Monotone Complexity

Michelangelo Grigni[*] *Michael Sipser*[†]

Abstract

We give a general complexity classification scheme for monotone computation, including monotone space-bounded and Turing machine models not previously considered. We propose monotone complexity classes including mAC^i, mNC^i, $mLOGCFL$, $mBWBP$, mL, mNL, mP, $mBPP$ and mNP. We define a simple notion of monotone reducibility and exhibit complete problems. This provides a framework for stating existing results and asking new questions.

We show that mNL (monotone nondeterministic log-space) is not closed under complementation, in contrast to Immerman's and Szelepcsényi's nonmonotone result [Imm88, Sze87] that $NL = co\text{-}NL$; this is a simple extension of the monotone circuit depth lower bound of Karchmer and Wigderson [KW90] for st-connectivity.

We also consider $mBWBP$ (monotone bounded width branching programs) and study the question of whether $mBWBP$ is properly contained in mNC^1, motivated by Barrington's result [Bar89] that $BWBP = NC^1$. Although we cannot answer this question, we show two preliminary results: every monotone branching program for majority has size $\Omega(n^2)$ with no width restriction, and no monotone analogue of Barrington's gadget exists.

1. Introduction

A computation is *monotone* if it does not use the negation operation. Monotone circuits and formulas have been studied as restricted models of computation with the goal of developing techniques for the general problem of proving lower bounds.

[*]Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1Z2. Work completed while this author was at MIT.

[†]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. Research supported by NSF grant 8912586-CCR and DARPA contract N00014-89-J-1988.

In this paper we seek to unify the theory of monotone complexity along the lines of Babai, Frankl, and Simon [BFS86] who gave a framework for communication complexity theory. We propose a collection of monotone complexity models paralleling the familiar nonmonotone models. This provides a rich classification system for monotone functions including most monotone circuit classes previously considered, as well as monotone space-bounded complexity classes which have previously received little attention. This classification gives a language for discussing existing results and for suggesting new problems.

To illustrate our objective, let us consider two of the main results on monotone complexity: 1) Razborov's theorem showing that the clique function is not computable by polynomial size monotone circuits and 2) Karchmer and Wigderson's theorem showing that the *st*-connectivity function cannot be computed by log-depth monotone circuits. These may be viewed as saying that, with respect to monotone computation, $P \neq NP$, and that $NC^1 \neq NL$.

We ask which complexity class containments carry over to their monotone analogues. Many of the obvious containments carry through, e.g. the simulation of Turing machines by circuits. In particular we consider two recent surprising containment results in space-bounded complexity:

1. Immerman [Imm88] and Szelepcsényi [Sze87] showed that NL is closed under complementation, and

2. Barrington [Bar89] showed that $BWBP$ (bounded width branching programs) contains all of NC^1.

These results are interesting because the simulation techniques do not seem to carry over to the corresponding monotone models.

We show that the first result does not hold in a monotone world, i.e. the monotone complexity class corresponding to NL is not closed under complementation, so the inductive counting technique used by these authors cannot be replaced by a monotone simulation; the proof of this is an extension of the proof of Karchmer and Wigderson [KW90].

We have not resolved the second result, i.e. whether monotone bounded width branching programs may simulate monotone formulas. Nevertheless we present evidence that Barrington's simulation technique does not carry over to the monotone world.

2. Monotone Complexity

In this section we define monotone computational models and complexity classes analogous to the usual notions of circuits, branching programs, and Turing machines. We generally will not distinguish uniform and nonuniform models unless stated explicitly. Our notation will be for a typical complexity class C, to define the monotone analogue mC. This is in contrast with the collection of functions in C that happen to be monotone, a class we refer to as $C \cap \mathbf{mono}$. We show that the standard simulations may be carried out in a monotone fashion so that most familiar class containments still hold. We consider the notion of monotone reducibility and complete problems for these classes.

2.1. Monotone Functions

Before proceeding further we ask: What is a monotone boolean function? This is generally defined to be one where changing any input from 0 to 1 can only change the function value from 0 to 1 and not from 1 to 0. But with this definition there is no way for a monotone class to be closed under complementation, since the complements of monotone functions are not themselves monotone. This covers over an important issue since there is a natural way to redefine what we mean by monotone in such a way that mP is closed under complement yet other monotone classes remain not closed. This gives additional structural information about the monotone classes. Our definition is as follows. Call a function with the above property *positive monotone*, and the complement of a positive monotone function *negative monotone*. A function is *monotone* if it is either positive or negative monotone. We note that this usage of the terminology is more consistent with its counterpart in real analysis.

2.2. Monotone Computational Models

Monotone circuits. The standard definition of monotone circuits is to allow AND and OR gates, or perhaps some larger basis of positive monotone gates, but no negations. In order to compute negative monotone functions as well we consider a circuit to be monotone if the only negations are on the input variables, and either all of the input variables appear with negations or none of them appear with negations.

Monotone formulas. As above where all gates have outdegree 1. There has been much recent work (e.g. [KW90, RW89, RW90]) in monotone formula depth with no restriction on size, due to an exact characterization of depth by the communication complexity of certain two-party problems.

