Counting paths in planar width 2 branching programs

Meena Mahajan, Nitin Saurabh, Karteek Sreenivasaiah

Institute of Mathematical Sciences, Chennai 600113, India.
Email: {meena,nitin,karteek}@imsc.res.in

Abstract

We revisit the problem of counting paths in width-2 planar branching programs. We show that this is hard for Boolean NC^1 under ACC^0[5] reductions, completing a proof strategy outlined in [3]. On the other hand, for several restricted instances of width-2 planar branching programs, we show that the counting problem is TC^0-complete. We also show that non-planar width-2 programs can be planarized in AC^0[2]. Using the equivalence of planar width-2 programs with the reduced-form representation of positive rationals, we show that the evaluation problem for this representation in the Stern-Brocot tree is also NC^1 hard. In contrast, the evaluation problem in the continued fraction representation is in TC^0.

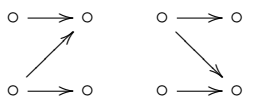
1 Introduction

Barrington's celebrated theorem [5] shows that branching programs (BPs) of bounded width and polynomial size characterize the class NC^1 of languages accepted by Boolean polynomial-size formulas. A natural question to ask is whether this result arithmetizes. That is, does counting paths in bounded width polynomial size branching programs characterize counting proof trees in NC^1 circuits? More generally, do bounded width polynomial size algebraic branching programs characterise arithmetic NC^1? The result of [7] shows that this is indeed the case over rings, and even width 3 suffices; see also [8]. And this result is tight: a very recent result in [4] shows that over arbitrary fields, width 2 algebraic branching programs (ABPs) are not universal; there are efficiently computable polynomials that are provably not computable by width 2 ABPs of any size. For the path-counting version, we are interested in natural numbers, and the operations $+, \times$, so we do not have a field or even a ring structure. We may even assume that the inputs are Boolean (zero-one-valued). Even in this setting, while paths in bounded width polynomial size branching programs can be counted in arithmetic NC^1 (usually stated as: #BWBP$\subseteq$#NC^1), the converse is not known.

Some special cases of this question have been addressed in the literature. In [3], it is shown that in a restricted type of planar BWBP, where the edge connections between adjacent layers must come from a specified set of patterns (call such restricted planar programs rGPs: restricted grid programs), path counting is in fact possible with arithmetic circuits of polynomial size and constant depth $\#AC^0$, and hence even the bit representation of the number of paths can be computed in in TC^0, a subclass of Boolean NC^1. It is also shown that without this grid restriction, path-counting even in width-2 planar BPs is hard for Boolean NC^1 under ACC^0 (mod 5) reductions. In [16, 13], the rGP restriction is explored further. It is shown that Boolean/arithmetic NC^1 is characterized by Boolean/algebraic polynomial size rGPs of any width (this construction works over fields/rings/naturals), and the equivalence holds even if the width is restricted to be logarithmic.

Such fine distinctions between what is possible in AC^0, in TC^0 and in NC^1 are important because the currently known machinery for proving circuit lower bounds stops precisely in this region. We have lower bounds against AC^0, against uniform TC^0 ([2]), but none against NC^1.

In this note, we return to the width 2 case. Counting paths in width k BPs is equivalent (under the weakest possible uniform reductions, projections) to multiplying sequences of $k \times k$ matrices over 0,1. (Each matrix is the adjacency matrix of connections between vertices at consecutive layers of the BP.) At width-2, planar BPs correspond to 2×2 matrices where at least one of the off-diagonal elements is zero. We refer to such matrices as planar matrices. Two planar matrices are of special importance: $L = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $U = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. Products over these matrices are equivalent to planar width-2 BPs where the interconnections between layers is one of the forms shown alongside.



Products over L and U have nice connections to many special numbers. For instance, for $n \geq 1$, $(UL)^n = \begin{pmatrix} F_{2n+1} & F_{2n} \\ F_{2n} & F_{2n-1} \end{pmatrix}$, where F_n is the nth Fibonacci number. (Hence, by the result of [17] that integer matrix powering for constant-order matrices is in TC^0, and the fact that Bit-Count is in TC^0, there is a family of multi-output TC^0 circuits such that C_n, on input x, outputs the binary representation of F_j, where j is the number of 1s in x.) Products over L and U are also intimately connected with the question of representing positive rationals without repetition. The positive rationals are in bijection with and hence can be represented in the following two forms:

1. Reduced form: $\langle m, n \rangle$, where m, n are relatively prime positive integers (or using notation from

[11], $m \perp n$), uniquely represents the rational $\frac{m}{n}$.

2. Continued fractions form: $\langle a_0, a_1, \ldots, a_{k-1} \rangle$, where each a_i is a non-negative integer, $a_0 \geq 0$, $a_i \geq 1$ for $i \geq 1$, $a_{k-1} \geq 2$ unless $k = 0$ in which case $a_0 \geq 1$, uniquely represents the rational

$$a_0 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_{k-1}}}}.$$

We can consider the computational complexity of conversion between the two representations. This is best handled via the Stern-Brocot tree, a well-studied binary search tree in which the vertices are in bijection with the positive rational numbers, and also with products of sequences over the matrices L, U. (See Sections 4.5 and 6.7 in [11]; see also [9].)

