$ and let $t = \lceil m/c \rceil$. For any $i \leq m$ and $j \leq t-1$, define $b_{i,j}$ as follows:

$$b_{i,j} = \begin{cases} 2^{i-jc} & \text{if } 2^{jc} \leq 2^i < 2^{(j+1)c} \\ 0 & \text{otherwise} \end{cases}$$

(It may help the reader's intuition to think of the m-bit number 2^i as being divided into "blocks" $b_{i,1}, b_{i,2}, \ldots$ of c bits each. Thus all of these blocks $b_{i,j}$ will be zero, except for the one block containing a single 1.) Choose t numbers $r_0, \ldots, r_{t-1}$ in the range $1 \leq r_j \leq \log^5 n$ uniformly and independently at random (and note that this amounts to choosing $O(\log n)$ random bits). Finally, define w_i to be equal to $\sum_{j=0}^{t-1} b_{i,j} r_j$. The weight of a set A_k is then $\sum_{i \in A_k} w_i$. By Proposition 2 of [CRS93], if $\mathcal{F}$ is not empty, then there is a unique minimal weight set in $\mathcal{F}$.

This paragraph explains how to implement this system as a constant-depth circuit. Note first that for any $k \leq n$ and for any constant $l \leq \log^7 n$ there is a depth 2 circuit of PARITY gates and small-fan-in AND gates that evaluates to one iff the weight of A_k is equal to l. (The only inputs to this circuit are the $O(\log n)$ probabilistic bits. The DNF expression for this function thus can be computed by a polynomial number of AND gates feeding into a PARITY gate Since this subcircuit depends only on $O(\log n)$ bits, the fan-in of each AND gate is trivially $O(\log n)$.) Taking the AND of this circuit with the input bit x_k results in a depth three circuit that evaluates to one iff $A_k \in \mathcal{F}$ and the weight of A_k is equal to l. Thus there is a polynomial-size depth-4 circuit with a PARITY gate at the root that evaluates to one iff there are an odd number of sets in $\mathcal{F}$ that have weight l. Hence there is a uniform depth-5 circuit with an OR at the root that evaluates to 1 iff there is some weight l such that there are an odd number of sets in $\mathcal{F}$ having weight l. By the remarks in the preceding paragraph, if the OR of $x_1, \ldots, x_n$ evaluates to one, then with probability at least one half, our depth-5 circuit will also. (Clearly, if the OR is zero, then the depth-5 circuit also evaluates to zero.) If we replace the OR gate at the root with AND and negate each of the PARITY gates that feed into that OR gate (by adding a constant 1 input to each) we obtain our desired circuit for the NOR function. Let us denote this circuit by $C(x, r)$.

It remains only to reduce the error probability from $\frac{1}{2}$ to $\frac{1}{n^l}$, without using too many additional probabilistic bits. Consider a graph with vertices for each of our $O(\log n)$ probabilistic sequences, the edge relation is given by the construction of an expander graph presented in [GG79], where each vertex has degree five. Inspection of [GG79] shows that there is a uniform circuit of PARITY gates and small-fan-in AND gates of polynomial size and constant depth that takes as as input one of our original probabilistic sequences r as well as a new probabilistic sequence $s \in \{1,2,3,4,5\}^{cl \log n}$ (for some constants c and l) and outputs the vertex r' reached by starting in vertex r and following the sequence of edges indicated by s. (Since this function depends on only $O(\log n)$ bits, it suffices to express the DNF using PARITY and AND.) Let this circuit be denoted by $R(r,s)$.

Thus we can construct a constant-depth circuit that computes the AND for all $i \leq cl \log n$ of $C(x, R(r, s[1..i])$ (where $s[1..i]$ denotes the prefix of s of length i, where r and s are probabilistically chosen. By Section 2 of [IZ89], this circuit computes the NOR correctly with probability $1 - \frac{1}{n^l}$. Adding a PARITY gate at the root allows us to compute the OR, as desired. This completes the proof of the lemma. ▮

Using this claim, take an AC^1 circuit, replace all AND gates by OR and PARITY gates (using DeMorgan's laws), and then replace each OR gate in the resulting circuit with the subcircuit guaranteed by the claim (for l chosen so that n^l is much larger than the size of the original circuit), with the same $O(\log n)$ probabilistic bits re-used in each replacement circuit. The result is a probabilistic, polynomial-size circuit that, with high probability, provides the same output as the original circuit. Note that replacing AND gates by $\times$ and PARITY gates by $+$, one obtains an arithmetic circuit, the low-order bit of whose output is the same as the output of the original AC^1 circuit with high probability. The degree of this circuit is $O(\log n)^{O(\log n)} = n^{O(\log \log n)}$

It remains to make the circuit deterministic. First we make use of the "Toda polynomials" introduced in [To91]. For example, there is an explicit construction in [BT91] of a polynomial P_k of degree $2k-1$ such that $P_k(y) \bmod 2^k = y \bmod 2$. (A nice alternative construction is presented in [Ko94].) If we implement this polynomial in the obvious way[6] and apply it to the arithmetic

[6]It is observed in [AG94] that this polynomial can be implemented via *uniform* constant-depth circuits.

circuit constructed in the preceding paragraph, we obtain an arithmetic circuit of degree $n^{O(\log\log n)}$ and polynomial size, whose low order bit is the same as the output of the original AC^1 circuit with high probability, and with the additional property that the other $c \log n$ low-order bits of the result are always zero (where c is the constant such that there are $c \log n$ probabilistic bits).

Now we merely make n^c copies of the circuit, with a different sequence of probabilistic bits hardwired into each copy, and add the output gates of each of those circuits. Note that the bit $c \log n$ positions to the left of the low-order bit is exactly the majority vote of these circuits, and thus is equal to the output of the original circuit. ▮

5 Optimization Classes

The class OptP was defined by Krentel [Kr88] as the class of functions that can be defined as $f(x) = \max\{y$: there is some path of M that outputs y on input $x\}$, where M is a nondeterministic polynomial-time Turing machine. The analogous class OptL was defined in terms of logspace-bounded nondeterministic machines [AJ93, AJ92]. Vinay considered the analogous class defined in terms of LOGCFL machines (nondeterministic logspace-bounded AuxPDAs running in polynomial time). Since LOGCFL is precisely the class SAC^1, he called this class OptSAC^1. However, this notation is misleading, as will be illustrated in this section. So in this paper we refer to this class as OptLOGCFL.

For any alphabet Σ, one obtains the semiring $(\Sigma^* \cup \{\perp\}, +, \times)$ where $\times$ denotes concatenation (and $\perp \times x = x \times \perp = \perp$ for all x) and $+$ denotes lexicographic maximum (where $x + \perp = \perp + x = x$ for all x). We will usually denote this semiring as (Σ^*,max,concat). We use the notation OptNC^k, OptSAC^k and OptAC^k to denote uniform $O(\log^k n)$-depth bounded, semi-unbounded and unbounded (max,concat) circuits.

While the Opt classes consist of functions from Σ^* to Σ^*, the Boolean classes map Σ^* to $\{0,1\}$. Nonetheless, we may talk of an optimizing function f belonging to a Boolean class of functions $\mathcal{B}$ in the following sense: $f \in \mathcal{B}$ if the language $L_f = \{\langle x, i, b\rangle :$ the ith symbol of $f(x)$ is $b\}$ is in $\mathcal{B}$.

5.1 Relating Optimization Classes and (max,concat) Circuits

It is shown in [AJ92] that OptL is contained in AC^1 (a later proof may be found in [ABP92]); it is also shown in [AJ92] that iterated matrix multiplication over (Σ^*,max,concat) is complete for OptL. As was pointed out by Jenner [Je93], it is not hard to use the techniques of [Ve92, To92] to show:

Proposition 5.1 *OptL is the class of functions computed by uniform families of skew arithmetic circuits over* (Σ^*, *max,concat*).

In [Vi91] it was claimed that OptLOGCFL coincides with OptSAC^1, but this claim was later retracted [Vi91a]. Instead, the following is easy to show (using the techniques of e.g., [Vi91, Ve91]):

Proposition 5.2 *OptLOGCFL is the class of functions computed by uniform families of arithmetic circuits of polynomial degree over* (Σ^*, *max,concat*).

Proof: ($\subseteq$) Let AuxPDA M be given; it can be assumed that the worktape of M keeps track of

- the number of output symbols that have been produced thus far in the computation, and
- the number of steps executed so far.

Also assume that each time an output symbol is produced, it is preceded by a push and followed by a pop, and that the stack changes height by one on all other moves.

The circuit we build will have gates with labels of the form (C, D, i, j, a, b), which should evaluate to the maximum of all words w that can be produced as bits i through j of a string output in a segment of computation beginning at time a and ending at time b, beginning in surface configuration C and ending in surface configuration D, where (C, D) is a realizable pair. The leaves will be of the form $(C, D, i, i + c, a, a + 1)$ (where $c \in \{-1, 0\}$) which will evaluate to $\sigma \in \Sigma \cup \{\lambda\}$ if M can move in one step from C to D outputting σ (and i, c and a agree with C and D) and will evaluate to $\perp$ otherwise; note that this leaf will depend on the input. (Strictly speaking, this "leaf" will be implemented by a subcircuit of the form $\Sigma_{a\in\Sigma}[x_j, a, \sigma_a, \perp]$.)

Non-leaf nodes of the form (C, D, i, j, a, b) are the maximum over all E, F, k, and c $(a+1 \leq c \leq b-2)$ of concat$((C, E, i, k, a, c), (E, D, k+1, j, c+1, b))$ and $(E, F, i, j, a+1, b-1)$ (where in this last expression only those E and F are considered where $C \vdash E$ via a push and $F \vdash D$ via a pop of the same meta-symbol). Standard analysis ([Co71]) shows that gates defined in this way have the properties outlined in the preceding paragraph. The output of the circuit is the maximum over all m of $(C_{init}, D_{accept}, 1, m, 1, n^k)$. The degree of any node (C, D, i, j, a, b) can be seen to be $b - a$, and thus is polynomial.