Monotone nondeterministic circuits. Nondeterministic circuits are defined to be ordinary circuits whose inputs are divided into two parts: the nondeterministic inputs and the standard inputs. A nondeterministic circuit accepts a setting of the standard inputs if there is some setting of the nondeterministic inputs which causes evaluation to 1. Each nondeterministic input bit may be input only once to the circuit; this is important in space-bounded circuits. A *monotone nondeterministic circuit* is one where the monotonicity requirement applies only to the standard inputs; that is, we may use a negation gate in the circuit as long as it depends only on nondeterministic input bits. Note that if the nondeterministic inputs were treated monotonely as well, then their only interesting setting would be all 1's or all 0's.

Monotone randomized circuits. Treat the random input bits like the nondeterministic input bits above. That is, negations are allowed which depend only on the random bits, and each random bit is input only once. In the nonuniform case, random models are generally equivalent to the deterministic model, so a proper treatment of random monotone complexity requires a uniform monotone model such as the monotone nondeterministic Turing machine model below.

Monotone nondeterministic branching programs. A nondeterministic branching program is an acyclic network of nodes and directed edges with distinguished start and finish nodes, where each edge is either labeled with the constant 1 or with a literal (a variable or a negated variable). This defines a boolean connectivity function in a natural way: accept iff there exists a path from the start to the finish, such that all literals appearing on the path are true. *Monotone nondeterministic branching programs* are those where either none or all of the variables appear negated.

A branching program or circuit is *leveled* if the nodes are arranged in a sequence of levels with internal wires allowed only from one level to the next (inputs may be used at any level), the start node in the first level and the accept node in the last. The maximum size of a level is the *width* of the branching program or circuit.

Monotone nondeterministic Turing machines. The usual nondeterministic Turing machine (NTM) has existential nondeterminism, a read-only input tape, a read-write work tape, and a program specified by a finite set of states and transition rules. A (positive) monotone nondeterministic Turing machine (mNTM) is a NTM with the following restriction on its transition rules: whenever the machine may make a transition with a 0 on its input tape, it may make that same transition with a 1 on its input tape. Note there is no such restriction on the bits of its work tape. A negative mNTM may be defined similarly: for every 1-transition there is also a 0-transition. This mNTM model may include randomization as well, with no restriction on the use of random bits.

The above description applies to existential nondeterminism. We may generalize this to universal nondeterminism and alternation as follows. We define a monotone alternating Turing machine as the usual alternating Turing machine, but with restrictions on how it may read its input bits. An input bit x may be referenced only by either the existential construction $\exists z : (z \leq x) \wedge (\ldots)$, or by the universal construction $\forall z : (z > x) \vee (\ldots)$, where $(\ldots)$ stands for the rest of the computation. Note we do not have a uniform monotone analogue for deterministic Turing machines; however with monotone alternation we may define the classes $mALOGTIME$ and $mALOGSPACE$, which serve as uniform versions of mNC^1 and mP, respectively.

2.3. Monotone Complexity Classes

Using these models we define uniform and nonuniform monotone complexity classes.

The circuit classes mAC^i, mNC^i, and mP are standard. $mLOGCFL$ may be defined as monotone polynomial size log-depth circuits with unbounded OR gates and bounded AND gates. More generally, if the depth is $O(\lg^i n)$, we call this $mSAC^i$. mL may be defined as the class of monotone log-width polynomial size leveled circuits. From these circuit classes we may straightforwardly construct nondeterministic variants such as mNL and mNP; we note that these nondeterministic classes also have uniform definitions in terms of monotone nondeterministic Turing machines.

Polynomial size nondeterministic branching programs define the nonuniform analogue to NL; mNL is the corresponding monotone class. We remark that this is equivalent to the nondeterministic version of the circuit class mL above, where each nondeterministic bit may only be input once. Showing that the circuit model contains the branching program model is not entirely trivial;

it involves representing each node internally by a binary address and the negation of that address.

Bounded width branching programs are leveled nondeterministic branching programs such that each level has size bounded by some constant. Polynomial size bounded width branching programs define the (nonuniform) class $BWBP$, and $mBWBP$ is the corresponding monotone class.

2.4. Monotone Simulations

As mentioned before, many of the familiar simulations and containments from general complexity carry over to the monotone world. For example we may define mNC^1 either in terms of polynomial size monotone formulas or in terms of bounded fan-in log-depth monotone circuits, since it is easy to show that these are equivalent in power. Similarly mNC^1 contains $mBWBP$, since the standard simulation preserves monotonicity.

Other simulations need a little more argument. For example we have three potential models for mNL: branching program, circuit, and Turing machine. It is reassuring to argue that these models are still equivalent up to uniformity. We make these arguments in the next three paragraphs for positive monotone functions; the negative cases are similar.

Turing machine to circuit. Given a positive monotone nondeterministic Turing machine using an $O(\lg n)$-size work tape, for a given n we want to simulate it with a monotone $O(\lg n)$-width circuit with read-once nondeterministic inputs. We follow the usual tableau construction, with the following modification: where an input bit x_i is input, we guess a nondeterministic bit z, and test if $z \leq x_i$. We then use z in the place of x_i as the tableau input. Thus the tableau may contain arbitrary negations, since they will depend only on nondeterministic bits. Also if x_i is input at many different points, we may use a different nondeterministic bit z at each point, since the monotone Turing machine is guaranteed to compute the right function even if some 1-inputs are sometimes read as 0-inputs. Finally AND together the results of all the $z \leq x_i$ comparisons with the final output of the tableau.