Our contributions in this note are as follows:

- We identify and fix a small but subtle flaw in Theorem 16 of [3], which shows that path-counting even in planar width-2 BPs is hard for Boolean NC^1 under ACC^0 (mod 5) reductions. (Section

2 Definitions and Preliminaries

A branching program is a directed acyclic graph in which the vertex set is partitioned into layers $V_0, V_1, \ldots, V_m$, and the edge set E is contained in $\cup_{i=1}^{m} V_{i-1} \times V_i$. Edges are labeled by variables $x_1, \ldots, x_n$ or their negations or the constant 1. There are special nodes $s \in V_0$ and $t \in V_m$. The branching program is said to accept an input $a \in \{0,1\}^n$ if there is a path from s to t where all edge labels take the value 1 under the assignment $x = a$. A family of branching programs $\{B_n\}_{n \geq 0}$ accepts a language L if each B_n accepts exactly $L^{=n}$. BWBP denotes the class of (languages accepted by) branching program families where each B_n has width c and size $O(n^c)$, for some fixed constant c. (Note that in this definition, the branching programs are non-deterministic. Note also that the graphs are required to be layered, since otherwise width does not make sense.)

NC^i denotes the class of (languages accepted by) circuits of polynomial size and $O((\log n)^i)$ depth using bounded fan-in gates. We are concerned with only NC^1 and NC^0 here.

AC^0 denotes the class of (languages accepted by) circuits of polynomial size and $O(1)$ depth using unbounded fan-in $\vee$ and $\wedge$ gates and negation gates. $\mathrm{ACC}^0[p]$ denotes the class of (languages accepted by) circuits of polynomial size and $O(1)$ depth using unbounded fan-in $\vee$ and $\wedge$ gates, negation gates, and MOD_p gates that output a 1 if and only if the number of 1s in their input is non-zero modulo p. The union $\mathrm{ACC}^0[p]$ is denoted ACC^0.

TC^0 denotes the class of (languages accepted by) circuits of polynomial size and $O(1)$ depth using unbounded fan-in Majority gates and negation gates. A Majority gate outputs a 1 if and only if at least half if its inputs are 1.

It is known that $\mathrm{NC}^0 \subseteq \mathrm{AC}^0 \subseteq \mathrm{ACC}^0 \subseteq \mathrm{TC}^0 \subseteq \mathrm{NC}^1 = \mathrm{BWBP}$. Further, if the circuit / branching program families are uniform, then NC^1 languages can be accepted in logarithmic space.

Arithmetic versions of NC^1 and AC^0 are circuits with $+$ and $\times$ gates instead of $\vee$ and $\wedge$ and the same size-depth bounds. It is known that uniform arithmetic NC^1 functions (that is, the bit representations of numbers computed by arithmetic circuits) can be computed in logarithmic space.

Arithmetic versions of BPs (that is, BPs computing functions from strings to numbers) can be defined in many ways. The simplest way is counting paths. A more generalised way is where edges in the BP may be labeled by literals or by integer constants. Such a BP computes the function that adds up the total weight of all paths between two designated nodes. (The weight of a path is the product of the weights of edges on the path.) Path-counting in width-k BPs is equivalent to iterated multiplication, over integers, of $k \times k$ matrices with (0,1) entries. The length of the BP translates to the number of matrices to be multiplied.

(Remark: The "path-counting"model described above is less general than algebraic BPs, defined by Nisan in 1991 ([18]). In that model, the BP computes polynomials over an underlying field; edges can be labeled by arbitrary linear forms. It is also somewhat different from the arithmetic BPs defined by Beimel and Gal [6], which actually decide languages rather than compute functions, but with an acceptance criterion that depends on the path count. It is known folklore that the path-counting model captures classes of counting functions based on nondeterministic machine classes.)

We say that a problem A reduces to a problem B via AC^0 reductions if there is an AC^0 circuit family augmented with oracle gates for B that correctly solves A. Other reductions (ACC^0, TC^0) are analogously defined.

A projection is a mapping $\Sigma^* \to \Delta^*$ where each output symbol depends on at most one input symbol. In particularly, over binary alphabets, a circuit computing the projection merely duplicates and re-routes wires from the inputs to the output.

See for instance [1, 20] for a detailed treatment of these topics.

3 Fixing a flaw in Theorem 16 of [3]

Theorem 16 in [3] (ICALP 1999) says that computing the number of paths in planar width-2 BPs is complete for NC^1 under ACC^0 (mod 5) reductions.

Though the Theorem claims completeness, as is clear from the proof, only hardness is established. In private correspondence, the authors of [3] clarified that the completeness claim is an oversight and they only show hardness. In fact, as far as we know, whether paths in planar width 2 branching programs can be counted in Boolean NC^1 is still open.

The hardness proof as stated is flawed, but fixable. Here is the way the proof is stated.

(a) The 2x2 integer matrices with determinant 1 mod 5, with the binary operation of matrix multiplication in Z_5, form a non-solvable group (commonly denoted SL(2,5)). So, by Barrington's result ([5]), the word problem over this group is complete for NC^1.

(b) By [12] (FOCS 99 Theorem 3.1), every matrix over non-negative integers with determinant 1 can be written as a product of a sequence over the matrices L and U. So the word problem over SL(2,5) reduces uniformly to evaluating products over $L, U$and I. This product is a width-2 planar BP.

(c) Hence every NC^1 language can be reduced to counting paths mod 5 in a width 2 planar BP.

The flaw is in step (b). The matrices U and L have determinant 1 over the integers. Thus any product over U and L will have determinant 1 over the integers. It cannot produce a matrix with determinant, say, 6 or 11. But such matrices are present in SL(2,5). One cannot produce matrices like $\begin{pmatrix}3 & 3\\ 1 & 3\end{pmatrix}$ or $\begin{pmatrix}0 & 2\\ 2 & 0\end{pmatrix}$ using U, L.

So to use Gurevich's construction, one first needs to show that for every matrix M in SL(2,5), there is a matrix N with non-negative integers, with determinant 1 over integers, such that each entry of N is equivalent, modulo 5, to the corresponding entry in M. It turns out that this statement is indeed true, but it is not needed at all. Even Gurevich's construction is not needed. Just replace step (b) in the proof by the following:

(b') Dickson's theorem for finite groups (see for instance [10]; see also the Appendix) tells us that SL(2,5) is exactly the group generated by $\begin{pmatrix}1 & 0\\ 2 & 1\end{pmatrix}$ and U. But the first matrix is just L^2, so L and U generate SL(2,5).

