Electronic Colloquium on Computational Complexity, Report No. 48 (2001)

BRANCHING PROGRAM, COMMUTATOR, AND ICOSAHEDRON, PART I

JUI-LIN LEE

ABSTRACT. In this paper we give a direct proof of $N_0 = N_0'$, i.e., the equivalence of uniform NC^1 based on different recursion principles: one is OR-AND complete binary tree (in depth $\log n$) and the other is the recursion on notation with value bounded in $[0, k]$ and $|x|(= n)$ many steps. A byproduct is that the multiple product of $p(\log n)$ many Boolean matrices with size $q(\log n) \times q(\log n)$ (where p, q are polynomials and n is the input size) is computable in uniform NC^1. We also investigate the computational power of $LR(f), RL(f), DC(f)$ according to the associativity and commutativity of f and the size of B.

1. INTRODUCTION

In this paper we will focus on uniform parallel complexity classes within NC^1.

Consider the following two types of computational schemes: Divide-and-Conquer and sequential recursion. Apply them on a simple binary operation $f : B \times B \to B$ with a nonempty finite domain B, i.e., for input sequence $\overrightarrow{b} = b_1 b_2 \dots b_n \in B^*$, the scheme Divide-and-Conquer computes the following:

$$\begin{array}{ll} \text{for } n = 2, & DC(f)[b_1 b_2] = f(b_1, b_2); \\ \text{for } n = 2^{m+1}, & DC(f)[b_1 \dots b_{2^{m+1}}] = f(DC(f)[b_1 \dots b_{2^m}], DC(f)[b_{2^m+1} \dots b_{2^{m+1}}]). \end{array}$$

And the from-left-to-right sequential recursion computes the following:

$$\begin{array}{ll} \text{for } n = 2, & LR(f)[b_1 b_2] = f(b_1, b_2); \\ \text{for } n > 2, & LR(f)[b_1 \dots b_n] = f(LR(f)[b_1 \dots b_{n-1}], b_n). \end{array}$$

The dual from-right-to-left sequential recursion $RL(f)[b_1 \dots b_n]$ is defined similarly.

The Boolean formula depths for $DC(f)$ and $LR(f)$ are $O(\log n)$ and $O(n)$ respectively. Though these two schemes seem different, they have the same computational strength, i.e., uniform NC^1= uniform $AC^0 + \{DC(f) \mid f$ is a finite function$\}$ = uniform $AC^0 + \{LR(f) \mid f$ is a finite function$\}$. It is shown that $N_0(= A_0 + tree) = ALOGTIME$ in [6] and $N_0'(= A_0 + k\text{-BRN}) = ALOGTIME$ in [7]. Here A_0 is the function algebra for uniform AC^0, $tree(x)$ is the OR-AND complete binary tree applying to the first 4^m bits of x where $4^m < |x| \leq 4^{m+1}$ (actually $tree(x) = DC(|)$, where $|$ is the Sheffer's stroke), and k-BRN is the recursion on notation with value bounded in $[0, k]$ and $|x|$ many steps of recursion.

In this paper we first prove $N_0 = N_0'$ directly. In Section 2 we introduce function algebras which characterize uniform $AC^0, AC^0(Mod_k), NC^1$. In Section 3 we introduce the notion of expressibility. We prove that $N_0' \subseteq N_0$ in Section 4. To compute k-BRN by $tree(x)$, the idea is straightforward: instead of computing k-BRN iteratively, we express each step of recursion, which is a finite function (for k

Date: May 31, 2001.

This research was supported by the NSC 88-2411-H-194-027.

ISSN 1433-8092

is a constant), as a Boolean matrix of size $(k+1) \times (k+1)$. Since the composition of functions (or Boolean matrix product) is associative, one can use Divide-and-Conquer to compute k-BRN in depth $O(\log n)$. Note that we still need to check the uniformity, i.e., what kind of computation will suffice to arrange the bits in x to plug into $tree$ for simulating k-BRN by $tree$.

We apply this technique to get the following result: The product of $p(\log n)$ many Boolean matrices with size $q(\log n) \times q(\log n)$ (where p, q are polynomials and n is the input size) is computable in uniform NC^1 (see Theorem 4.10).

In Section 5 we prove that $N_0 \subseteq N_0'$. The converse direction is based on [2]. Barrington used the simple group A_5 to simulate logic connective $\wedge$ by branching program. The crucial part is the existence of three 5-cycles σ, δ, τ with $\sigma\delta\sigma^{-1}\delta^{-1} = \tau$, i.e., 5-cycles do not degenerate in commutator operation. Again we need to check the uniformity.