($\subseteq$) This direction is also completely standard. The AuxPDA will start exploring the circuit C_n at the root. To explore a $\times$ gate, put the right child on the stack and explore the left child. To explore a $+$ gate, nondeterministically choose a child and explore it. To explore a leaf, output the value of the leaf (this might depend on the input); then pop the top node off the stack and explore it. (If a $\perp$ is encountered, halt and reject.)

It is easy to see by induction that the time required to explore a gate g is $O(\text{depth}(g) \times \text{degree}(g) \times t(n))$ where t is the time required to check connectivity between gates. Thus the entire running time is polynomial. ∎

Since OptSAC^1 has polynomial degree, it is contained in OptLOGCFL. We investigate below the extent to which OptLOGCFL itself can be characterized in terms of circuits of small depth. No parallel algorithm for OptLOGCFL is presented in [Vi91a], and in fact this is explicitly listed as an open problem there; instead, attention is drawn to the negative results of [Ni91, Ko90] showing that depth reduction is not possible in general for non-commutative semirings.

5.2 Depth Reduction for (max,concat) Circuits

In this section we show that (max,concat) circuits of polynomial size and degree can be simulated by (max,concat) circuits of polynomial size and logarithmic depth, when unbounded fanin gates are allowed. In other words, we show that OptLOGCFL is contained in OptAC^1, and hence in OptNC^2. Since functions with quasi-polynomial degree can be computed in OptAC^1 (and hence there are functions $f \in$ OptAC^1 with $|f(x)|$ not polynomial in $|x|$), OptLOGCFL $\subseteq$ OptAC^1 is a proper containment.

Our first proof of this depth reduction was rather complicated, and was similar in spirit to the proof given section 3 for the commutative case. Although we feel that a proof in this vein is instructive, the proof given below is extremely simple, and is very similar to the argument in [ABP92]. Further, the proof we give here explicitly uses the algorithm from [MRK88] to construct individual bits of the output; it thus follows from this construction (Lemma 5.5) that OptLOGCFL is in AC^1.

Note that, in order to achieve logarithmic depth, we see no way to avoid using unbounded fanin concat gates; it remains an open question if the equalities LOGCFL = SAC^1 and #LOGCFL = #SAC^1 translate to the (max,concat) setting as OptLOGCFL = OptSAC^1. In section 6.4 we will describe a restriction on the AuxPDAs that characterizes OptSAC^1.

Theorem 5.3 *If f is computed by a family of arithmetic circuits over (Σ^*, max, concat) of polynomial size and degree, then f is computed by a family of arithmetic circuits over (Σ^*, max, concat) of polynomial size with depth $O(\log^2 n)$. (In fact, f is computed by a family of unbounded-fan-in arithmetic circuits of logarithmic depth, and f is also in the Boolean class AC^1.)*

Proof: The outline of our proof is as follows: Given an input x and a polynomial size, polynomial degree circuit, we first convert the circuit to a normal form guaranteed by Lemma 5.4. We then build an equivalent circuit over the (commutative) semiring (**Z**,max,plus), and evaluate this circuit using the [MRK88] algorithm (which in this setting can be implemented in AC^1). This is described in Lemma 5.5. Finally, we turn this AC^1 algorithm into a family of arithmetic circuits.

The following definition is similar to the notion of a proof tree or "accepting subtree" studied in [VT89]: Let g be a $+$ gate, and let h be an input to g, and consider the behavior of the circuit when given some input x. We say that h *contributes* to the value of g if the value of h is equal to the value of g (that is, the value of h is the largest value that is input to g). More generally, we say that a gate g contributes to the value of g' (where g' is not necessarily adjacent to g) if there is a path from g to g' such that every edge $h \to h'$ on this path (where h' is a $+$ gate) has the property that h contributes to the value of h'. We say that g contributes to the value of the circuit if it contributes to the value of the output gate. A *contributing subcircuit at h* is a subcircuit where each $+$ gate has one child and each $\times$ gate has two children, and all nodes in the subcircuit contribute to the value of h.

The following lemma is immediate from the proof of Proposition 5.2.

Lemma 5.4 *For any circuit family of polynomial size and degree, there is an equivalent circuit family of polynomial size and degree such that each node (other than the output node) is labeled with a pair i, j, and if node h is labeled with i, j, then it contributes to the value of the circuit only if the value of h is equal to symbols i through j of the output. (For convenience later on, we will number symbols from the right, starting with position 0 at the rightmost end.)*

Assume that the alphabet $\Sigma = \{\mathbf{0},\mathbf{1}\}$; the argument for other alphabets is similar.

Lemma 5.5 *If f is computed by a family of arithmetic circuits over $(\Sigma^*$, max, concat) of polynomial size and degree, then L_f is in AC^1.*

Proof: Let input x and circuit C_1 be given. Replace each leaf of C_1 that evaluates to $\mathbf{1}$ ($\mathbf{0}$) with a pair of leaves evaluating to $\mathbf{11}$ ($\mathbf{10}$); this has the effect of forcing any output of non-zero length to have some 1's in it. Call this new circuit C. Let n^k be an upper bound on the number of bits in the output of $C(x)$ (this follows from the degree bound on C).

Now build a (max,plus) circuit (operating over the integers) as follows.[7] Recall that each leaf node of C is labeled with a pair as in Lemma 5.4. For each leaf labeled with pair (i, i) that evaluates to 1, change that leaf to the number 2^i. (Note that 2^i can be computed by a subcircuit of depth $\log i$.) Any leaf that evaluates to $\perp$ will be replaced by a leaf evaluating to -2^{1+n^k}. All other leaves receive the value 0. Call the new circuit C'.

It is now easy to observe that the output of C' is the number whose binary representation is the value of the output gate of C.

The individual bits of the output of circuit C' can be evaluated using the algorithm of [MRK88], which consists of $O(\log n)$ applications of a routine called *Phase*. A single application of *Phase* consists of matrix multiplication over

[7]Formally, it is necessary to include the element $-\infty$ in order to make this structure a semiring. This is irrelevant for our purposes.

($\mathbf{Z}$,max,plus), and hence can be done in AC^0. Thus $O(\log n)$ applications of *Phase* can be done in AC^1, resulting in an AC^1 circuit computing the function computed by C'.

This shows that the language $L_{f'} = \{\langle x, i, b\rangle : \text{the } i\text{th bit of } C'(x) \text{ is } b\}$ is in AC^1. A trivial modification now shows that $L_f = \{\langle x, i, b\rangle : \text{the } i\text{th bit of } f(x) \text{ is } b\}$ is in AC^1. ▮

Now we can build log-depth arithmetic circuits over (max,concat) for C_1 in an essentially trivial way. Namely, note that ($\{\perp, \lambda\}$,max,concat) is isomorphic to $(\{0,1\}, \vee, \wedge)$. Thus we can build log-depth arithmetic circuits (using unbounded fanin max and concat gates) of the form $[i, b]$ that evaluate to λ if (x, i, b) is in L_f, and evaluate to $\perp$ otherwise. The final arithmetic circuit is the maximum (over all output lengths m) of the result of concatenating (for $m \geq i \geq 1$) the maximum over all bits b of concat$(b, [i, b])$. ▮

6 Depth Reduction in Non-Commutative Settings

In the commutative setting, any poly-degree uniform circuit can be depth-reduced (Theorem 3.1). In the non-commutative case, we know of one semi-ring where this holds (the (max,concat) circuits, described in Section 5), and another where it does not (the (union,concat) circuit lower bounds from [Ko90, Ni91]). Where exactly does the proof of Theorem 3.1 break down in this setting?

Let us say that a circuit is "right-lopsided" if at each $\times$ gate, the degree of the left child does not exceed the degree of the right child. The construction in Theorem 3.1 (in fact, all depth-reduction constructions so far) uses right-lopsidedness to make depth reduction possible — the heavier child on the right is stacked while the left child is processed. Since the circuit is of polynomial degree, this ensures that the stacking level is only logarithmic. (In general, the stacking or recursion level is log(degree).) This then is the role commutativity plays — it allows any $\times$ gate to be rewritten in a right-lopsided way (as we assumed in the proof of Theorem 3.1).

Clearly, then, in the non-commutative case, if we are given a circuit which is of polynomial degree and is already right-lopsided, the same construction goes through.

In this section, we show that for non-commutative circuits, right-lopsidedness is not necessary (though sufficient) for depth reduction. We identify a property called short-left-paths (of which right-lopsidedness is a special case), and show that poly-degree circuits with this property can be depth-reduced. We even show that the symmetric property of short-right-paths (and hence left-lopsidedness) suffices. The resulting depth-reduced circuits are semi-unbounded, and we also characterize them via generalized AuxPDA machines.

6.1 Sufficient Conditions for Depth reduction

We consider a generalization of left-skewness called *short-left-paths*, defined in terms of a new labeling scheme. For any proof tree in an algebraic circuit, consider the labeling of each gate by an integer according to the following rules. The root gate is labeled 0, all children of a $+$ gate labeled k are labeled k, the right child of a $\times$ gate labeled k is labeled k, and the left child of a $\times$ gate labeled k is labeled $k+1$. It is easy to determine the label of any node in a given proof tree according to this scheme: Label each edge by N or L or R if the edge is from a $+$ gate to a child, or from a $\times$ gate to its left child, or from a $\times$ gate to its right child respectively. Then the label of the gate is simply the number of edges labeled L on the unique path from the root to this gate in the tree. (Note that in the *circuit*, the gate to which the node corresponds might be reachable by more than one path.)

In a left-skew circuit, all gates in any proof tree get labels 0 or 1. Generalizing this, we say that an algebraic circuit has short-left-paths if for every proof tree in the circuit, the maximum label used by this labeling scheme is $O(\log n)$.