Remark: The group SL(2,5) is a perfect group; it equals its commutator subgroup. Hence, following Barrington's construction, when reducing an NC^1 language to the word problem over SL(2,5), any element of SL(2,5) can be chosen as the accepting element. If we choose, say, the matrix L, which differs from I only in the $[2,1]$ entry, then the hardness result above can be restated as follows:

Theorem 1 (Theorem 16 of [3]) *For every language A in NC^1, there is a uniform polynomial-sized projection $r : \Sigma^* \longrightarrow \{L, U, I\}^*$ such that for every $x \in \Sigma^*$, if $r(x) = M_1 M_2 \dots M_n$, then*

$$x \in A \implies \left(\prod_{i=1}^{n} M_i\right)[2,1] \equiv 1 \bmod 5$$

$$x \notin A \implies \left(\prod_{i=1}^{n} M_i\right)[2,1] \equiv 0 \bmod 5$$

4 Planarizing width-2 BPs

Theorem

We first recall that a simpler reduction (without post-computation) is known (see for instance [1]) in the generalised model where edges are labeled by $\{-1, 0, 1\}$. Robinson [19] showed that every language in NC^1 reduces to the 2-sided Dyck language with two generators. Lipton and Zalcstein [15] showed that the free group on two generators, say g_1, g_2, is isomorphic to the group of invertible matrices over rationals, with the isomorphism taking g_1 to L^2 and g_2 to U^2. Since over rationals, $L^{-1} = L^{-1} = \begin{pmatrix}1 & 0\\ -1 & 1\end{pmatrix}$ and $U^{-1} = \begin{pmatrix}1 & -1\\ 0 & 1\end{pmatrix}$, we can put these together to obtain the following:

Proposition 2 ([19], [15]) *For every language A in NC^1, there is a uniform polynomial-sized projection $r : \Sigma^* \longrightarrow \{L, U, L^{-1}, U^{-1}, I\}^*$ such that for every $x \in \Sigma^*$, if $r(x) = M_1 M_2 \dots M_n$, then*

$$x \in A \Longleftrightarrow \prod_{i=1}^{n} M_i = I$$

(All arithmetic is over integers.)

Note that for 2×2 matrices over integers, we can consider restrictions of differing degrees: (1) Only (0,1) entries, (pure path-counting) (2) Only non-negative integers, (3) Only $\{-1, 0, 1\}$ entries, or (4) Any integers. And for each of these cases, we have planar and non-planar matrices. (Recall that we say a 2×2 matrix is planar if it has at least one off-diagonal entry that is zero.) Theorem

Here, we observe that such a reduction from NC^1, if one exists, will need a different technique, since we provably cannot planarize such products via a pure projection (without post-computation). The reasons are simple: firstly, all planar (0,1) matrices have determinant 0 or 1, and secondly, their products have non-negative integers. Hence products over them cannot generate the matrices $\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$ and $\begin{pmatrix}2 & 1\\ 0 & 1\end{pmatrix}$, with determinants -1 and 2 respectively. In fact, planar non-negative matrices have non-negative determinants, and planar $\{-1, 0, 1\}$ matrices have determinant in $\{-1, 0, 1\}$, so we cannot trade off planarity for different restrictions on the entries.

We show below that we can planarize (0,1) matrix products without post-computation, provided we relax the requirement that the reduction be a projection. That is, we allow more pre-computation, and we piece together the final matrix via a projection. This is good enough in the computational settings we are interested in.

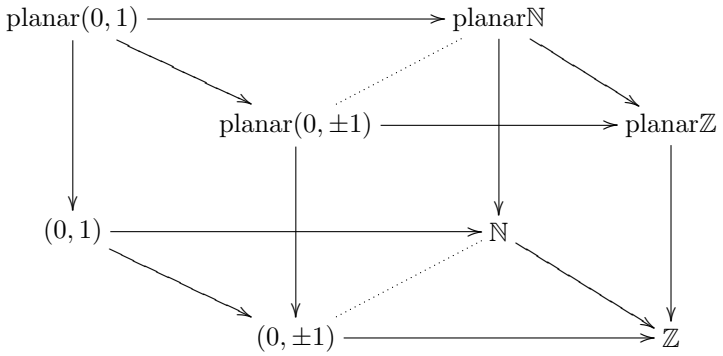


Figure 1: Different cases for width-2 BPs. Arrows denote "special case of". Dotted lines denote incomparability.

Theorem 3 *Path-counting in width 2 BPs reduces to Path-counting in planar width-2 BPs via uniform $AC^0[2]$ reductions.*
More precisely, there is an $AC^0[2]$ circuit family $\{C_n\}$ such that given any sequence of 2×2 $(0,1)$ matrices $\langle M_1, M_2, \ldots, M_n\rangle$, C_n outputs a sequence of 2×2 $(0,1)$ planar matrices $\langle U_1, U_2, \ldots, U_{2n+1}\rangle$, and two more $(0,1)$ planar matrices $U^{(1)}$ and $U^{(2)}$, satisfying the following $\forall u, v \in \{1,2\}$:

$$\left(\prod_{i=1}^{n} M_i\right)[u,v] = \left(\left(\prod_{j=1}^{2n+1} U_j\right) U^{(v)}\right)[u,v]$$

Proof. We use the following equivalences:

$$\begin{pmatrix}0 & 1\\ 1 & 1\end{pmatrix} = LX;\ \begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix} = UX$$

$$\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix} = \begin{pmatrix}0 & 1\\ 0 & 1\end{pmatrix}\begin{pmatrix}0 & 0\\ 1 & 1\end{pmatrix};$$

In the first stage, replace each matrix M_i by the pair A_{2i-1}, A_{2i}, where, if $M_i = X$, then $(A_{2i-1}, A_{2i}) = (I, X)$, if M_i equals any of the other non-planar matrices, use one of the equivalences above, and if M_i is planar, then $(A_{2i-1}, A_{2i}) = (M_i, I)$. This gives a sequence of length $2n$ where the only non-planar matrices are all X. Further, set $A_{2n+1} = A_{2n+2} = X$. Since $X^2 = I$, we have $\prod_{i=1}^{n} M_i = \prod_{j=1}^{2n+2} A_j$.