The uniformity we use here is actually quite restricted. To use $LR(f)$ (or $DC(f)$) to simulate other function (say, g(x)), we need to generate an input sequence $\overrightarrow{b}$ to plug into $LR(f)$ (or $DC(f)$). $\overrightarrow{b}$ is of polynomial size (with respect to $|x|$), and its bits are from $\{0, 1, x_i, \dots, \neg x_i, \dots\}$. During the arrangement of $\overrightarrow{b}$, we know nothing about x except its length (actually only knowing an upper bound of the length will be enough).

We are interested in the computational power of uniform $AC^0 + LR(f)$ (or $DC(f)$) with a single f. That is because somehow the proof of $N_0 = N_0'$ is based on NC^1 complete functions $tree (= DC(|))$ and $LR(\circ) (= DC(\circ)$, where $\circ$ is the group multiplication on A_5).

We investigate the computational power of LR, RL, DC according to the associativity and commutativity of f and the size of B, and we have the following results:
(1) If $|B| \le 4$, then $AC^0(LR(f)) \subseteq AC^0(Mod_6)$: In Section 6, we actually prove that the composition of functions $f_n \circ \cdots \circ f_1$ with each $f_i : B \to B$ and $|B| \le 4$ is computable in uniform $AC^0(Mod_6)$. This method also classifies the computational strength of 1-BRN, 2-BRN, and 3-BRN.
(2) If f is associative and commutative, then $AC^0(LR(f)) \subseteq ACC$: Since f is associative and commutative, and its domain is finite, for each $a \in Dom(f)$ we may use modular counting according to the behavior of powers of a (with respect to f). (Note that the power of a is periodic and the length of its period $\le |Dom(f)|$.)
(3) If f is associative and $|B| < 60$, then $AC^0(LR(f)) \subseteq ACC$: This is based on a result of [4]. See Section 7.
(4) There is a nonassociative, commutative f with $|B| = 2$ such that $AC^0(DC(f)) = NC^1$: That is because $DC(|) = tree$.
(5) There is a nonassociative, commutative f with $|B| = 5$ such that $AC^0(LR(f)) = NC^1$: With $|B| = 5$ (say, $B = \{a, b, c, d, e\}$), one can construct a commutative multiplication table for f, such that $f(\cdot, a)$ performs a 5-cycle permutation $(a\,b\,c\,d\,e)$, and $f(\cdot, b)$ performs a 2-cycle $(a\,c)$:

$$\begin{array}{lllll}
f(a,a) = b, & f(a,b) = c, & f(a,c) = d, & f(a,d) = e, & f(a,e) = a, \\
f(b,a) = c, & f(b,b) = b, & f(b,c) = a, & f(b,d) = d, & f(b,e) = e, \\
f(c,a) = d, & f(c,b) = a, & f(c,c) = e, & f(c,d) = e, & f(c,e) = e, \\
f(d,a) = e, & f(d,b) = d, & f(d,c) = e, & f(d,d) = e, & f(d,e) = e, \\
f(e,a) = a, & f(e,b) = e, & f(e,c) = e, & f(e,d) = e, & f(e,e) = e.
\end{array}$$

Since one 5-cycle and one 2-cycle will generate S_5, we have $AC^0(LR(f)) = NC^1$.

Finally in Section 8 we consider the commutator operation $*_c$ on a subset of A_5 (twelve 5-cycles and one identity element). This operation $*_c$ is not associative. We then describe the computing power of $LR(*_c)$ and $DC(*_c)$, and explain how it is related to icosahedron.

2. Function algebras

In this section we define function algebras $A_0, A_0(k), N_0, N_0'$. Roughly speaking, a function algebra is the smallest class of functions containing some basic functions and closed under some schemes. Examples of schemes are composition, iteration, recursion with some limitation. The advantage of function algebraic approach is that it is not machine dependent. We will define the function algebras $A_0, A_0(k), T_0$ which characterizes uniform AC^0, $AC^0(Mod_k)$, TC^0 respectively. (For details see [6].)

In function algebras all functions have domain and codomain $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.

Definition 2.1. $zero(x) = 0$; $s_0(x) = 2x$; $s_1(x) = 2x+1$; $i_k^n(x_1, \ldots, x_n) = x_k$; $|x| = \lceil \log_2(x+1) \rceil$; $x\#y = 2^{|x|\cdot|y|}$; $(x)\, mod2 = x - 2 \cdot \lfloor x/2 \rfloor$; $Bit(i,x) = (\lfloor x/2^i \rfloor)\, mod2$; $|x|_2 = ||x||$; for $k \geq 2$, $|x|_{k+1} = ||x|_k|$.