Circuit to branching program. Given a leveled width-w ($w = O(\lg n)$ for mNL) monotone nondeterministic circuit with AND, OR, and 'input' gates, we wish to simulate it with a monotone polynomial size branching program. We follow a level-by-level reduction, with each vertex in the given level of the branching program corresponding to one of the 2^w possible states of the circuit

level, where 'state' refers to the w-vector of truth values output by the gates in that circuit level. We design the program so that a vertex is reachable in the branching program iff the corresponding state is *less than or equal* to a state achievable in that level of the circuit. With this modification to the usual requirement, the argument proceeds straightforwardly. Logic gates (AND and OR) and nondeterministic input gates are modeled by constant wires in the branching program. Real input gates (which in the nonmonotone case would be modeled by branching program wires labeled x_i and $\neg x_i$) may in the monotone case be modeled by edges labeled x_i and 1. We omit an argument for correctness.

Branching program to Turing machine. Essentially we need to show a monotone logspace nondeterministic Turing machine may still compute the directed st-connectivity function. Given the input as an adjacency matrix, the usual nonmonotone algorithm in fact works on a monotone Turing machine as well.

Similar simulation arguments show that our two potential models for mNP are equivalent up to uniformity. The following containments are now immediate:

Theorem 2.1 $\mathrm{mAC}^0 \subseteq \mathrm{mBWBP} \subseteq \mathrm{mNC}^1 \subseteq \mathrm{mL} \subseteq \mathrm{mNL} \subseteq \mathrm{mLOGCFL} \subseteq \mathrm{mAC}^1 \subseteq \mathrm{mNC}^i \subseteq \mathrm{mAC}^i \subseteq \mathrm{mNC}^{i+1} \subseteq \mathrm{mP} \subseteq \mathrm{mNP}$.

2.5. Trivial Monotone Classes

We say that a monotone class mC is *trivial* if it satisfies $mC = C \cap \mathbf{mono}$.

Given any kind of (existential) nondeterministic machine, the corresponding positive monotone machine is one where for each 0-transition—a transition allowed when some input bit $x_i = 0$—the corresponding 1-transition is also allowed. This generalizes our definition for mNTMs, similar definitions apply for negative monotone machines. We say a nondeterministic machine is read-once if it reads each input bit at most once on any computation path; the bits do not have to be read in the same order on each path.

Theorem 2.2 *Given complexity class* C *defined in terms of a read-once nondeterministic machine, the corresponding monotone class* mC *is trivial.*

Proof : Given a (nonmonotone) nondeterministic machine accepting a positive monotone language L (i.e. given $x \in L$, if we replace any 0 in x by a 1 then the resulting string is still in L), we need to show L is also accepted

by a monotone machine. We convert the given automaton into a monotone automaton by simply allowing a 1-transition whenever the original machine allowed a 0-transition.

Suppose the new monotone machine accepts string y. Then the computation path accepting y may have used some of the new 1-transitions, but if we replace each such 1 in y by a 0, we get a string x that was accepted by the original automaton, hence $x \in L$. Since L is monotone and $x \in L$, we have $y \in L$; thus the monotone machine accepts exactly L. □

It follows that *mREG* (via nondeterministic finite state automata), *mCFL* (via nondeterministic push-down automata), and *mNP* (since an *mNP* machine may copy its entire input to its internal work tape) are all trivial classes.

We note that a similar argument applies in some circuit models, such as the circuit version of *mNP* or monotone read-once branching programs. Finally we note the triviality of $AC[2]$ circuits (polynomial size CNF or DNF formulas).

Theorem 2.3 mAC[2] *is trivial.*

Proof : For an AND of ORs computing a positive monotone function, it suffices to replace each negated input by a 0. Similarly for an OR of ANDs, it suffices to replace each negated input by a 1. □

This method fails for depth three; we do not even know if $AC[3] \cap \mathbf{mono} \subset mP$.

2.6. Rephrasing Known Results

We may rephrase some known results on monotone complexity as follows:

Theorem 2.4 ([Raz85a]) $\mathrm{mP} \neq \mathrm{mNP}$.

Theorem 2.5 ([Raz85b]) $\mathrm{mP} \neq \mathrm{P} \cap \mathbf{mono}$.

Theorem 2.6 ([AG87]) $\mathrm{mAC}^0 \neq \mathrm{AC}^0 \cap \mathbf{mono}$.

Theorem 2.7 ([KW90]) $\mathrm{mNC}^1 \neq \mathrm{mNL}$.

Theorem 2.8 ([Yao89]) $\mathrm{mTC}^0 \neq \mathrm{mNC}^1$.

Theorem 2.9 ([RW90]) $\mathrm{mNC}^1 \cap \mathbf{mono} \not\subset \mathrm{mNC}$.

2.7. Monotone Separations

As listed above, there are many separations of monotone classes with no corresponding separation of the corresponding nonmonotone classes; so far no monotone separation methods have successfully carried over. It seems then that nonmonotone separations are at least as hard as monotone separations. On the other hand showing a containment between classes is a stronger result in the monotone world than in general, since we need to check that monotonicity is preserved. Here we argue that these intuitions are correct, at least for a wide range of classes.