The idea now is to pair up the Xs and let them demolish each other. Note that if $D_1, \ldots, D_t$ are planar matrices, then $X\left(\prod_{i=1}^{t} D_i\right)X = IX\left(\prod_{i=1}^{t}(D_iXX)\right)XI = I\left(\prod_{i=1}^{t}(XD_iX)\right)X^2I$. So in the sequence of matrices (A_i), we can locally replace the A_is that occur between the pairs by XA_iX, and the Xs by Is. Since there may be an odd number of Xs to begin with, we pad the sequence with the two Xs at the end, and use one of them if necessary to complete the pairing. Detecting whether an A_i occurs between a pair rather than between pairs requires a parity computation; hence the reduction is an $\mathrm{AC}^0[2]$ reduction. The last crucial observation is that for planar D, the matrix XDX is also planar. The details follow.

For $j = 1, \ldots, 2n+1$ define bits b_j, c_j as follows:

$$b_j = \begin{cases} 1 & \text{if } A_j = X \\ 0 & \text{otherwise} \end{cases}$$

$$c_j = \sum_{i=1}^{j} b_j \bmod 2$$

For $j = 1 \ldots 2n$, define matrices B_j as follows:

$$B_j = \begin{cases} I & \text{if } A_j = X \\ A_j & \text{if } A_j \neq X \text{ and } c_j = 0 \\ XA_jX & \text{if } A_j \neq X \text{ and } c_j = 1 \end{cases}$$

Further, if $c_{2n} = 0$ then $B_{2n+1} = B_{2n+2} = I$, otherwise $B_{2n+1} = I$ and $B_{2n+2} = X$. ($c_{2n} = 1$ means that A_{2n+1} will be paired to its left, so A_{2n+2} remains X.) It follows that $\prod_{i=1}^{n} M_i = \prod_{j=1}^{2n+2} B_j$.

If $B_{2n+2} = I$, then we have obtained a planar product. The reduction outputs $U_j = B_j$ for $j = 1, \ldots, 2n+1$, and $U^{(1)} = U^{(2)} = B_{2n+2}$.

If $B_{2n+2} = X$, we define $U^{(1)}$ and $U^{(2)}$ such that we can separately extract the columns of the product matrix, and eliminate B_{2n+2}. It suffices to choose $U^{(1)} = \begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix}$ and $U^{(2)} = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}$ □

5 Some special cases of 2×2 iterated matrix multiplication over non-negative integers

For a width 2 planar BP, the interconnections between adjacent layers may be from any of the 11 patterns shown in Figure

Let $\mathcal{C}$ be the set of 8 matrices corresponding to planar interconnections other than I, L, U. Our first bound shows that over $\mathcal{C} \cup \{I\}$, that is if neither L nor U appear, then path-counting is easy.

Lemma 4 *Path-counting in width-2 planar BPs where neither of the interconnection patterns L, U appears is in TC^0.*

Proof. Assume there is no I in the interconnection patterns; if there are, we preprocess the sequence and move all occurrences of I to the

Theorem 5 *Path-counting in width-2 planar BPs is hard for TC^0 even if both of the interconnection patterns L, U do not appear. That is, Computing products of sequences of matrices from the set $\mathcal{C}$ is hard for TC^0.*

Proof. The canonical complete problem for TC^0 is checking whether at least half of the input bits are 1. Given a sequence $b_1, \ldots, b_n$, construct the sequence of matrices $M_1, \ldots, M_{2n}$ where

$$M_{2i-1}, M_{2i} = \begin{cases} I, I & \text{if } b_i = 0 \\ \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} & \text{otherwise} \end{cases}$$

Let $M = \prod M_i$, and let $a_n a_{n-1} \ldots a_0$ be the binary representation of $M[1,1]$. Then $\sum_i b_i \geq n/2 \iff \vee_{j=n/2}^{n} a_j = 1$. □

Next we show that computation is easy even if both L and U appear, provided they are always "well-separated".

Theorem 6 *Path-counting in width-2 planar BPs where occurrences of the interconnection patterns L, U are separated by at least one matrix that is not in $\{L, U, I\}$ is in TC^0.*

Proof. Assume there is no I in the interconnection patterns; if there are, we preprocess the sequence in TC^0 and move all occurrences of I to the end. Let the sequence of matrices be $M_1, \ldots, M_n$.

Imagine a boundary placed after each M_i satisfying any one of the following conditions:

1. $M_i = L$ and $M_{i+1} \neq L$,
2. $M_i = U$ and $M_{i+1} \neq U$,
3. $M_i \neq L$ and $M_{i+1} \neq U$.

(Assume $M_{n+1} = I$ for testing this condition.)

Now mark alternate boundaries starting from the beginning.