If this labeling is extended to the circuit, then the label of a gate may not be uniquely defined. In this case, we make multiple copies of the gate, each with a unique label. This does not cause the size of the circuit to be non-polynomial, since the number of labels required is small.

Short-right-paths are similarly defined.

It is easy to see that for circuits of polynomial degree, right-lopsidedness implies short-left-paths, and left-lopsidedness implies short-right-paths.

The next lemma shows that lopsidedness is not necessary for depth reduction. (Recall that all previous depth-reduction results have used the notion of lopsidedness; hence lopsidedness is sufficient for depth reduction.) We show that it is sufficient that the proof trees be reduced in one direction: hence short-left-paths. One should notice that the final depth-reduced circuit has both short-left and short-right paths.

Lemma 6.1 *Let R be any semi-ring. The class of functions computed by uniform polynomial size semi-unbounded arithmetic circuits over R of depth $(O(\log n))$ is equal to the class of functions computed by uniform polynomial size arithmetic circuits over R of polynomial degree, with short-left-paths.*

Proof: The inclusion from left to right follows from the observation that semi-unbounded logarithmic depth circuits have both short-left-paths and short-right-paths. This holds even after they are converted to bounded fanin equivalents, since the conversion only increases the depth through $+$ nodes, which do not affect our labeling.

The inclusion from right to left: Let C be the polynomial-degree circuit with short-left-paths. Then by a trivial modification of the proof of Lemma 3.2 there is an equivalent uniform circuit C' where each gate description carries the unique label of the gate under the short-left-path labeling scheme, as also the algebraic degree of the gate, and no path in the circuit contains more than two consecutive $+$ gates, and if there is a path from one gate g to another gate h that encounters no $\times$ gates, then there is *exactly* one path from g to h.

Let us understand how we may accomplish the depth reduction. The task is to make *all* proof trees into small depth trees. Note that the proof trees are already short in one direction. So it is the other direction (the right paths) that needs to be compressed. The task will be accomplished by dividing the proof tree into smaller subtrees. In general, we can talk of a proof tree rooted at one gate g and terminating at another gate h, which is not necessarily a leaf. Such a proof tree evaluates to $[g, h]$, as described in the proof of Theorem 3.1. As further described there, being the terminal node of an exploration from g is equivalent to being reachable from g via a path using only $+$ gates and the *second* edges (labeled R) out of $\times$ gates. We call such a path the *current focus path (CFP)*, and concentrate on compressing this path.

Note that the length of a CFP can be a polynomial. To achieve logarithmic depth, a divide-and-conquer scheme is called for; this is precisely what the algebraic degree tag on each gate allows us to do. Consider two gates g and

h with degrees dg and dh respectively. By assumption, g and h are on a CFP rooted at g. Now, either there are $\times$ gates between g and h or there are none. In the latter case, they are said to be *adjacent* to each other.

Consider the former case. All $\times$ gates on the path from g to h have degrees in the range $[dg, dh]$ in decreasing order. Consequently, there are adjacent gates z_1 and z_2 such that their degrees satisfy

$$dz_1 \geq \frac{dg + dh}{2} > dz_2 \quad (*)$$

(It is possible that z_1 is g or z_2 is h, but obviously not both at the same time.) These gates are unique to a proof tree. The output of the proof tree (which is the product of the leaves in left to right ordering) may be decomposed into three parts: leaves encountered while traversing the tree from (1) g to z_1 (2) z_1 to z_2 (3) z_2 to h. The product of the leaves in this ordering is the product of the leaves in the traversal from g to h.

In the latter case, i.e. when g and h are adjacent, then if g is a $\times$ gate, then the leaves encountered in traversing the pruned proof tree from g to h are precisely the leaves encountered while traversing the proof tree rooted at the left child of g. So we move the CFP down by one level; the new CFP represents the proof tree rooted at the left child of g. (If g is a $+$ gate, then the leaves encountered are exactly the same as those encountered when traversing the proof tree at some gate g' adjacent to g; recall that without loss of generality the path from g to g' has length at most 2.)

Let $[g; dg, Lg]$ denote the function computed at g where dg is the degree of the gate and Lg is the number of L's in a path from the root to g. Though this information is implicit in the gate labels, making it explicit makes it easy to analyze the construction. Let $[g, h; dg - dh, Lg]$ denote the function computed at $[g; dg, Lg]$ if the proof trees are pruned at h. $[g, h, 0; dg - dh, Lg]$ is the same as $[g, h; dg - dh, Lg]$ except that additionally, g and h are adjacent on some CFP. Note that it is easy to verify the adjacency of two $\times$ gates because the number of consecutive $+$ gates on any path is bounded.

A schematic diagram of the construction is shown in Figure 1.

We may summarize the above discussion by:

$$[g; dg, Lg] = \sum_{l:\ \text{leaf}} [g, l; dg - 1, Lg] \text{ if } g \text{ is a } \times \text{ gate}$$

$$[g; dg, Lg] = \sum_{h:\text{ leaf or } \times \text{ gate adjacent to } g} [h; dh, Lh] \text{ if } g \text{ is a } + \text{ gate}$$

$$[g, h; dg - dh, Lg] =$$
$$\sum_{z_1, z_2:\text{ as in } (*)} [g, z_1; dg - dz_1, Lg] \times [z_1, z_2, 0; dz_1 - dz_2, Lz_1] \times [z_2, h; dz_2 - dh, Lz_2]$$

$$[g, h, 0; dg - dh, Lg] = \sum_{l:\text{ leaf}} [g_L, l; dg_L - 1, Lg + 1]$$

$$[g, g; 0, Lg] = g \text{ if } g \text{ is a leaf. } 1 \text{ otherwise.}$$

This describes the semi-unbounded circuit construction.

We need to argue that the circuit has small depth. Consider the potential function Ψ of a gate $[g; dg, Lg]$, defined as $\Psi = \langle \log(dg), ht \rangle$. Assuming that the degree of the circuit is n^k, and that left-paths do not exceed $c \log n$, the potential function at the output gate is $\langle k \log n, 0 \rangle$. (The output gate has zero left-path length.) The leaves have a potential of $\langle 0, c \log n \rangle$. As we come from the root to any leaf in the constructed circuit, at each $\times$ gate (i.e.. at alternate levels) either the first component decreases by 1 (the degree comes down by a factor of 2) or the second component increases by 1 (left-path length increases by 1). Consequently, the depth of the circuit is no more than $(2c + 2k) \log n$. ∎

Remark: This proof is very similar to the proof of Theorem 3.1. We represent a collection of potential proof trees, using $O(\log n)$ bits, by specifying the "root" and by the "leaf" on the CFP. Though many proof trees may have the same representation at some level, this is irrelevant because at some point of the compression, two distinct proof trees must necessarily differ. A discerning reader would notice that this representation is precisely the so-called realizable pair of surface configurations.

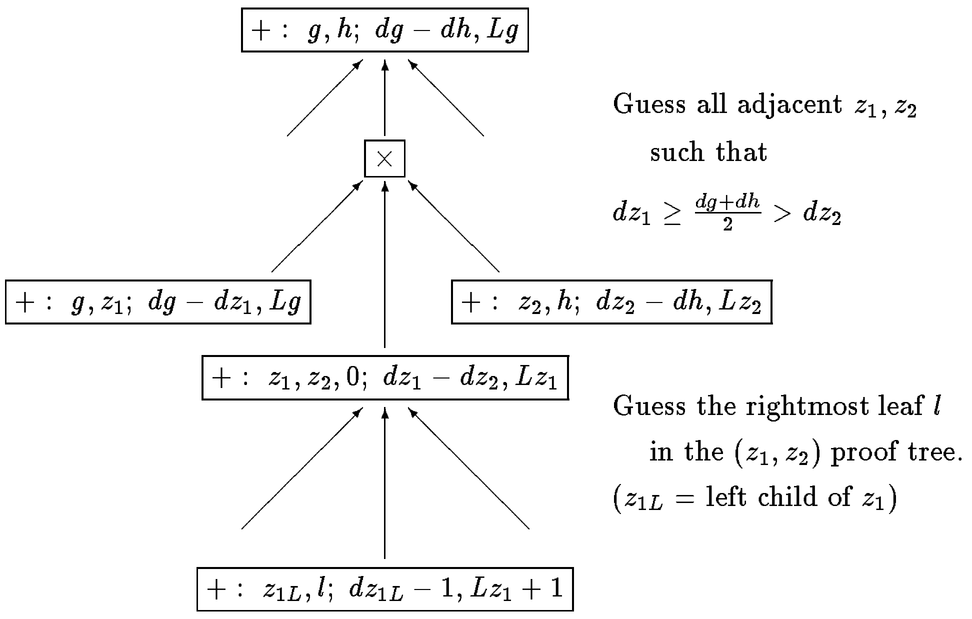
$+:\ g,h;\ dg-dh, Lg$
$\times$
$+:\ g,z_1;\ dg-dz_1, Lg$
$+:\ z_2,h;\ dz_2-dh, Lz_2$
$+:\ z_1,z_2,0;\ dz_1-dz_2, Lz_1$
$+:\ z_{1L},l;\ dz_{1L}-1, Lz_1+1$
Guess all adjacent z_1, z_2 such that $dz_1 \geq \frac{dg+dh}{2} > dz_2$
Guess the rightmost leaf l in the (z_1, z_2) proof tree. (z_{1L} = left child of z_1)

Figure 1: Depth reduction of a circuit (Lemma 6.1)

The proof of the next lemma requires a technical strengthening of the result of Lemma 6.1. Lemma 6.1 shows how to construct a log-depth semi-unbounded family $\{D_n\}$ equivalent to a given family $\{C_n\}$ with short-left-paths, in the sense that for all x of length n, $C_n(x) = D_n(x)$. In fact, we have established a somewhat stronger notion of equivalence, and this becomes important for us below. For this we need some additional definitions.