For reductions we use polynomially bounded boolean projections, from Skyum and Valiant [SV85]. Given two boolean function families $\{f_n\}$ and $\{g_n\}$, we say that f is a projection of g if for each n there exists $p(n)$ and σ such that $f_n(x) = g_{p(n)}(\sigma(x))$. Here $p(n)$ is bounded by some polynomial and $\sigma(x)$ is a substitution; i.e. every argument of g is either a constant, an argument of f, or a negated argument of f. If we disallow negative (or positive) argument substitutions, then we say that f is a monotone projection of g.

If C is a complexity class and mC is its monotone analogue, then we are interested in the following question: does there exist a monotone function f complete for C under general projections and also complete for mC under monotone projections? We say such an f is a *monotone complete function* for C. This notion is motivated by the following observation.

Theorem 2.10 *If* C_1 *and* C_2 *are complexity classes with monotone complete functions, then* $\mathrm{mC}_1 \subseteq \mathrm{mC}_2$ *implies* $\mathrm{C}_1 \subseteq \mathrm{C}_2$.

Proof : Let f_1 and f_2 be the corresponding monotone complete functions. Since f_1 is a monotone projection of f_2, it is also a general projection. □

For all classes we have considered, there is such a function. For circuit classes the canonical monotone complete function is the circuit value problem for an appropriate monotone universal circuit. For example a monotone complete problem for NC^1 is the balanced formula of $O(\lg n)$ depth with alternating levels of ANDs and ORs (with distinct variables at the inputs) since this may be restricted to compute any log-depth formula. Simple universal circuit constructions (simulating level-by-level) apply to AC^i, SAC^i, mL, mP, and mNP, while mNC^i may be handled by simulating $O(\lg n)$ levels at a time using the mNC^1 universal formula as a building block.

For nondeterministic branching program classes the appropriate graph connectivity problem (where each directed edge is represented by a variable) is monotone complete. Directed st-connectivity is monotone complete for mNL.

A similar statement (for each bounded width) holds for bounded width digraph connectivity and $mBWBP$.

3. Separating mNL from $co\text{-}mNL$

3.1. Semi-Unbounded Circuits

We first consider the semi-unbounded circuits of Venkateswaran [Ven87]. As above, let SAC^k denote the class of polynomial-size $O(\lg^k n)$-depth circuits with bounded fan-in AND-gates, unbounded fan-in OR-gates, and negations allowed on the inputs (we borrow this notation from [BCD$^+$89], see [Bus87] and [Ven87] for properties of $SAC^1 = LOGCFL$). Without loss of generality, the bounded fan-in gates have fan-in two. Similarly define $co\text{-}SAC^k$ with bounded OR-gates and unbounded AND-gates. Define the monotone variants $mSAC^k$ and $co\text{-}mSAC^k$ similarly but with monotone inputs. Computing the directed connectivity function by repeated squaring of the adjacency matrix, it follows that $NL \subseteq SAC^1$ and $mNL \subseteq mSAC^1$.

A result of Borodin et. al. [BCD$^+$89], closely related to Immerman's result, states that $SAC^k = co\text{-}SAC^k$ for $k \geq 1$. We show that $mSAC^1 \neq co\text{-}mSAC^1$ in the monotone model.

3.2. The Separation

Let **ustconn** denote the undirected st-connectivity function. For a graph of n vertices with two distinguished vertices s and t, **ustconn** has an input variable for each of the $\binom{n}{2}$ edges; clearly **ustconn** $\in mNL$. Karchmer and Wigderson [KW90] showed that any bounded fan-in monotone formula computing **ustconn** requires depth $\Omega(\lg^2 n)$. We extend their proof to show that any $co\text{-}mSAC$ circuit for **ustconn** also requires depth $\Omega(\lg^2 n)$. It then follows that **ustconn** $\notin co\text{-}mNL$, i.e. large fan-in AND-gates do not help to compute the **ustconn** function.

An *l-path* in the graph is a sequence of l vertices (perhaps with repetitions) defining a path from s to t. There are n^l such paths. A *cut graph* is formed by dividing the n vertices into two sets, with s and t in different sets, and putting edges between vertices in the same set. There are 2^{n-2} such cuts. Say that a positive monotone function is an (n, l, α, β) approximator if it outputs 1 for at least fraction α of l-paths, and it outputs 0 for at least fraction β of cut-graphs.

Consider a positive $co\text{-}mSAC$ circuit C which is an (n, l, α, β) approximator. If the top gate of C is an OR-gate, then one of the two children subcircuits

is an $(n, l, \alpha/2, \beta)$ approximator. If the top gate of C is an AND-gate with fan-in at most $d(n)$, then one of the children subcircuits is an $(n, l, \alpha, \beta/d(n))$ approximator. Iterating this process down the circuit for k levels leads to an $(n, l, \alpha/2^k, \beta/d(n)^k)$ approximator.