Recall that $\mathcal{C}$ is the set of 8 matrices corresponding to planar interconnections other than I, L, U. Between any two marked boundaries, the subsequence of matrices has the form AB where $A, B \in \mathcal{C} \cup \{L^k, U^k \mid k \in \mathbb{Z}^{>0}\}$ and at least one of A, B is in $\mathcal{C}$. For each such subsequence, the product AB is a matrix of one of the following forms:

$$\begin{pmatrix} 0 & 0 \\ \alpha & \beta \end{pmatrix}, \begin{pmatrix} \alpha & \beta \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & \alpha \\ 0 & \beta \end{pmatrix}, \begin{pmatrix} \alpha & 0 \\ \beta & 0 \end{pmatrix},$$

$$\begin{pmatrix} 1 & 1 \\ \alpha & \alpha \end{pmatrix}, \begin{pmatrix} \alpha & \alpha \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} \alpha & 1 \\ \alpha & 1 \end{pmatrix}, \begin{pmatrix} 1 & \alpha \\ 1 & \alpha \end{pmatrix}$$

where α and β are non negative integers. Each of these can thus be decomposed as follows.

$$\begin{pmatrix} 0 & 0 \\ \alpha & \beta \end{pmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} [\alpha \quad \beta]; \qquad \begin{pmatrix} 0 & \alpha \\ 0 & \beta \end{pmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} [0 \quad 1];$$

$$\begin{pmatrix} \alpha & \beta \\ 0 & 0 \end{pmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} [\alpha \quad \beta]; \qquad \begin{pmatrix} \alpha & 0 \\ \beta & 0 \end{pmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} [1 \quad 0];$$

end. This involves only counting the number of occurrences of I to the left of each position, and hence can be done in TC^0.

The matrices corresponding to other 8 patterns can be decomposed as follows:

$$\begin{pmatrix}1 & 1\\ 0 & 0\end{pmatrix} = \begin{bmatrix}1\\0\end{bmatrix}[1 \quad 1]; \qquad \begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix} = \begin{bmatrix}1\\0\end{bmatrix}[1 \quad 0];$$

$$\begin{pmatrix}0 & 0\\ 1 & 1\end{pmatrix} = \begin{bmatrix}0\\1\end{bmatrix}[1 \quad 1]; \qquad \begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix} = \begin{bmatrix}0\\1\end{bmatrix}[1 \quad 0];$$

$$\begin{pmatrix}1 & 0\\ 1 & 0\end{pmatrix} = \begin{bmatrix}1\\1\end{bmatrix}[1 \quad 0]; \qquad \begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix} = \begin{bmatrix}0\\1\end{bmatrix}[0 \quad 1];$$

$$\begin{pmatrix}0 & 1\\ 0 & 1\end{pmatrix} = \begin{bmatrix}1\\1\end{bmatrix}[0 \quad 1]; \qquad \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix} = \begin{bmatrix}1\\0\end{bmatrix}[0 \quad 1].$$

Now we show how to construct a TC^0 circuit family to evaluate an iterated product of a sequence over the above 8 matrices. Each matrix in the sequence is given as a 4-bit string. Let the ith matrix be decomposed as $\begin{bmatrix}v_{i1}\\v_{i2}\end{bmatrix} \cdot [v_{i3} \quad v_{i4}]$. Regrouping the terms in the product, we want to compute

$$\begin{aligned}
& M_1 M_2 \ldots M_n \\
= \; & \left(\begin{bmatrix}v_{11}\\v_{12}\end{bmatrix} \cdot [v_{13} \quad v_{14}]\right)\left(\begin{bmatrix}v_{21}\\v_{22}\end{bmatrix} \cdot [v_{23} \quad v_{24}]\right) \\
& \ldots \left(\begin{bmatrix}v_{n1}\\v_{n2}\end{bmatrix} \cdot [v_{n3} \quad v_{n4}]\right) \\
= \; & \begin{bmatrix}v_{11}\\v_{12}\end{bmatrix} \cdot \left([v_{13} \quad v_{14}]\begin{bmatrix}v_{21}\\v_{22}\end{bmatrix}\right)\left([v_{23} \quad v_{24}]\begin{bmatrix}v_{31}\\v_{32}\end{bmatrix}\right) \\
& \ldots [v_{n3} \quad v_{n4}] \\
= \; & \left\{\left([v_{13} \quad v_{14}]\begin{bmatrix}v_{21}\\v_{22}\end{bmatrix}\right)\left([v_{23} \quad v_{24}]\begin{bmatrix}v_{31}\\v_{32}\end{bmatrix}\right)\ldots\right\} \\
& \left\{\begin{bmatrix}v_{11}\\v_{12}\end{bmatrix} \cdot [v_{n3} \quad v_{n4}]\right\} \\
= \; & (a_1 \times a_2 \times \ldots \times a_{n-1})\, A
\end{aligned}$$

Layer 1 (Decomposition): Obtain from each matrix M_i the corresponding row and column vectors. This can be done in NC^0.

Layer 2 (Inner product): For each $1 \leq i < (n-1)$, compute the product $a_i = [v_{i3} \quad v_{i4}]\begin{bmatrix}v_{(i+1)1}\\v_{(i+1)2}\end{bmatrix}$. Since each v_{ik} is 0 or 1, this can be done in NC^0, and gives a sequence of integers $a_1 \ldots a_{n-1}$ each in the range $\{0, 1, 2\}$. Also compute the 2×2 matrix $A = \begin{bmatrix}v_{11}\\v_{12}\end{bmatrix} \cdot [v_{n3} \quad v_{n4}]$; this can also be done in NC^0. Each entry in A is 0 or 1.

Layer 3 (Iterated multiplication): Compute $a = a_1 \times a_2 \times \ldots \times a_{n-1}$. This can be done in TC^0.