For any circuit C and input x, let C_x denote the circuit which results from replacing each leaf node $[x_i, a, b, c]$ by the element of R to which this leaf evaluates on input x. Thus C_x has no input variables, the leaves are labeled by constants, and hence it computes a constant function. Corresponding to C_x is a formal polynomial $P(C, x) = \sum_t c_t \prod(t)$; the next few sentences provide a definition of $P(C, x)$. As in the proof of Theorem 3.1, the value of C_x can be expressed as $\sum_e \prod(e)$, where the sum is taken over all explorations e, and $\prod(e)$ denotes the product of all the values encountered at leaves of C_x along that exploration. Note that some two explorations e and e' may encounter exactly the same sequence of values at the leaves they visit. In this case, we say that there is some term t such that $\prod(t) = \prod(e) = \prod(e')$. Thus we may group these terms, and express the value of C_x as $\sum_t c_t \prod(t)$, where c_t is equal to the ring element $(1 + 1 + \ldots + 1)$ that results by adding 1 j times, where j is the number of distinct explorations e such that $\prod(t) = \prod(e)$. It will cause no confusion to think of c_t as being the natural number j. (Also note that this sum is an infinite sum over *all* terms t in $\mathcal{G}^*$; for all but finitely many $t, c_t = 0$.)

Definition 6.2 *Two circuits C_1 and C_2 are said to be* strongly equivalent *if, for all x, and all terms t, the coefficients of term t in $P(C_1, x)$ and $P(C_2, x)$ differ only if 0 appears in the term t.*

To illustrate, consider the three circuits with one input variable x over the (max,concat) semiring:

$$C_1 = \max\{(\mathbf{1} \cdot x), (x \cdot \mathbf{1})\}$$

$$C_2 = \max\{(\mathbf{1} \cdot x), (\mathbf{0} \cdot \mathbf{0}), \mathbf{1}\}$$

$$C_3 = \max\{(\mathbf{1} \cdot x), (x \cdot \mathbf{1}), (\perp \cdot x)\}$$

All three of these circuits give the same output on all inputs $x \in \{\mathbf{0}, \mathbf{1}\}$. However, C_1 and C_3 are strongly equivalent, whereas C_1 and C_2 are not.

If two circuits give the same output but are not strongly equivalent, then the proof that they are not strongly equivalent usually requires detailed analysis of the underlying structure of the semiring R. Thus it is not surprising that examination of the proof of Lemma 6.1 shows that we have actually established

Lemma 6.3 *For any uniform family $\{C_n\}$ of polynomial-size arithmetic circuits over R of polynomial degree, with short-left-paths, there is a uniform family $\{D_n\}$ of semi-unbounded arithmetic circuits over R of depth $O(\log n)$, such that for all n, C_n and D_n are strongly equivalent.*

In general, the above construction converts a polynomial size circuit of degree d and largest left-path-label l to a semi-unbounded circuit of depth $O(\log d + l)$, provided $d \in O(poly)$. So some depth-reduction can be achieved even if left-path-labels are polylogarithmic.

A short-right-path non-commutative circuit can obviously have high push-down height when simulated on an AuxPDA. But we show below that it can be depth-reduced! This shows that it is sufficient for paths to be short in one of the two directions, either left or right. It also indicates the weakness of showing depth-reduction via the AuxPDA model, and the advantages of staying entirely within the circuit model, as in section 3.

Lemma 6.4 *Let R be any semi-ring. The class of functions computed by uniform polynomial size semi-unbounded arithmetic circuits over R of depth $(O(\log n))$ is equal to the class of functions computed by uniform polynomial size arithmetic circuits over R of polynomial degree, with short-right-paths.*

Proof: The inclusion from left to right follows from the observation in the previous lemma.

To show the inclusion from right to left, consider a function f computed by a uniform polynomial size arithmetic circuit C over R of polynomial degree, with short-right-paths. Let C' be the circuit that results by swapping the right and left inputs of each $\times$ gate (like a *mirror image* of C). It is clear that C' has short left paths, and thus, applying Lemma 6.1, there is a strongly equivalent C'' of logarithmic depth. Now, by swapping left and right inputs of all the $\times$

gates again, we get a circuit C'''. We show below that C''' computes the same function f that the original circuit C does. (This is easy to see intuitively. However there are some subtle points that require the formal proof below. In particular, it is important that C' and C'' be *strongly* equivalent, instead of merely producing the same output. The examples C_1 and C_2 given above show that taking the mirror images of two circuits that produce the same output may produce two circuits that do *not* compute the same function.)

As in the discussion preceding the definition of strong equivalence, for any input x, we obtain "constant" circuits C_x, C'_x, C''_x, and C'''_x such that for each x, C_x evaluates to the same element of R as does the circuit C on input x, etc.

Let $\sum_t c_t \prod(t)$ be the formal polynomial $P(C, x)$. (Note that the coefficient c_t actually depends on x. However, to simplify notation below, we supress this additional subscript.) For any term t, t^R denotes the same term multiplied in reverse order. The following simple inductive argument shows that the formal polynomial associated with $C'_x, P(C', x)$ is precisely $\sum_t c_t(\prod(t^R)$.

Basis: C_x is a single leaf: Trivial.

Inductive step: Case 1: C_x has a $+$ output gate, whose inputs are several smaller circuits $C_{x,i}$, where $P(C_i, x) = \sum_t c_{i,t}(\prod(t^R)$.

Then the formal polynomial associated with C_x, say $\sum_t c_t \prod(t)$, is equal to $\sum_i (\sum_t c_{i,t} \prod(t))$. Thus $c_t = \sum_i c_{i,t}$. Similarly, considering the formal polynomial for $C'_{x,i}$, we have $c'_t = \sum_i c'_{i,t}$.

By the induction hypothesis, the formal polynomial associated with each $C'_{x,i}$ is $\sum_t c_{i,t} \prod(t^R)$. Thus $c'_{i,t} = c_{i,t^R}$, and the claimed result follows.

Case 2: C_x has a $\times$ output gate, whose inputs are sub-circuits $C_{x,1}$ and $C_{x,2}$. Then the formal polynomial associated with C_x, $\sum_t c_t \prod(t)$, is precisely $(\sum_{t1} c_{1,t1} \prod(t1)) \times (\sum_{t2} c_{2,t2} \prod(t2))$, which equals $\sum_{t1\times t2}(c_{1,t1} \times c_{2,t2}) \prod(t1 \times t2)$. Doing the same analysis for C'_x shows that it has formal polynomial $\sum_{t1\times t2}(c_{1,t1} \cdot c_{2,t2}) \prod(t2^R \times t1^R)$

This completes the induction. Note that this also establishes that if $P(C''', x)$ is equal to $\sum_t c'''_t \prod(t)$, then $P(C'', x) = \sum_t c'''_t \prod(t^R)$.

Now, for every input x, the value of $C(x)$ is $P(C, x) = \sum_t c_t(\prod(t))$, and the value of $C'''(x)$ is $P(C''', x) = \sum_t c'''_t(\prod(t))$. To prove that C and C''' produce the same output on each input, it thus will suffice to show that for any term t

that does not contain 0, for each x, the coefficients c_t and c_t''' in $P(C,x)$ and $P(C''',x)$, respectively, are equal.

But we do know this, because $P(C',x) = \sum_t c_t(\prod(t^R))$, and $P(C'',x) = \sum_t c_t'''(\prod(t^R))$, and C' and C'' are strongly equivalent, meaning that for any term t^R that does not contain 0, c_t and c_t''' are equal.

∎

Theorem 6.5 *Over any semiring, the classes of functions computed by the following are equal.*

1. *Polynomial size circuits of polynomial degree, with short-left-paths.*

2. *Polynomial size circuits of polynomial degree, with short-right-paths.*

3. *Polynomial size semi-unbounded circuits of logarithmic depth.*

In particular, for left-skew or right-skew circuits, depth reduction is always possible. This cannot be extended to skew circuits, since the linear depth lower bound given by [Ko90, Ni91] is for a function with skew circuits.

6.2 The Generalized LOGCFL model

In the case of the Boolean ring and computation over integers, we know the relationship between SAC^1 circuits and LOGCFL machines (AuxPDAs) [Ve91, Vi91]. To extend the analogy to the general setting, we consider a generalized definition of LOGCFL machines. The machine computes a function $f(x_1,\ldots,x_n)$ in the following manner.

- The machine takes $x_1,\ldots,x_n$ as input, where x_i belongs to the (finite) set of generators of the appropriate semi-ring.

- The output symbols produced by the machine belong to the finite set of generators or constants of the semi-ring, or are projections of some input value.

- Let ρ be a computation path of the LOGCFL machine. Let $\rho_1\rho_2\ldots\rho_m$ be the sequence of output symbols written along this path. Then, over any semi-ring, the generalized LOGCFL machine is said to compute the function

$$\sum_{\rho\text{: a valid path}} \rho_1 \times \rho_2 \times \ldots \times \rho_m$$

Recall that the LOGCFL machine is said to be $h(n)$-height-bounded if, on all computation paths of the machine, the height of the pushdown never exceeds $h(n)\log n$ (ie, the stack never contains more than $h(n)$ meta-symbols, each $\log n$ symbols long).

We show that bounding pushdown height in LOGCFL machines to $O(\log n)$ meta-symbols corresponds exactly to restricting circuits to having short-left-paths. This then gives a machine characterization of SAC^1 circuits over the appropriate semi-ring.

Lemma 6.6 *Let R be any semi-ring, not necessarily commutative. The class of functions computed by uniform polynomial size arithmetic circuits over R of polynomial degree, with short-left-paths, is equal to the class of functions computed by generalized* $\log n$*-height-bounded LOGCFL machines.*

Proof: (1) Left-to-right inclusion:
From Lemma 6.1, it suffices to show that semi-unbounded logarithmic depth circuits can be simulated by $\log n$-height-bounded generalized LOGCFL machines.