To show that $\Omega(\lg^2 n)$ depth is necessary, periodically restrict the circuit to get a better approximator to a slightly smaller instance of **ustconn**. This is done with a random restriction $\rho \in \mathcal{R}_k$, which chooses k vertices other than s or t, splits these k into two sets, and then contracts one set with s and the other set with t. We use the following lemma of Boppana [BS90]:

Lemma 3.1 *Let positive monotone function f be an (n, l, α, β) approximator. Suppose $100l/\alpha \leq k \leq n/(100l)$ and $\beta \geq 2^{-n/(100k)}$. Then there is a restriction $\rho \in \mathcal{R}_k$ such that f_ρ is an $(n-k, l/2, \sqrt{\alpha}/2, \beta k/(2n))$ approximator.*

Note α increases greatly while n, l, and β decrease in a controlled way. If f is computed by a positive monotone circuit, then f_ρ is computed by the same circuit with some trivial monotone modification of the inputs. We now prove our theorem:

Theorem 3.2 *Let C be a positive monotone circuit computing* **ustconn** *with fan-in two OR-gates and fan-in $d(n)$ AND-gates, where $d(n) = 2^{\sqrt{n}/(\lg^2 n)-15}$. Then C has depth greater than $(\lg^2 n)/100$.*

Proof : Suppose C has depth $(\lg^2 n)/100$. Divide C into $(\lg n)/10$ blocks of $(\lg n)/10$ levels each. Let $l = n^{1/4}$ and let $k = \sqrt{n}$. Note C is an $(n, l, 1, 1)$ approximator.

Repeatedly explore down one block of C, and then apply the lemma to find a restricted subcircuit with boosted α. At no point during the exploration does α fall below $(n^{-1/5})/4$. When we reach the bottom, we have found a monotone depth-0 circuit that is an $(n', l', \alpha', \beta')$ approximator, where $n' = n - \sqrt{n}(\lg n)/10$, $l' = l/n^{1/10} = n^{1/5}$, $\alpha' = (n^{-1/10})/4$, and $\beta' = d(n)^{-(\lg^2 n)/100}(2\sqrt{n})^{-\lg n/10}$.

A depth-0 circuit is either a constant or a single positive variable. Since $\alpha' > 0$ and $\beta' > 0$, the gate is not a constant. But a single variable cannot accept such a large fraction α' of l-paths. Hence we have a contradiction.

The only new thing to check in this proof is that $\beta' \geq 2^{-n/(100k)}$, which is satisfied by our choice of $d(n)$. □

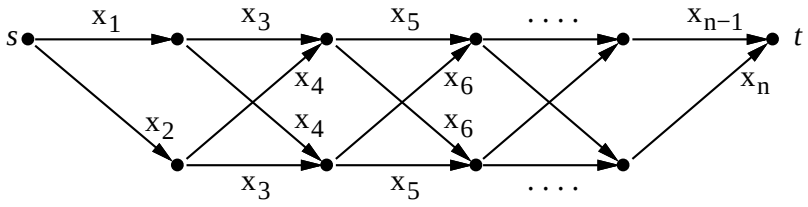

Figure 1. A width-2 $mBWBP$ function outside mAC^0.

Corollary 3.3 co-mSAC *circuits for* **ustconn** *require depth* $\Omega(\lg^2 n)$.

Corollary 3.4 co-mNL *branching programs for* **ustconn** *have size* $n^{\Omega(\lg n)}$.

Straightforward constructions show both the above lower bounds are tight. Suitably adjusting the constants in the above proof shows that even if the AND-gates have fan-in $2^{n^{1-\epsilon}}$, the circuit must still have depth $\Omega(\epsilon^2 \lg^2 n)$. Note the depth lower bound does not depend on circuit size, except as an upper bound on $d(n)$. Also since **ustconn** is in RL, we have (nonuniformly)

Corollary 3.5 $\mathrm{L} \cap \mathbf{mono} \not\subseteq \mathrm{mNL}$.

4. Towards Separating $mBWBP$ from mNC^1

Motivated by Barrington's surprising construction showing $BWBP = NC^1$, a natural question is whether $mBWBP$ (monotone bounded-width branching programs) equals mNC^1.

We remark that $mBWBP$ is equivalent to monotone straight-line boolean programs with a bounded number of registers (if the program may take the AND of two registers, then the branching program width is exponential in the number of registers). Also it is clear that $mBWBP$ is strictly greater than mAC^0, since a width-2 branching program (see figure 1) computes a function which, if it were in mAC^0, would put parity in AC^0.

To show $mBWBP$ is strictly contained in mNC^1, one could consider a complete formula for mNC^1, or try to show that $mBWBP$ is not closed under complementation. We conjecture that the majority function (known to be in mNC^1 [Val84, AKS83]) is not in $mBWBP$. Proving this would require some new technique, since previous methods in monotone complexity do not seem to apply to slice functions. We present two preliminary results.

4.1. A Lower Bound on Size

The monotone branching program model is very close to the (undirected) notion of relay networks studied by Shannon and Moore [MS56] in the context of probabilistic amplification, and identical to "relay-diode bipoles" as studied by Markov [Mar62].

Both papers give versions of the following simple lower bound. Given a positive monotone function f, let the *length* of f be the minimum size of any minterm, and let the *width* of f be the minimum size of any maxterm. Denote these quantities by $l(f)$ and $w(f)$.

Theorem 4.1 (Markov) *Any monotone nondeterministic branching program computing f has at least $l(f) \cdot w(f)$ variable edges.*

Proof : Given such a branching program, for each node u define $d(u)$ as the minimum number of variables that need to be set to 1 to establish a directed path from the start node s to u. In particular $d(t) = l(f)$ for the finish node t.