Layer 4 (Scalar product): Finally, compute aA. Since A is a 0-1 matrix, this requires only NC^0 circuitry. □

It is easy to see that this upper bound is tight:

$$\begin{pmatrix}1 & 1\\ \alpha & \alpha\end{pmatrix} = \begin{bmatrix}1\\\alpha\end{bmatrix}[1 \quad 1]; \qquad \begin{pmatrix}\alpha & \alpha\\ 1 & 1\end{pmatrix} = \begin{bmatrix}\alpha\\1\end{bmatrix}[1 \quad 1];$$

$$\begin{pmatrix}\alpha & 1\\ \alpha & 1\end{pmatrix} = \begin{bmatrix}1\\1\end{bmatrix}[\alpha \quad 1]; \qquad \begin{pmatrix}1 & \alpha\\ 1 & \alpha\end{pmatrix} = \begin{bmatrix}1\\1\end{bmatrix}[1 \quad \alpha].$$

Now the strategy is similar to that used in proving Lemma

To see that all these operations can be done in TC^0, note that

1. Delineating alternate boundaries requires only counting modulo 2.
2. To obtain products within a subsequence, we count the maximal number of consecutive L's or U's (in TC^0) and then perform integer addition (in AC^0).
3. All products have $O(\log n)$ bit entries, so the decomposition can be done in AC^0.
4. For the same reason, inner products can also be computed in AC^0.
5. Multiplying the obtained scalars is a TC^0 operation.
6. The remaining $O(1)$ multiplications of 2×2 matrices is also a TC^0 operation.

□

Observe that the above operations continue to be in TC^0 even for numbers represented with $O(n)$ bits. Thus

Corollary 7 *Products of sequences of matrices from the set $\mathcal{C} \cup \{L^k, U^k \mid k \in \mathbb{Z}^{>0}\}$ can be computed in TC^0.*

Finally we observe that if both L and U appear, not well-separated but in a "regular" fashion, then computation is easy.

Lemma 8 *Products of the form $(L^a U^b)^m$ can be computed in TC^0.*

Proof. This follows from the facts that $L^a U^b = \begin{pmatrix}1 & b\\ a & ab+1\end{pmatrix}$, and that powering of $O(1)$-sized matrices is in TC^0 ([17]).

(Note that computing ab is not an issue: from the input sequence, we can compute a and b in TC^0, and these numbers are implicitly given in unary representation in the input.) □

6 Locating rationals in the Stern-Brocot tree

The problem of path-counting in planar width-2 BPs is closely connected with that of locating positive rationals in the well-studied Stern-Brocot tree. We describe the tree and the connection below.

The Stern-Brocot tree is an infinite binary tree whose nodes are in bijection with the set of positive rationals. The labeling of nodes with rationals is such that the tree forms a binary search

tree. The labeling is constructive (see sections 4.5 and 6.7 of [11]); however, the complexity of computing the labeling depends on the representation chosen for the rationals. The bijection between the tree itself and the positive rationals can be described as follows: Each node of the tree is associated with an (open) interval and a "centre", or a mediant. The interval is described by a 4-tuple $\langle a, b, c, d\rangle$ and is the set of all positive rationals q such that $\frac{a}{b} < q < \frac{c}{d}$. The rational $\frac{a+c}{b+d}$ is associated with the node; we refer to it as the mediant for the interval. A node with interval $\langle a, b, c, d\rangle$ has as its children the nodes with intervals $\langle a, b, a+c, b+d\rangle$ and $\langle a+c, b+d, c, d\rangle$ respectively. The root of the tree is associated with the interval $\langle 0, 1, 1, 0\rangle$ and has 1 as the mediant. It is well-known that the representation of the mediant so obtained is already in reduced form.

The following computational question concerning locating rationals in the Stern-Brocot tree is intimately connected to the question of path-counting in planar width-2 BPs:
Stern-Brocot Evaluation: Given a binary string w denoting a path from the root of the Stern-Brocot tree, find the representation of the positive rational at the node reached.
We describe the connection in Lemmas

It is known that every 2×2 matrix over non-negative integers with determinant 1 can be written as a product of a sequence over L, U (see for instance [12] Thm 3.1). And every sequence over L, U gives such a matrix. These sequences are also exactly the sequences that arise in computing the reduced form representation of a rational. Thus path counting in width-2 planar BPs allows us to solve the Evaluation problem for the reduced form representation of rationals; a Boolean NC^1 circuit for the former implies one for the latter. More formally,

Lemma 9 *The Stern-Brocot Evaluation problem, where the output is required to be in reduced form, can be solved by AC^0 circuits with oracle gates for counting paths in planar width-2 BPs.*

Proof. The circuit is constructed as follows: convert each bit in w to an instance of L or U to obtain a sequence of matrices $M_1, \ldots, M_{|w|}$, and feed this sequence to oracle gates that compute the bits of the planar width-2 Path Counting problem. The outputs of the oracle gates are the binary representations of the 4 numbers m, m', n, n' in the product matrix $\begin{pmatrix} n & n' \\ m & m' \end{pmatrix}$. The desired rational (the mediant at the node of the tree specified by path w) is then given by $\frac{m+m'}{n+n'}$, and $(m + m') \perp (n + n')$; see [11] for details. So placing appropriate AC^0 circuitry above the oracle gates yields the desired reduced form representation. □

We now show the converse: the Evaluation problem for the reduced form representation can be used to perform path counting in planar width-2 BPs.

Lemma 10 *The bit representation of the number of paths in planar width-2 BPs can be computed by TC^0 circuits with oracle gates for the Stern-Brocot Evaluation problem where the rationals are output in reduced form.*

Proof. As described in Section

Use the equivalence between planar path-counting and multiplying planar matrices. Let $M_1, M_2, \ldots, M_n$ be the given sequence of matrices to be multiplied; each M_i is one of L, U, I. Since sorting is in TC^0, we can sift out all occurrences of I to the end, getting the sequence $N_1, N_2, \ldots, N_k$ followed by $n - k$ occurrences of I. Now each N_i is either L or U. Encode L as 1 and U as 0 to obtain a binary string $w = w_1 \ldots w_k$, which is fed to the oracle gate for Evaluation. Let $\langle m, n\rangle$ be the output of Evaluation on w. As described in Equation 4.34 in [11], if $\prod N_i = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, then $m = C + D$ and $n = A + B$. To retrieve C, D from m and A, B from n, let $\langle m', n'\rangle$ be the output of Evaluation on $w_1 \ldots w_{k-1}$. Assume $w_k = 0$, the other case is handled identically. Then
$\begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} E & F \\ G & H \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} E+F & F \\ G+H & H \end{pmatrix}$,
and $m' = G + H$ and $n' = E + F$. Thus we can construct the required output: $\prod_{i=1}^{k} N_i = \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} n' & n - n' \\ m' & m - m' \end{pmatrix}$. Since addition and subtraction are in AC^0, this part is an AC^0 reduction.