The generalized LOGCFL machine begins with the description of the root gate on its worktape. Let the gate described on its worktape be g. If g is a $+$ gate, it nondeterministically chooses a child g' of g, and replaces g on its worktape with this gate. If g is a $\times$ gate with children g_L and g_R, it stacks g_R and replaces g on its worktape with g_L. If g is a leaf evaluating to a (this may depend on the input), it outputs a, and replaces g on its worktape with the gate label topmost on the stack. The machine halts when it has just processed a leaf and the stack is empty.

Clearly, this machine computes the same function as the circuit. Each exploration of the LOGCFL machine corresponds to a proof tree in the circuit. Since the circuit has polynomial degree, any exploration of the LOGCFL machine is polynomial-time-bounded. Also, since the circuit has logarithmic depth, it is straightforward to see that the LOGCFL machine so defined is $\log n$-height-bounded. (Only right children are stacked, so the stack has at most one gate at each level.)

(2) Right-to-left inclusion:
Fix a LOGCFL machine. Without loss of generality assume that (a) the machine halts in a unique configuration, C_{fin} (b) the machine pushes or pops on every move, so the total number of moves is even, (c) the machine outputs something only on *peak* configurations. (A configuration is a peak configuration if the previous step is a push and the next step is a pop.)

The circuit has gates with labels of the form (P, Q, h, p, q). The labels have the following interpretation: P is a surface configuration at time p, Q is a surface configuration at time q, and (P, Q) is a realizable pair with common pushdown height h.

If $q - p = 2$, then in the circuit the gate (P, Q, h, p, q) is represented by a constant-depth sub-circuit. This sub-circuit uses predicates $[x_i, a, \lambda, \perp]$ to simulate the LOGCFL machine's access to its input. The entire sub-circuit evaluates to $\perp$ if there is no legal sequence of 2 moves from P to Q on the given input. Otherwise, there is a peak configuration between P and Q; this peak configuration produces a symbol c, or λ, as output. In this case, the sub-circuit also evaluates to c or λ. The construction of this sub-circuit is straightforward but tedious and is omitted here.

If $q - p > 2$, then we focus on the profile of the potential computation. A *profile* of a computation sequence is a graph depicting the behaviour of the pushdown height over time for that computation sequence. Fix a profile for the realizable pair (P, Q, h, p, q). There are now two possibilities.

Firstly, the pushdown height may be strictly greater than h throughout the computation from P to Q (excluding the endpoints, of course). In such a case, we can find a realizable pair (Z_1, Z_2), such that P pushes some meta-symbol b to reach Z_1 in one step, and Z_2 pops the same meta-symbol b to reach Q in one step. Besides, there will be at least two moves in the sequence from Z_1 to Z_2 (since $q - p > 2$), and both Z_1 and Z_2 will have pushdown height $h + 1$. In the circuit, we represent this possibility by a sum (guess), over all choices of Z_1 and Z_2, of the product of two sub-circuits: the right sub-circuit has its root labeled by $(Z_1, Z_2, h+1, p+1, q-1)$, and on the left is a constant-depth sub-circuit evaluating to λ if $P \overset{push(b)}{\longrightarrow} Z_1$ and $Z_2 \overset{pop(b)}{\longrightarrow} Q$ (where b is the top-of-stack symbol in Z_1 and Z_2) and to $\perp$ otherwise. (The left sub-circuit validates the moves from P to Z_1 and from Z_2 to Q).

Secondly, there may be a surface configuration Z such that (P, Z) and (Z, Q) are realizable pairs. In such a case, there is a Z closest (in time) to P; let this Z occur at time t, $p < t < q$. While it is quite correct to say that (P, Q, h, p, q) should evaluate to the product of (P, Z, h, p, t) and (Z, Q, h, t, q), this will not ensure that Z is closest to P. We would like to ensure this, because it will let us keep the circuit depth down. So in this product, we directly represent the left term by its expansion, assuming that there is no surface configuration at the same height as, and between, P and Z. The expansion is as described in the previous paragraph. In other words, we represent this possibility by the sum, over all choices of Z_1, Z_2, Z and t where $p < t < q$, of the product of the following 3 terms: (1) $(Z_1, Z_2, h+1, p+1, t-1)$, (2) a constant-depth sub-circuit evaluating to λ if $P \overset{push(b)}{\longrightarrow} Z_1$ and $Z_2 \overset{pop(b)}{\longrightarrow} Z$ (where b is the top-of-stack symbol in Z_1 and Z_2) and to $\perp$ otherwise, and (3) (Z, Q, h, t, q).

There is one case where the above construction will not work. If the guessed Z is just 2 moves away from P, then by expanding the P-to-Z computation one step further, we end up with a gate labeled $(Z_1, Z_2, h+1, t-1, t-1)$. This means that Z_1 should be the same as Z_2, and is a peak configuration which could potentially produce some output. This output symbol is not accounted for in the above construction. To avoid this situation, we allow, in the above sum, only configurations far away from P; t should satisfy $p+2 < t < q$. We also have a third independent possibility, when $t = p+2$. This is represented by the sum, over all choices of Z, of the product of $(P, Z, h, p, p+2)$ and $(Z, Q, h, p+2, q)$.

Finally, the gate (P, Q, h, p, q), is the sum of the three circuits described in the preceding three paragraphs.

We have described how to build up a circuit rooted at a gate labeled (P, Q, h, p, q). To complete the construction, we observe that the output gate of the desired circuit carries the label $(C_{in}, C_{fin}, 0, 0, n^k)$.

It is clear that the resulting circuit has polynomial size and polynomial degree and computes the same function as the AuxPDA. (The degree of each gate is related to the number of output symbols accounted for by the sub-circuit rooted at that gate.) The labels have been assigned so that at each $\times$ gate, the left child is either a constant-depth sub-circuit, or has an increased value of the parameter h. Note that h increases as we go down from root to leaf. Since the AuxPDA is $O(\log n)$-height-bounded, the value of h is bounded by $O(\log n)$; it follows that every proof tree in the circuit has short-left-paths. ▮

Theorem 6.7 *Over any finitely-generated semiring, the classes of functions computed by the following are equal.*

1. *Polynomial size circuits of polynomial degree, with short-left-paths.*
2. *Polynomial size circuits of polynomial degree, with short-right-paths.*
3. *Polynomial size semi-unbounded circuits of logarithmic depth.*
4. $\log n$*-height-bounded generalized LOGCFL machines.*

6.3 Short Paths in Different Directions Intertwined

In the next theorem, we show that a small number of nested subcircuits that alternately satisfy the short-left-paths and short-right-paths conditions allow depth reduction. This allows many circuits to be depth-reduced.

Given a circuit C with a gate g in it, let C_g denote the subcircuit of C rooted at g. Then by $[C : C_g]$, we mean the circuit in which the subcircuit C_g is excluded from C. This circuit has, as circuit inputs, the circuit inputs of C and new variables representing the outputs of gates in C_g. If C_g has size $s(n)$, then $[C : C_g]$ has $n + s(n)$ circuit inputs. We restrict our attention to polynomial-size circuits, so $s(n)$ is always bounded by a polynomial.

If gates $g_1, \ldots, g_t$ are gates in C such that $C_{g_1}, \ldots, C_{g_t}$ are mutually disjoint, then the notation $[C : C_{g_1}, \ldots, C_{g_t}]$ is the natural extension of $[C : C_g]$; it represents the circuit where $C_{g_1}, \ldots, C_{g_t}$ are excluded and outputs of gates in these subcircuits are replaced by new variables.

Now consider the case when $[C : C_{g_1}, \ldots, C_{g_t}]$ has short-left-paths but $C_{g_1}, \ldots, C_{g_t}$ have short-right-paths. We say that the circuit C has intertwining depth 1. Similarly if $[C : C_{g_1}, \ldots, C_{g_t}]$ has short-right-paths but $C_{g_1}, \ldots, C_{g_t}$ have short-left-paths, then again the intertwining depth is 1. (A circuit which has short-left- or short-right- paths has intertwining depth zero.) If $C_{g_1}, \ldots, C_{g_t}$ themselves have intertwining depth k, then C has intertwining depth $k+1$.

Theorem 6.8 *Let C be a polynomial size polynomial degree circuit with intertwining depth $k(n)$. Then there is an equivalent polynomial size semi-unbounded circuit C', of depth $O(k(n) \log n)$, computing the same function.*

Sketch of proof: We consider the case when the intertwining depth is 1 and there is only one nested gate g (i.e. $t = 1$). The construction of Lemma 6.1 can be applied to reduce the depth of $[C : C_g]$, giving circuit $[C' : C_g]$. For each gate h in C_g, consider the circuit $C_g(h)$ which is the same as C_g but has h as the output gate. Lemma 6.4 can be applied to each such circuit to reduce its depth, giving circuit $C'_g(h)$. Patch the circuit $[C' : C_g]$ by putting a copy of the circuit $C'_g(h)$ at those inputs which correspond to the variable representing gate h. The resulting circuit is still of polynomial size, and its depth is depth($[C' : C_g]$) + $\max_h$depth($C'_g(h)$). ▮

Note that the above construction is nonuniform because the intertwining structure has to be given as an advice to the constructor of the depth-reduced circuit.

Corollary 6.9 *Let C be a polynomial size polynomial degree circuit.*
(1) If C has $O(1)$ intertwining depth, it has an equivalent log-depth circuit.
(2) If C has $O(\log^{O(1)} n)$ intertwining depth, it has an equivalent polylog-depth circuit.

This result cannot be further improved to circuits which have linear intertwining but sublinear depth, because, referring back to the function cited in [Ko90, Ni91], the corresponding skew circuit has $O(n)$ intertwining depth.

6.4 A New Optimization Class and its Circuit Characterization

Consider a restricted version of OptLOGCFL, where the underlying AuxPDA transducer's pushdown height is bounded by $O(\log n)$. (Recall that each stack symbol is assumed to be $O(\log n)$ bits long. Thus effectively the stack holds $O(\log^2 n)$ bits.) We denote this class by R-OptLOGCFL. If we consider the Boolean or counting versions of LOGCFL, then restricting the pushdown to logarithmic depth does not make the class any weaker; see [Vi91, Lemma 3.1]. However for the optimizing functions it may make a difference.