For $0 \leq i \leq l(f)$ let S_i be the set of nodes u such that $d(u) = i$. If u is connected to v by a constant 1 edge (i.e. not a variable edge) then $d(u) \geq d(v)$, hence there are no constant edges from S_i to S_j for $i < j$. Thus for $0 \leq i < l(f)$, the set of variable edges out of S_i forms an (s,t)-cut of the branching program. Such a cut contains a maxterm, hence at least $w(f)$ distinct variable edges. □

Consider the threshold-k function $T_{n,k}$, which is 1 iff at least k of its n inputs are 1. Then $l(T_{n,k}) = k$ and $w(T_{n,k}) = n-k+1$, so every monotone branching program has at least $k \cdot (n-k+1)$ variable edges. This bound is tight for unbounded width branching programs, for example see figure 2.

Corollary 4.2 *Monotone bounded width branching programs for majority have length $\Omega(n^2)$.*

We know of no better lower bound for the bounded width case. By contrast, the best lower bounds on general bounded width branching programs for explicit functions are superlinear by only logarithmic factors, using much more involved techniques.

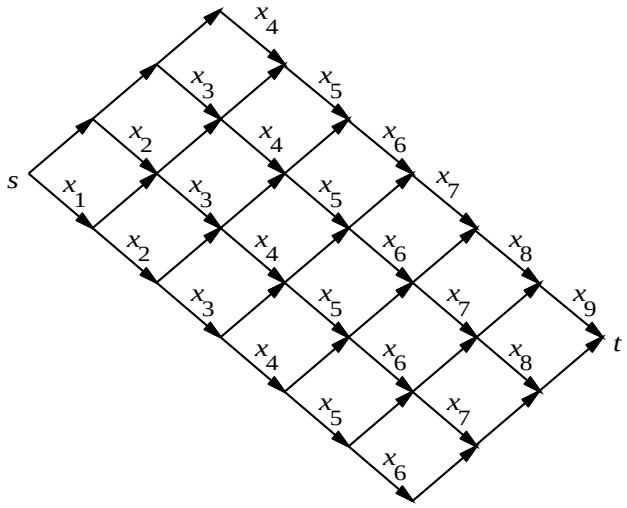

Figure 2. The naive threshold-k branching program has $k \cdot (n-k+1)$ variable edges (here $n = 9$, $k = 6$).

4.2. There is no Monotone Barrington Gadget

Here we show that there is no monotone gadget (of any length or width) like Barrington's gadget composed of four 5-cycles [Bar89]. We need a few definitions to state this precisely.

For constant width w, define a (positive) monotone width-w branching program as in section 2.2.. The branching program computes a 1 iff there exists a directed path from the start node s to the finish node t such that every variable on the path is set to 1.

For levels i and j, $i \leq j$, let $C(i,j)$ be the w by w matrix of boolean functions describing the connectivity from level i to j, i.e. $C(i,j)_{u,v}(x)$ is true iff there exists a path (valid with respect to x) from node u in level i to node v in level j. Given the simple single-step matrices $C(i) = C(i, i+1)$, all the others are defined by boolean matrix multiplication: $C(i,j) \cdot C(j,k) = C(i,k)$.

We consider a constant boolean matrix M (all 0's and 1's) to be a monotone function from P to P, where P is the poset of all 2^w subsets of the w nodes in a level. Given a subset I of $\{1, \ldots, w\}$, M maps I to $M(I)$, the set of all nodes reachable from I via M. More precisely, if χ_I is the characteristic row vector for I, then the characteristic vector for $M(I)$ is $\chi_I \cdot M$. Note that by this definition the applicative order of matrices is from left to right, so that $(AB)(I) = B(A(I))$.) The argument that follows will apply in the

more general setting where P is some finite poset and the M's are monotone functions from P to P.

Given a pair of constant matrices (M_0, M_1) and a boolean variable z, let $M(z)$ denote the variable matrix M_z. If $M_0 = M_1$, we say that $M(z)$ is trivial. If $M_0 \leq M_1$ (i.e. $M_0(I) \leq M_1(I)$ for all $I \in P$) we say that $M(z)$ is monotone.

Barrington's construction [Bar89] (and similar later constructions) using nonsolvable groups may be stated as finding a finite poset P, nontrivial (and nonmonotone) $M(z)$, and constant matrices A_i, B_i $(0 \leq i \leq 4)$ such that the following hold:

$$\begin{aligned} M(x \wedge y) &= A_0 M(x) A_1 M(y) A_2 M(x) A_3 M(y) A_4, \\ M(x \vee y) &= B_0 M(x) B_1 M(y) B_2 M(x) B_3 M(y) B_4. \end{aligned}$$

Given such equations, it is straightforward to show how $BWBP$ may simulate NC^1 with at most quadratic blowup from formula size to the length of the program. Similarly if one can find more equations for a larger basis of functions (e.g. an equation for $M(x \wedge (y \vee z))$) then it is possible to reduce the quadratic blowup to some smaller exponent.