One minor detail is that the number k of non-trivial matrices is a variable, whereas the oracle gate has a fixed number of inputs. To handle this, use oracle gates for all values of k from 1 to n, and use additional circuitry to determine which is the correct value. This additional circuitry only needs to obtain the correct count k, and hence can be implemented in TC^0. □

¿From Lemma

The other commonly used representation for positive rationals is the continued fraction representation. In this representation, however, the Stern-Brocot Evaluation problem is significantly easier:

Lemma 11 *The Stern-Brocot Evaluation problem, where the output is required in the continued fraction representation, is in uniform TC^0.*

Proof. We follow the presentation from [11]; the only additional thing is the observation that the required computations are in TC^0.

We are given $w \in \{0,1\}^*$. For $w = \epsilon$, the rational is 1, with representation $\langle 1\rangle$. Otherwise, let w be a string of length $n \geq 1$, written as $1^{a_0}0^{a_1}1^{a_2}\ldots0^{a_{k-1}}$ where k is even, $a_0 \geq 0$, $a_{k-1} \geq 0$, and all other $a_i \geq 1$. Then the rational at the node reached is

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots + \cfrac{1}{a_{k-1} + \frac{1}{1}}}}.$$

So the continued fraction representation is $\langle a_0, a_1, \ldots, a_{k-1} + 1\rangle$ if $a_{k-1} \geq 1$, and

$\langle a_0, a_1, \ldots, a_{k-2} + 1\rangle$ if $a_{k-1} = 0$. We fix an encoding where the output has n numbers $a_1, \ldots, a_n$, each $\log n$ bits long, and a control block of length $\log n$ that tells us how many of these numbers are useful. Constructing the encoding only requires counting how many blocks precede a bit position; since Bit-Count is in TC^0, the encoding can be computed in TC^0. □

Thus in a concrete computational setting the reduced form representation is computationally harder to work with than the continued fraction representation.

One can also ask the the following computational question which is in some sense the inverse of the Evaluation problem:
Stern-Brocot Path-search: Given the representation of a positive rational r, and given an index i, find the ith bit of the path from the root of the Stern-Brocot tree leading to r. (The path may be described as a sequence of moves `left`, `right` starting from the root. Or, coding these as 0 and 1, the path can simply be described as a binary string w.)
Note that the length of the path from a node to the root of the tree is polynomial in the value of the rational, not in the bit size. (eg. the rational $N = 2^n$ with bit size n in both representations appears along the rightmost path at distance N from the root.) So in looking for feasible computation, we specify an index position as above and ask for the bit there, rather than asking for the entire path.

For this question too, the reduced form representation seems harder. For the continued fraction representation, we have a TC^0 upper bound:

Lemma 12 *In the continued fraction representation, Stern-Brocot Path-Search is in uniform TC^0.*

Proof. Essentially invert the process described in Lemma

Given $\langle a_0, a_1, \ldots, a_{k-1}\rangle$, the path is $1^{a_0}0^{a_1}1^{a_2}\ldots 0^{a_{k-1}-1}$ if k is even, and is $1^{a_0}0^{a_1}\ldots 1^{a_{k-1}-1}$ if k is odd. Note that the the length of the path is at most n, but its actual length depends on the blocks. So the circuit family we design simply computes all the prefix sums $\sum_{i=0}^{j} a_i$ and locates i in the correct block with comparisons. Computing the block lengths, checking whether k is even or odd, and comparing numbers can all be done in TC^0. □

For the reduced form representation, however, we have no upper bound other than P. The problem is related to the Extended Euclidean greatest-common-divisor (gcd) algorithm but could well be easier. Recall that given two numbers a, b, the extended Euclidean gcd algorithm performs a number of steps proportional to $\max\{\lceil\log a\rceil, \lceil\log b\rceil\}$, and finally yields not only $g = \gcd(a, b)$ but also integers such that $as+bt = g$. At each step j, it subtracts some multiple m_j of the smaller number from the larger. If m, n are co-prime, then these multiples precisely describe the path: m_1 moves left, m_2 moves right and so on. Thus any upper bound for implementing the extended Euclidean gcd algorithm also yields an upper bound for the Stern-Brocot Path-Search problem in reduced form representation. Note that to date we do not even know whether the gcd of two integers can be computed in NC,not just by the extended Euclidean method, but by any method whatsoever. However, the instances arising here are somewhat easier in the sense that the numbers are known *a priori* to be co-prime. it is conceivable that for such instances, the extended Euclidean method has a parallel implementation that yields the intermediate multiples.

Another question to which also we do not know the answer concerns conversion between the two representations of positive rationals. An obvious way is to go from a representation to the path in the Stern-Brocot tree via Path-Search, and from the path to the other representation via Evaluation. This approach, however, does not work because not only do we not have good upper bounds for Path-Search, but we also know that the path itself can be exponentially long. Generating it as a sub-computation is not a feasible option.

7 Open questions

Several questions are still open.

Regarding Stern-Brocot trees: What is the complexity of these problems?