From the resource bounds on the transducers computing these optimizing functions, it is clear that OptL $\subseteq$ R-OptLOGCFL $\subseteq$ OptLOGCFL. The next result follows from Theorem 6.7, and gives a circuit characterization of the class R-OptLOGCFL as OptSAC^1.

Theorem 6.10 *R-OptLOGCFL = OptSAC^1; a function is computed by a uniform family of polynomial size log depth semi-unbounded (max,concat) circuits iff it is computed by a generalized LOGCFL machine, over (max,concat), whose pushdown height is bounded by $O(\log n)$.*

Recall, from section 5.1, that OptL corresponds to skew circuits, and OptLOGCFL to polynomial degree circuits. Thus, in terms of circuits, the containment OptL $\subseteq$ R-OptLOGCFL $\subseteq$ OptLOGCFL says that skew (max,concat) circuits can be converted to semi-unbounded logarithmic depth (max,concat) circuits, which in turn, are of polynomial degree.

Note that a (max,concat) circuit can be converted to a family of Boolean circuits in a trivial way: replace max gates by equivalent AC^0 circuits, and represent concat by juxtaposition of wires. (We need to assume that the length of the output of each concat gate is fixed and known.) This operation, on an OptSAC^1 circuit, yields an AC^1 circuit, giving an alternative proof that OptL and OptSAC^1 are in AC^1. This direct conversion predictably fails for OptAC^1; we end up with a log depth but quasi-polynomial size circuit.

7 Circuit Size Lower Bounds for (union,concat) Generator Circuits

For any alphabet Σ, consider the semiring $(2^{\Sigma^*}, +, \times)$, where $+$ denotes set union and $\times$ denotes set concatenation. We will consider arithmetic circuits over this semiring, where each gate in the circuit evaluates to a subset of Σ^*. We consider $\Sigma = \{\mathbf{0}, \mathbf{1}\}$. The empty set $\emptyset$ is the additive identity or bottom element $\perp$, and $\{e\}$, the set containing the empty string, is the multiplicative identity λ. $\{\{\mathbf{0}\}, \{\mathbf{1}\}, \perp, \lambda\}$ is a finite set of generators, and the input gates are labeled by elements of this set.

As in previous work on this semiring ([Ko90, Ni91]), our interest will focus on the ability of circuits of this sort to *generate* languages, as opposed to computing functions. More precisely, consider a circuit family $\{C_n\}$ over this semiring where each C_n computes a *constant* function. (That is, none of the n variables for C_n are connected via any path to the output gate; note that for different n, the circuit C_n may be computing a different output, and fairly large size may be required to compute this output.) If each C_n produces a set $A_n \subseteq \Sigma^n$ as output, then we will say that the family $\{C_n\}$ *generates* the language $\bigcup_n A_n$.

Note that if $\{C_n\}$ generates a set A, then the formal polynomial $P(C_n, w)$ does not depend on the word w (because C_n has no input variables) and thus we will denote this formal polynomial as $P(C_n)$, and we can write this polynomial as $P(C_n) = \bigcup_w c_w\{w\}$, where c_w is the number of distinct explorations of C_n that visit leaves whose product is the set $\{w\}$. (Although in this semiring, $0 = \bot = \emptyset$ and $1 = 1 + 1 = \ldots = \{e\} = \lambda$, it will be more useful to us to continue to view c_w as a natural number.) Thus for every word w of length n, $w \in A$ iff the coefficient c_w in the formal polynomial $P(C_n)$ is greater than zero.

We say that $\{C_n\}$ generates A *unambiguously* if $w \in A$ implies $c_w = 1$, and $c_w = 0$ otherwise.

The reason we are interested in (union,concat) generators is that the only explicit depth lower bounds known for non-commutative computation have been shown for such generators [Ko90, Ni91]. In particular, in [Ko90] and [Ni91] it is shown that the language $L_1 = \{ww^R | w \in \{0,1\}^*\}$ has no sub-linear-depth generator of any size. Nisan further extends the argument to show that there are no sub-linear-depth generators for the non-commutative permanent and determinant, or even for any function weakly equivalent to these functions. The proof technique used is to relate the branching program size $B(f)$ of a homogeneous degree d function f, the formula size $F(f)$, and the depth $D(f)$ as follows: $B(f) \leq O(dF(f))$, and $F(f) \leq 2^{D(f)}$. Then, through matrix rank arguments, a lower bound on $B(f)$ is shown.

The above technique does not yield any lower bound on circuit size. However, we observe that a branching program is nothing but a left-skew circuit; thus the matrix rank argument does give a lower bound on left-skew circuit size.

In this section, we extend this argument to skew circuits which are not necessarily left-skew but nonetheless have a fixed pattern among the $\times$ gates. We call such circuits *clone-skew circuits*, since each proof tree of the circuit shows the same pattern of skew gates. However, before formally defining clone-skew circuits, we first establish some useful connections between one-way language acceptors and (union,concat) generators.

7.1 One-way Acceptors and (union,concat) Generators

In this subsection we explore the connection between language acceptors with one-way read-only input tapes and (union,concat) generator circuits. The following lemmas show that left-skew generators generate exactly the languages accepted by 1-NLOG machines, and the languages generated unambiguously are exactly the languages accepted by 1-ULOG machines (where 1-NLOG (1-ULOG) refers to logspace-bounded machines that have a one-way input head, and are nondeterministic (respectively, nondeterministic with at most one accepting path on any input)). (This is similar to the relationship between skew circuits and NLOG or OptL). The one-way read-only nature of the input tape guarantees left-skewness.

Lemma 7.1 *A language L is accepted by a 1-NLOG machine iff there is a uniform polynomial size left-skew (union,concat) circuit that generates it.*

Proof: Let M be a 1-NLOG machine accepting L. We construct a polynomial size circuit C generating L as follows. Without loss of generality we assume that in each non-halting configuration of M, M either reads an input bit and changes state, or changes the scanned tape symbol and state, but not both. Thus each configuration is either a Read configuration or a Move configuration. We further assume that Read configurations are deterministic.

The circuit C has gates labeled by configurations of M; there are polynomially many of them. Let c be a Move configuration with successors $\{c_1, \ldots, c_k\}$. Then in C, c is a union gate with children labeled by $\{c_1, \ldots, c_k\}$.

If c is a Read configuration, let c_i be the resulting configuration when the input bit read is i ($i \in \{0, 1\}$). Then in C, c is a union gate with two children, c'_0 and c'_1. Each c'_i is a concat gate with left child receiving the constant i and right child the gate labeled c_i. If c is a Halt configuration, then the gate c is an input gate, receiving the constant λ ($\perp$) if c is an accepting (rejecting) configuration. (Note that the machine would actually know that it has reached the end of its input by reading an end marker. In our setting we will label a "move" configuration as accepting or rejecting depending on whether the machine would accept if it were to read the end-marker after reading the nth symbol; we can assume that the machine records the number of symbols read. Details are routine and are left to the reader.)

It is straightforward to see that this circuit is left-skew and logspace uniform. Let $w \in L$. Then there is at least one computation path of M leading to acceptance. The corresponding path in C ensures that c_w is a non-zero coefficient in the formal polynomial of C. In fact, the coefficient c_w in $P(C)$ is exactly the $\#L$ function computed by M on w.

For the converse inclusion, let $\{C_n\}$ be a uniform family of left-skew circuits generating L. Our 1-NLOG machine will guess the length n of its input, and will store the output gate of C_n on its tape. It then begins an exploration of C_n as follows. If the current gate g being explored is a $+$ gate, then it guesses a gate that is input to g and stores that on its tape. If the current gate g is a $\times$ gate, then if the left input of g does not match the next input symbol, the machine halts and rejects. Otherwise, it stores the gate h that is the right input to g on its tape, and proceeds to explore g. If g is a leaf, then we halt and accept iff the symbol input to g is the next unread symbol, as well as being the nth and final input symbol. It is straightforward to verify that accepting computation paths correspond to explorations evaluating to words in L. ▌

Corollary 7.2 *A language L is accepted by a 1-ULOG machine iff there is a uniform polynomial size left-skew (union,concat) circuit that generates it unambiguously.*

Since left-skew circuits have short-left-paths, it follows from Theorem 6.7 that all languages in 1-NLOG have polynomial size logarithmic depth semi-unbounded generator circuits. (The reduced-depth circuit is no longer left-skew; in fact it is not even skew.)

As an example of the computational power of left-skew circuits, we mention the by-now-standard example from [AJ93]: the set of all unsatisfying assignments of a 3SAT formula can be computed by the above circuits. This is because a 1-NLOG machine can recognize $\langle F, u \rangle$, where u is an assignment that does not satisfy the 3SAT formula F.

Similarly, we can relate AuxPDAs with one-way inputs to polynomial-size generators. The algebraic degree of the generators is related to the run-time of the AuxPDAs. Let 1-AuxPDA denote the class of languages accepted by one-way nondeterministic logspace-bounded auxiliary pushdown automata, and let 1-LOGCFL denote the class accepted by 1-AuxPDAs that run in polynomial time.

Lemma 7.3 *A language L is be accepted by a 1-LOGCFL machine iff there is a polynomial size, polynomial degree circuit generating L. A language L is be accepted by a 1-AuxPDA machine iff there is a polynomial size circuit generating L.*

Proof: Let L be accepted by a 1-LOGCFL machine M. We sketch how to modify the proof of Lemma 6.6 to build a polynomial degree circuit generating L. Note first that Lemma 6.6 shows how to build a log-depth circuit to simulate an AuxPDA that has a small height bound. If the AuxPDA is *not* assumed to have a small height bound, then the circuit constructed is still correct, but it will no longer have small depth. (However, small depth is not required for this lemma.) Another problem that must be addressed is that our AuxPDA is *accepting* a language, and we are supposed to build a circuit that *generates* the language, which is different from what is required in Lemma 6.6. The only change that is required to address this problem is to change the constant-depth circuit constructed in the case $q - p = 2$ in the proof of Lemma 6.6; instead construct a constant-depth circuit that evaluates to the set of all strings x such that (P, Q) is a realizable pair because the machine can start in surface configuration P and reach configuration Q consuming input x. Details are left to the reader.