Our result here is that if $M(z)$ is restricted to be monotone, then there is no such pair of equations of any length. For any finite poset P, any $k, l \geq 2$, any sequences of constant matrices A_i, B_j $(0 \leq i \leq k,\ 0 \leq j \leq l)$ and variables $v_i, w_j \in \{x, y\}$ $(1 \leq i \leq k,\ 1 \leq j \leq l)$, there is no nontrivial monotone solution $M(z)$ to the equations

$$M(x \wedge y) = A_0 M(v_1) A_1 M(v_2) A_2 \cdots M(v_k) A_k, \tag{1}$$

$$M(x \vee y) = B_0 M(w_1) B_1 M(w_2) B_2 \cdots M(w_l) B_l. \tag{2}$$

Define the *rank* of a constant matrix M to be the size of the range of M as a function on P: $r(M) = |M(P)|$ where $M(P) = \{M(I) : I \in P\}$. For a two-valued matrix $M(z)$, define $r_0(M(z)) = r(M_0)$, $r_1(M(z)) = r(M_1)$, and $r(M(z)) = |M_0(P) \cup M_1(P)|$.

Theorem 4.3 *If $M(z)$ is a monotone solution to equations 1 and 2, then $M(z)$ is trivial.*

Proof : Without loss of generality the last variable in each equation is y, i.e. $v_k = w_l = y$. Setting $y = 1$ in the first (AND) equation yields $M(x) = F(x) M_1 A_k$, where $F(x)$ is some arbitrary monotone matrix function of x. Thus for all inputs I to $M(x)$ and no matter how we vary x, the range of $M(x)$ is no larger than the range of M_1, i.e. $r(M(z)) \leq r_1(M(z))$. By a

similar argument applied to the second (OR) equation with $y = 0$, we have $r(M(z)) \leq r_0(M(z))$. But since $r(M(z))$ is at least as large as both $r_0(M(z))$ and $r_1(M(z))$, they all represent the same range R of size r, regardless of whether z is 0, 1, or variable.

We consider any $M(v_i)$ in the first equation for $i \geq 2$. Since the whole equation has range r for any setting of x and y, and since the number of different inputs that $M(v_i)$ can receive from $M(v_{i-1})A_i$ is also at most r, $M(v_i)$ must be a one-to-one function from its r possible inputs to r possible outputs R.

More precisely, let D_i be the domain (set of all possible inputs) of $M(v_i)$. We argued that D_i does not depend on the settings of x and y. Now we consider M_0 and M_1 restricted to D_i.

Lemma 4.4 *Given bijections N_0 and N_1 from a set D to finite poset R, such that $N_0(x) \leq N_1(x)$ for all $x \in D$, then $N_0 = N_1$.*

Proof : Necessarily $|D| = |R|$. Define $f = N_1 N_0^{-1}$ mapping R to R, then $y \leq f(y)$ for all $y \in R$. If for some y we have $y < f(y)$, then it follows that $f(y) < f(f(y))$. Iterating yields an infinite chain in R, violating the finiteness of R. Hence f is the identity and $N_0 = N_1$. □

Applying the lemma to $D = D_i$, with N_0 and N_1 the restrictions of M_0 and M_1 to D, we see that $M_0 = M_1$ on D_i, i.e. $M(v_i)$ doesn't really depend on v_i at all!

Since this applies to all $i \geq 2$, equation 1 reduces to $M(x \wedge y) = A_0 M(v_1) C$ where C is a constant matrix. By setting v_1 (say $v_1 = y$) to 1, we get $M(x) = A_0 M_1 C$, so $M(x)$ is trivial. □

The above argument generalizes to "grammars" of matrices: if we have a finite collection of two-valued matrices $M^1(z)$, $M^2(z)$,..., such that each $M^k(x \wedge y)$ and $M^k(x \vee y)$ can be expanded as an equation in the others, then they are all trivial.

Of course a separation of $mBWBP$ from mNC^1 would imply the above theorem, but perhaps this simple rank method (or the metric method of Theorem 4.1) will help resolve the general question.

We remark that in the special case where the underlying poset P is a chain, then the space of all monotone functions on P form an aperiodic monoid, and programs over such a monoid are computable in AC^0 by the work of Barrington and Thérien [BT88]. Similarly if we restrict to a solvable monoid of monotone functions on P, we get a monotone analogue of ACC^0, which may be more tractable than $mBWBP$.

5. Conclusion

We conclude with some open problems in monotone complexity.

1. Find a straightforward uniform monotone analogue for deterministic Turing machines. For example we still have no uniform model for mL.

2. Suggested by Larry Stockmeyer: show there is no polynomial size monotone projection from the directed st-connectivity function to the undirected st-connectivity function.

3. Many of the inclusions in Theorem 2.1 are still not known to be proper (e.g. can we separate higher levels of the mNC hierarchy or separate mNC from mP). Recently [GS91, Gri91] we have succeeded in separating mL from mNC^1, again by the communication complexity method.

4. Is $AC^0 \cap \mathbf{mono} \subseteq mP$? That is, is there any nontrivial monotone upper bound on the complexity of monotone functions which have very low nonmonotone complexity? Since $NC^2 \cap \mathbf{mono} \not\subseteq mP$ (by the matching function [Raz85a]), we cannot hope for much more in mP. The only limit we know here is that $mNP = NP \cap \mathbf{mono}$.