1. Given m, n in binary with $m \perp n$, and given an index i, find the ith bit of the path w in the Stern-Brocot tree leading to the node labeled m/n.
   The path can be found by repeatedly applying the steps of the gcd algorithm, but this process seems inherently sequential.
2. Given m, n in binary with $m \perp n$, given an index i, and also given proof that $m \perp n$ via non-negative integers s, t such that $ms = nt + 1$, find the ith bit of path w in the Stern-Brocot tree leading to the node labeled m/n.
   Same problem as above, but now we have additional information in s, t.

The most intriguing question in this context, of course, is pinpointing the complexity of computing greatest common divisors.

Regarding path counting, too, there are several open problems:

1. Is #BWBP equal to $\#NC^1$? That is, can arithmetic formulas over literals be expressed as path counting problems in constant-width BPs? (We know this to be the case if we allow negative constants, [7].)
2. Can path counting in width 2 BPs be done in NC^1? That is, is width-2 #BWBP in Boolean NC^1?
3. Is all of $\#NC^1$ in Boolean NC^1? Note that the gap here is very small; it is known ([14], see also [1]) that bits of $\#NC^1$ functions can be computed by polynomial size bounded fan-in Boolean circuits of depth $O(\log n \log^* n)$. The $O(\log^* n)$ gap has not been closed for the last 25 years.

Acknowledgements

The development of this note was heavily influenced by discussions the first author had with Eric Allender and Samir Datta. Discussions with Kristoffer Hansen first indicated the planarization for width-2 BPs.

References

[1] E. Allender. Arithmetic circuits and counting complexity classes. In Jan Krajicek, editor, *Complexity of Computations and Proofs*, Quaderni di Matematica Vol. 13, pages 33–72. Seconda Universita di Napoli, 2004. An earlier version appeared in the Complexity Theory Column, SIGACT News 28, 4 (Dec. 1997) pp. 2-15.

[2] Eric Allender. The permanent requires large uniform threshold circuits. *Chicago Journal of Theoretical Computer Science*, 1999(7), August 1999.

[3] Eric Allender, Andris Ambainis, David Mix Barrington, Samir Datta, and Huong LêThanh. Bounded depth arithmetic circuits: Counting and closure. In *Automata, Languages and Programming ICALP, LNCS 1644*, pages 702–702, 1999. full version at ECCC; TR99-012.

[4] Eric Allender and Fengming Wang. On the power of algebraic branching programs of width 2. In *ICALP, LNCS 6755*, pages 736–747, 2011.

[5] David Mix Barrington. Bounded-width polynomial size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[6] Amos Beimel and Anna Gál. On arithmetic branching programs. *J. Comput. Syst. Sci.*, 59:195–220, 1999.

[7] M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21:54–58, 1992.

[8] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998. Preliminary version in Proceedings of the 11th IEEE Conference on Computational Complexity, 1996, 12–21.

[9] Jeremy Gibbons, David Lester, and Richard Bird. Functional pearl: Enumerating the rationals. *J. Funct. Program.*, 16:281–291, May 2006.

[10] D. Gorenstein. *Finite groups*. Harper and Row, New York, 1968.

[11] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994.

[12] Y. Gurevich. Matrix decomposition problem is complete for the average case. In *SFCS '90: Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 802–811 vol.2, 1990.

[13] M Jansen, M Mahajan, and B V Raghavendra Rao. Resource trade-offs in syntactically multilinear arithmetic circuits. *Computational Complexity*, page to appear, 2011.

[14] Hermann Jung. Depth efficient transformations of arithmetic into boolean circuits. In *Fundamentals of Computation Theory*, FCT '85, pages 167–174, London, UK, 1985. Springer-Verlag.

[15] Richard J. Lipton and Yechezkel Zalcstein. Word problems solvable in logspace. *J. ACM*, 24:522–526, July 1977.

[16] Meena Mahajan and B. V. Raghavendra Rao. Arithmetic circuits, syntactic multilinearity and skew formulae. In *MFCS, LNCS vol. 5162*, pages 455–466, 2008. full version in ECCC TR08-048.

[17] C. Mereghetti and B. Palano. Threshold circuits for iterated matrix product and powering. *Theoretical Informatics and Applications*, 34:39–46, 2000.

[18] Noam Nisan. Lower bounds for non-commutative computation. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, STOC '91, pages 410–418, 1991.

[19] David Hill Robinson. *Parallel algorithms for group word problems*. PhD thesis, University of California at San Diego, La Jolla, CA, USA, 1993.

[20] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York Inc., 1999.

Appendix

A self-contained constructive proof of Dickson's theorem for SL(2,p)

Let $X = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ be an element of SL(2,p); $ad - bc = 1 \bmod p$. Then X can be expressed, mod p, as the product of a sequence of $4(p-1)$ matrices from L, U, I as follows: (Sequences below are of length at most $4(p-1)$; pad with Is.)

1. If $a = d = 1$, then $bc = 0$. So X is one of I, L^c, U^b.
2. If $c \neq 0$, then

$$\begin{aligned} X &= \begin{pmatrix} a & b \\ c & d \end{pmatrix} \\ &= \begin{pmatrix} 1 & e \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ c & 1 \end{pmatrix} \begin{pmatrix} 1 & f \\ 0 & 1 \end{pmatrix} \end{aligned}$$

where $e = (a-1)c^{-1}$ and $f = b-(a-1)c^{-1}d$. The corresponding width-2 program has

length at most $3(p-1)$, since each of the matrices on the right above is of the form L^k or U^k for some $k \leq (p-1)$.

3. If $c = 0$, then $a \neq 0$. Now write

$$X = \begin{pmatrix} a & b \\ 0 & d \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ (p-1) & 1 \end{pmatrix} \begin{pmatrix} a & b \\ a & b+d \end{pmatrix}$$

and then use the above step. The corresponding width-2 program has length at most $4(p-1)$.