For the converse, let L be generated by $\{C_n\}$. Then our 1-LOGCFL machine will first guess the length of the input, put the output gate of C_n on its worktape, and begin an exploration, much as in the proof of Lemma 7.1. To explore a $+$ gate, nondeterministically guess a child to explore. To explore a $\times$ gate, put the right child on the stack and explore the left child. When a leaf is encountered, match it against the next input symbol, and then explore the node stored on top of the stack. If the degree of the circuit is small, then the runtime will be polynomial.

If the degree of the circuit is *not* small, then the same routine will work, showing that anything that can be generated by polynomial-size circuits can be accepted by a 1-AuxPDA.

To complete the proof, we need only show that every set accepted by a 1-AuxPDA can be generated by a poly-size circuit. The crucial observation here is that if a 1-AuxPDA doesn't run in polynomial time, then it makes many moves without moving its input head. Thus we can build a poly-size circuit that evaluates to λ if (P, Q) is a realizable pair via a computation that consumes no input, and evaluates to $\emptyset$ otherwise. The rest of the construction is similar to the construction sketched above for 1-LOGCFL. (Related observations concerning AuxPDAs that have limits on the number of times they move their input heads are made in [Al89, ABP92]; providing a full proof is routine, using ideas presented there.) ▮

Corollary 7.4 *A language L is accepted by an unambiguous 1-LOGCFL machine iff there is a polynomial size, polynomial degree circuit generating L unambiguously. A language L is be accepted by an unambiguous 1-AuxPDA machine iff there is a polynomial size circuit generating L unambiguously.*

7.2 Lower Bounds for Circuits with Restricted Skewness Patterns

Given a semi-unbounded generator circuit C_n and a string w of length n, consider the problem of determining whether the coefficient c_w in the polynomial $P(C_n)$ is non-zero. If the coefficient is non-zero, then the circuit must have a proof tree for this monomial w.

For the purpose of analyzing how the tree constructs (or parses) the monomial, the union gates can be ignored; the parse structure is determined by the subtrees rooted at concat gates. If the circuit is left-skew, then all proof trees look identical, since each concat gate has its left subtree anchored to a leaf node. The same is true for right-skew circuits. Now consider classes of skew circuits which satisfy the following constraint: "All proof trees in the circuit are identical." We call this class the class of *clone-skew circuits.*

In any proof tree of a skew circuit, all concat gates lie on a single root-to-leaf path, with the leaf inputs of the concat gates hanging off this path on either side. Label a concat gate L (R) if it is left-skew (right-skew). (The L's and R's are not to be confused with the LR labeling in Section 6.) Now the sequence of the labels of the concat gates from root to leaf gives the parse structure of the proof tree. These sequences are the same for all proof trees in a clone-skew circuit. Let σ denote this sequence; we can then refer to σ-clone-skew circuits.

Consider a σ-skew tree computing a monomial m, and let N be some node on the tree computing the partial monomial v. Then we can write $m = l{\cdot}v{\cdot}r$, where l (r) is the product of the symbols seen at left-skew (right-skew, respectively) gates on the path from the root to N. Note that as we travel from root to leaf, all symbols in l as well as in r are seen before we reach the node N. Let u be the sequence of symbols seen on the root-to-N path, written in the order in which they are seen. Now, the sequence σ tells us how to obtain l and r from u. Namely, index symbols of u by elements from the sequence σ (a prefix of suitable length). l is the product, in left-to-right order, of the symbols of u that get indexed L. And r is the product, in reverse order, of the symbols of u indexed R.

Given the symbols in the order in which a root-to-leaf traversal sequence scans them, and given sequence σ, we can construct the monomial. We can do the same if part of the monomial is given directly. Given u and v as above, and sequence σ, we can splice u and v together to correctly construct m. Let us denote this by $\sigma(u, v) = m$.

The results in this section extend Nisan's results on non-commutative computation [Ni91], and thus it is necessary to express his framework using the circuit models we have used thus far in this paper. Nisan's work is motivated by the desire to understand the extent to which efficient computation of the permanent and determinant rely on commutativity. To explore this, he considers the non-commutative ring formed by adding non-commuting indeterminates $x_1, x_2, \ldots$ to the reals. (The indeterminates commute with the reals, but not with each other.) In this setting, then, the $n \times n$ permanent is defined to be the polynomial on n^2 indeterminates $x_{1,1}, \ldots, x_{n,n}$ given by

$$\sum_{\sigma \in S_n} \prod_i x_{i,\sigma(i)}$$

and the $n \times n$ determinant is defined similarly, with the sign of σ multiplied in.

Thus, just as in the case of (union,concat) circuits, Nisan's focus is on circuit families $\{C_n\}$ where C_n is computing a *constant* function (i.e., there are no input variables). It is perhaps counterintuitive to think of the $n \times n$ permanent or determinant as being a constant function; however in Nisan's setting a circuit C_n computing the $n \times n$ determinant is a circuit with no input variables, but having leaves labeled by indeterminates (which are semiring elements) and having the $n \times n$ determinant as its formal polynomial. (He does allow reals to appear as constants labeling leaves in the circuit; in his setting the formal polynomial $P(C_n) = \sum_w c_w \prod(w)$ where w is a finite sequence of indeterminates, and c_w is obtained by grouping together all terms having the same sequence w and adding those coefficients. (This is a departure from earlier sections, where the formal polynomial of a circuit would have constants embedded in the middle of terms.))

For language L, define functions $f_L(n)$; $f_L(n) = \sum_{w \in \Sigma^n} \chi_L(w)w$, where $\chi_L(w)$ evaluates to one of the semiring constants 0 or 1. (We drop the subscript n where it is obvious). Thus all coefficients in f_L are 0 or 1 ($\emptyset$ or $\{e\}$).

Now we follow notation and definitions from [Ni91].

Definition 7.5 ([Ni91]) *An Algebraic Branching Program (ABP) is a directed acyclic graph with one source and one sink. The vertices of the graph are partitioned into "levels" numbered from 0 to n, where edges may only go from level i to level $i+1$. n is the degree of the ABP. The source is the only vertex at level 0, and the sink is the only vertex at level n. Each edge is labeled with a homogeneous linear function of the form $\sum_i c_i x_i$. The size of the ABP is the number of vertices.*

An ABP computes a homogeneous polynomial of degree n; the function is the sum over all source-to-sink paths of the product of the linear functions labeling the edges on the path. Here each x_i is an element of Σ, and each coefficient c_i is one of the semiring constants 0 or 1.

ABPs are essentially leveled left-skew circuits. To generalize this to clone-skew circuits, we generalize the definition of ABPs as follows. Each edge (e) is labeled by a linear function f_e as well as a tag $t_e \in \{L, R\}$. The tag indicates whether f_e should pre-multiply (tag L) or post-multiply (tag R) the partial function already constructed.

Formally, the generalized ABP (GABP) computes a function which is the sum, over all source-to-sink paths ρ, of the function computed on the path ρ. The function computed by a path is defined as follows: Let σ_ρ denote the sequence of tags on the edges in path ρ, and let s_ρ denote the sequence of labels on the edges. Then the function computed by ρ is $\sigma_\rho(s_\rho, e)$, where e is the empty sequence. In other words, the labels on the edges of the path are rearranged according to the sequence σ_ρ and then multiplied.

It is easy to see that a σ-clone-skew circuit corresponds to a GABP where for any source-to-sink path $\rho = e_1 \dots e_k$, $t(e_1) \dots t(e_k) = \sigma$.

We can define, analogous to Nisan's matrices $M_k(f)$, matrices of the form $M_{k,\sigma}(f)$. Matrix $M_{k,\sigma}(f)$ has rows indexed by monomials of degree k, and columns indexed by monomials of degree $n - k$. The entry at $\langle u, v\rangle$ is the coefficient of the monomial $\sigma(u, v)$ in f.

The following lemma is an easy extension of Theorem 1 from [Ni91].

Lemma 7.6 *The size of the smallest σ-clone-skew circuit generating L unambiguously is exactly $\Sigma_{k=0}^{d} rank(M_{k,\sigma}(f_L))$.*

It is worthwhile observing that we can strengthen Lemma 7.6 slightly by deleting the word "unambiguously". The proof amounts to slightly modifying the notion of what it means for an ABP to compute a function.

An ABP is said to compute the function f which is the sum (in the appropriate semiring) over all source-to-sink paths, of the product of the labels of edges on the path. Here we are interested only in (union,concat) circuits, and the formal polynomial counts the number of explorations. So, for the ABP too, it makes sense to instead associate a function f which is defined as a formal polynomial: for word w, the coefficient of w in f is the *number* of source-to-sink paths in the ABP that evaluate to w. Of course, the same definition can be used for GABPs. If we use the new definition, then we have an equivalence between formal polynomials of σ-skew (union,concat) circuits and σ-GABPs. Now use Nisan's proof, with appropriate $L_{k,\sigma}$ and R_k matrices.

Lemma 7.7 *The size of the smallest σ-clone-skew circuit generating L with the formal polynomial f is exactly*

$$\Sigma_{k=0}^{d} rank(M_{k,\sigma}(f))$$

.