5. Separate mL from $mNL \cap co\text{-}mNL$. A candidate function is directed planar st-connectivity where s and t are on the outer face; it suffices to consider grid graphs. An initial question is whether this function requires $\Omega(\lg^2 n)$ depth.

References

[AG87] Ajtai M., Gurevich Y., *Monotone versus positive. J. ACM, Vol. 34 (1987), pp. 1004–1015.*

[AKS83] Ajtai M., Komlos J., Szemeredi E., *An $O(n \log n)$ Sorting Network. Combinatorica, Vol. 3 (1983), pp. 1–19.*

[AB87] Alon N., Boppana R., *The monotone circuit complexity of boolean functions. Combinatorica, Vol. 7 (1987), pp. 1–22.*

[And85] Andreev A., *On a method for obtaining lower bounds for the complexity of individual monotone functions. Doklady Akademii Nauk SSSR, Vol. 282 (1985), pp. 1033–1037.* English translation in *Soviet Mathematics Doklady, Vol. 31 (1985), pp. 530–534.*

[BFS86] Babai L., Frankl P., Simon J., *Complexity classes in communication complexity theory. Proceedings of 27th Annual IEEE Symposium on Foundations of Computer Science, (1986), pp. 337–347.*

[Bar89] Barrington D.A., *Bounded-width polynomial-size branching programs recognize exactly those languages in* NC^1. *J. Comput. System Sci., Vol. 38 (1989), pp. 150–164.*

[BT88] Barrington D.A., Thérien D., *Finite monoids and the fine structure of* NC^1. *J. ACM, Vol. 35 (1988), pp. 941–952.*

[BS90] Boppana R., Sipser M., *The complexity of finite functions. In Handbook of Theoretical Computer Science, Vol. A, Elsevier and MIT Press, 1990, pp. 757–804.*

[BCD+89] Borodin A., Cook S., Dymond P., Ruzzo W., Tompa M., *Two applications of inductive counting for complementation problems. SIAM J. Computing, Vol. 18 (1989), pp. 559–578.*

[Bus87] Buss S., *The Boolean formula value problem is in ALOGTIME. Proceedings of 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 123–131.*

[FSS84] Furst M., Saxe J., Sipser M., *Parity, circuits, and the polynomial time hierarchy. Mathematical Systems Theory, Vol. 17 (1984), pp. 13–27.*

[Gri91] Grigni M., *Structure in Monotone Complexity. Technical Report MIT/LCS/TR-520, Massachusetts Institute of Technology, 1991.*

[GS91] Grigni M., Sipser M., *Monotone separation of logspace from* NC^1. *Proceedings of the Sixth Annual IEEE Conference on Structure in Complexity Theory, 1991, pp. 294–298.*

[Imm88] Immerman N., *Nondeterministic space is closed under complementation. SIAM J. on Computing, Vol. 17 (1988), pp. 935–938.*

[KW90] Karchmer M., Wigderson A., *Monotone circuits for connectivity require super-logarithmic depth. SIAM J. Discrete Mathematics, Vol. 3 (1990), pp. 255–265.*

[Mar62] Markov A.A., *Minimal relay-diode bipoles for monotonic symmetric functions. Problemy Kibernetiki, Vol. 8 (1962), pp. 117–121.* English translation in *Problems of Cybernetics, Vol. 8 (1964), pp. 205–212.*

[MS56] Moore E.F., Shannon C.E., *Reliable circuits using less reliable relays. J. of the Franklin Inst., Vol. 262 (1956), pp. 191–208 and 281–297.*

[RW89] Raz R., Wigderson A., *Probabilistic communication complexity of boolean relations. Proceedings of 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 562–573.*

[RW90] Raz R., Wigderson A., *Monotone circuits for matching require linear depth. Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, 1990, pp. 287–292.*

[Raz85a] Razborov A.A., *Lower bounds on the monotone complexity of some Boolean functions. Doklady Akademii Nauk SSSR, Vol. 281 (1985), pp. 798–801.* English translation in *Soviet Mathematics Doklady, Vol. 31 (1985), pp. 354–357.*

[Raz85b] Razborov A.A., *A lower bound on the monotone network complexity of the logical permanent. Matematicheskie Zametki, Vol. 37 (1985), pp. 887–900.* English translation in *Mathematical Notes of the Academy of Sciences of the USSR, Vol. 37 (1985), pp. 485–493.*

[SV85] Skyum S., Valiant L., *A complexity theory based on boolean algebra. J. ACM, Vol. 32 (1985), pp. 484–502.*

[Sze87] Szelepcsényi R., *The method of forcing for nondeterministic automata. Bull. European Ass. for Theoretical Computer Science, Vol. 33 (1987), pp. 96–100.*

[Tar88] Tardos E., *The gap between monotone and nonmonotone circuit complexity is exponential. Combinatorica, Vol. 8 (1988), pp. 141–142.*

[Val84] Valiant L., *Short monotone formulae for the majority function. Journal of Algorithms, Vol. 5 (1984), pp. 363–366.*

[Ven87] Venkateswaran H., *Properties that Characterize LOGCFL. Proceedings of 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 141–150.*

[Yao89] Yao A., *Circuits and local computation. Proceedings of 21st Annual ACM Symposium on Theory of Computing, 1989, pp. 186–196.*