Theorem 7.8 *The class of languages generated unambiguously by left-skew circuits is strictly contained in the class of languages generated unambiguously by clone-skew circuits.*

Proof: Consider the language $L_1 = \left\{ww^R \mid w \in \{0,1\}^*\right\}$. In [Ni91, Theorem 4.2] it is shown that any left-skew circuit generating this language unambiguously must have exponential size. However there is a linear size circuit generating this language, and it is easily seen that this circuit is clone-skew and generates the language unambiguously. (σ = LRLRLR... for proof trees in this circuit.) ▌

The skewness pattern required for generating this language follows from the fact that the language is a linear context-free language. If we consider context-free languages which are not linear, then it is reasonable to expect that skew circuits for the languages must be large. We give one such instance; the language $L_2 = \{x \in \{0,1\}^* \mid x \text{ is not of the form } ww\}$ has no sub-exponential size unambiguous clone-skew generators. However, the lower bound heavily relies on unambiguity; the language does have polynomial size left-skew generators.

Lemma 7.9 *Any clone-skew circuit generating L_2 unambiguously must have exponential size.*

Sketch of proof: Let f be the function corresponding to L_2 for words of length n. First, we illustrate the proof idea by considering the case when the proof tree must have concat gates on any root-to-leaf path labeled σ = LRLRLR..., as described in the previous proof. Now consider the matrix $M_{n/2,\sigma}(f)$. This matrix has exactly one zero in each row and one zero in each column; it thus has rank $2^{n/2}$. The lower bound follows from Lemma 7.6; no σ-clone-skew circuit of sub-exponential size can generate L_2. This argument can be extended to σ-clone-skew circuits, for any σ. Thus no clone-skew circuit of sub-exponential size can generate L_2. ▮

Proposition 7.10 *L_2 can be generated by a polynomial-size left-skew circuit.*

Proof: This language can be accepted by a 1-NLOG machine which guesses an integer i and then verifies that the input bits at positions 1 and $i + n/2$ are distinct. The proposition now follows from Lemma 7.1. ▮

Thus unambiguity is a proper restriction:

Theorem 7.11 *The class of languages generated unambiguously by clone-skew circuits is strictly contained in the class of languages generated by clone-skew circuits.*

However, it is not true in general that non-linear context-free languages require large skew circuits. For instance, let L_3 be the Dyck language of balanced parentheses, where $\mathbf{0}$ ($\mathbf{1}$) is interpreted as an opening (closing) parenthesis. This set is easily seen to be in 1-DLOG, and thus has polynomial-size unambiguous left-skew circuits.

Lemma 2 of [Ni91] indicates that computing the permanent or determinant in a non-commutative setting via left-skew circuits requires at least exponential size. We show that this lower bound holds even if arbitrary skew circuits are allowed.

We need the following definition: A function f is said to be weakly equivalent to a function g if, for each monomial in g with a non-zero coefficient, there is a monomial in f with the same variables (though not necessarily in the same order) with a non-zero coefficient, and *vice versa.*

Theorem 7.12 *Any skew circuit family computing the permanent or determinant in a non-commutative setting must have at least exponential size.*

Proof: Consider any skew circuit computing the permanent. By treating the gates as commutative gates, the circuit can trivially be converted into a left-skew circuit of the same size. The function computed by such a circuit is clearly weakly equivalent to the permanent. By Theorem 2 of [Ni91], any function weakly equivalent to the permanent has exponential size. ▮

Acknowledgments

The first author thanks Shiyu Zhou and David Zuckerman for helpful discussions.

References

[Ad78] L. Adleman, *Two theorems on random polynomial time* Proc. 19th FOCS (1978) pp. 75–83.

[Al89] E. Allender, *P-uniform circuit complexity*, J. ACM 36 (1989) 912–928.

[ABP92] E. Allender, D. Bruschi, and G. Pighizzini, *The complexity of computing maximal word functions*, DIMACS tech report 92-15.

[AG94] E. Allender and V. Gore, *A uniform circuit lower bound for the permanent*, SIAM J. Comput. 23 (1994) 1026–1049.

[AH93] E. Allender and U. Hertrampf, *Depth reduction for circuits of unbounded fan-in*, Information and Computation 112 (1994) 217–238.

[AJ93a] E. Allender and J. Jiao, *Depth reduction for non-commutative arithmetic circuits*, in Proc. 25th Annual Symposium on Theory of Computing, 1993, pp. 515–522.

[AJ92] C. Álvarez and B. Jenner, *A note on log space optimization*, report, L.S.I., Universitat Politècnica Catalunya, Barcelona, 1992.

[AJ93] C. Álvarez and B. Jenner, *A very hard log-space counting class*, Theoretical Computer Science 107 (1993) 3–30.

[AO94] E. Allender and M. Ogihara, *Relationships among PL, #L, and the determinant*, Proc. 9th IEEE Structure in Complexity Theory Conference, 1994, pp. 267–278.

[BCH86] P. W. Beame, S. A. Cook, and H. J. Hoover, *Log depth circuits for division and related problems*, SIAM J. Comput. 15 (1986) 994–1003.

[BT91] R. Beigel and J. Tarui, *On ACC*, Proc. 32nd FOCS (1991) 783–792.

[BFS92] J. Boyar, G. Frandsen, and G. Sturtivant, *An arithmetical model of computation equivalent to threshold circuits*, Theoretical Computer Science 93 (1992) 303–319.

[CRS93] S. Chari, P. Rohatgi, and A. Srinivasan, *Randomness-optimal unique element isolation, with applications to perfect matching and related problems*, Proc. 25th STOC (1993) 458–467.

[Co71] S. Cook, *Characterization of pushdown machines in terms of time-bounded computers*, J. ACM 18 (1971) 4–18.

[Da91] C. Damm, *$DET = L^{\#L}$?*, Informatik-Preprint 8, Fachbereich Informatik der Humboldt-Universität zu Berlin, 1991.

[GG79] O. Gabber and Z. Galil, *Explicit constructions of linear size superconcentrators*, Proc. 20th FOCS (1979) 364–370.

[Gu95] S. Gupta, *Isolating an odd number of elements, linearly restrictable sets, and applications in complexity theory*, manuscript, Virginia Technical University.

[IL95] Neil Immerman and Susan Landau, *The Complexity of Iterated Multiplication,* Information and Computation 116, (1995), 103-116.

[IZ89] R. Impagliazzo and D. Zuckerman, *How to recycle random bits*, Proc. 30th FOCS (1989) 248–253.

[Je93] B. Jenner, personal communication.

[JS82] M. Jerrum and M. Snir, *Some exact complexity results for straight-line computations over semirings*, J. ACM 29 (1982) 874–897.

[KVVY93] R. Kannan, H. Venkateswaran, V. Vinay, and A. Yao, *A circuit-based proof of Toda's theorem*, Information and Computation 104 (1993) 271-276.

[Ko94] J. Köbler, *Extension of Toda's theorem to middle bit classes*, Proc. Workshop on Algebraic Methods in Complexity Theory (AMCoT), December 11-13, 1994, Institute of Mathematical Sciences, Madras (Technical Report IMSc-94/51).

[Ko90] S. R. Kosaraju, *On the parallel evaluation of classes of circuits*, Proc. 10th FST&TCS, Lecture Notes in Computer Science 472 (1990) 232–237.

[Kr88] M. Krentel, *The complexity of optimization problems*, JCSS 36 (1988) 490–509.

[MRK88] G. Miller, V. Ramachandran, and E. Kaltofen, *Efficient parallel evaluation of straight-line code and arithmetic circuits*, SIAM J. Comput. 17 (1988) 687–695.

[MT87] G. Miller and S.-H. Teng, *Dynamic parallel complexity of computational circuits*, Proc. 19th STOC (1987) 478–489.

[NRS94] A. Naik, K. Regan, and D. Sivakumar, *Quasilinear time complexity theory*, Proc. 11th STACS, Lecture Notes in Computer Science 778 (1994) 289–300.

[Ni91] N. Nisan, *Lower bounds for non-commutative computation*, Proc. 23rd STOC (1991) 410–418.

[RT92] J. Reif and S. Tate, *On threshold circuits and polynomial computation*, SIAM J. Comput. 21 (1992) 896–908.

[Ru81] W. Ruzzo, *On Uniform Circuit Complexity*, J. Comput. and System Sci. 21 (1981) 365–383.

[Su78] I. H. Sudborough, *On the tape complexity of deterministic context-free languages*, J. ACM 25 (1978) 405–414.

[To91] S. Toda, *PP is as hard as the polynomial-time hierarchy* SIAM J. Comput. 20 (1991) 865–877.

[To91a] S. Toda, *Counting problems computationally equivalent to the determinant*, manuscript.

[To92] S. Toda, *Classes of arithmetic circuits capturing the complexity of computing the determinant*, IEICE Trans. Inf. and Syst., vol. E75-D (1992) 116–124.

[Va79] L. Valiant, *Completeness classes in algebra*, Proc. 11th STOC (1979) 249–261.

[Va92] L. Valiant, *Why is Boolean complexity theory difficult?* in *Boolean Function Complexity*, edited by M. S. Paterson, London Mathematical Society Lecture Notes Series 169, Cambridge University Press, 1992.

[VSBR83] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, *Fast parallel computation of polynomials using few processors*, SIAM J. Comput. 12 (1983) 641–644.

[VV86] L. Valiant and V. Vazirani, V, *NP is as easy as detecting unique solutions* Theoretical Computer Science 47 (1986) 85–93.

[Ve91] H. Venkateswaran, *Properties that characterize LOGCFL*, JCSS 42 (1991) 380–404.

[Ve92] H. Venkateswaran, *Circuit definitions of nondeterministic complexity classes*, SIAM J. Comput. 21 (1992) 655–670.

[VT89] H. Venkateswaran and M. Tompa, *A new pebble game that characterizes parallel complexity classes*, SIAM J. Comput. 18 (1989) 533–549.

[Vi91] V. Vinay, *Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits*, Proc. 6th IEEE Structure in Complexity Theory Conference (1991) 270–284.

[Vi91a] V. Vinay, *Semi-unboundedness and complexity classes*, doctoral dissertation, Indian Institute of Science, Bangalore.