Definition 2.2. Suppose that $h_0(n, \overrightarrow{x}), h_1(n, \overrightarrow{x}) \leq 1$. The function f is defined by CRN (*concatenation recursion on notation*) from g, h_0, h_1 if

$$\begin{aligned} f(0, \overrightarrow{x}) &= g(\overrightarrow{x}), \\ f(s_0(n), \overrightarrow{x}) &= s_{h_0(n,\overrightarrow{x})}(f(n, \overrightarrow{x})) \text{ for } n > 0, \\ f(s_1(n), \overrightarrow{x}) &= s_{h_1(n,\overrightarrow{x})}(f(n, \overrightarrow{x})). \end{aligned}$$

Definition 2.3. A_0 is the smallest class of functions containing the basic functions $zero, s_0, s_1, i_k^n, |x|, \#, Bit(i,x)$, and closed under composition and CRN.

In [9] Immerman developed the notion of first order definability which captures uniform circuits without involving sequential or alternating Turing machines. Because of the robustness of this class, people believe that this notion is the right notion of uniform AC^0. In [6] Clote proved that $A_0 = FO$, where FO is one version of uniform AC^0 defined by first order definability. (We will not use this result later.)

Definition 2.4. Let $\overrightarrow{x} = x_1, x_2, \ldots, x_m$ be a sequence of natural numbers, $|\overrightarrow{x}| = max(|x_1|, \ldots, |x_m|)$, and $||\overrightarrow{x}|| = max(||x_1||, \ldots, ||x_m||)$. A function f is *sharply bounded* (or *doubly sharply bounded*) if there is a polynomial p (or a constant c) such that $f(\overrightarrow{x}) \leq p(|\overrightarrow{x}|)$ (or $f(\overrightarrow{x}) \leq c||\overrightarrow{x}||$) for all $\overrightarrow{x}$.

Now we recall some useful results from [3], [7], [8], [11]. While checking uniformity in Sections 4,5, we will use Lemmas 2.5, 2.7,2.10,2.13,2.16, CRN, $Seg(x,i,j)$,

Sharply bounded quantifiers are of the forms $\exists x < |t|$, $\forall x < |t|$. Lemma 2.5 shows that A_0 is closed under sharply bounded quantification.

Lemma 2.5. *If $g, h \in A_0$ and f is defined by*

$$f(x) = \begin{cases} 1 & \text{if } \exists i \leq |g(x)|\, [h(i,x) = 0]; \\ 0 & \text{else,} \end{cases}$$

then $f \in A_0$.

Definition 2.6. The function f is defined from g, h by *sharply bounded μ-operator* if

$$f(x) = \begin{cases} i_0 & \text{if } i_0 \leq |g(x)| \land h(i_0, x) = 0 \land \forall i < i_0\, (h(i,x) \neq 0); \\ |g(x)| & \text{else.} \end{cases}$$

This is denoted by $f(x) = \mu i < |g(x)|\, [h(i,x) = 0]$.

Since in such case "$h(f(x), x) = 0$?" can be easily checked, we also call it sharply bounded search.

Lemma 2.7. *A_0 is closed under the sharply bounded μ-operator.*

Definition 2.8.

$$pad(x,y) = 2^{|y|} \cdot x;$$

$$x * y = pad(x,y) + y \text{ (Concatenation of } x, y);$$

$$Seg(x,i,j) = \sum_{k=i}^{j} 2^{k-i} Bit(k,x) \text{ for } i < j \text{ (Segment of } x \text{ from bit } i \text{ to bit } j).$$

Obviously $x * y$, $Seg(x,i,j)$ are computable in A_0.

Definition 2.9. $Maxindex(f,x) = \mu i \leq |x|\, \forall k \leq |x|\, (f(k) \leq f(i))$.

It is clear that $Maxindex$ searches the maximum of $f(i)$ for $1 \leq i \leq |x|$.

Lemma 2.10. *If $f \in A_0$, then $Maxindex(f,x)$ is in A_0.*

Proof. See [11] for a direct proof. □

Definition 2.11. F is definable from g, h_0, h_1 by k-BRN (*k-bounded recursion on notation*) for $k \in \mathbb{N}$ if

$$F(0, \overrightarrow{x}) = g(\overrightarrow{x}),$$

$$F(2n, \overrightarrow{x}) = h_0(n, \overrightarrow{x}, F(n, \overrightarrow{x})) \text{ if } n > 0,$$

$$F(2n+1, \overrightarrow{x}) = h_1(n, \overrightarrow{x}, F(n, \overrightarrow{x})),$$

and $0 \leq F(n, \overrightarrow{x}) \leq k$ for all $n, \overrightarrow{x}$.

Definition 2.12. The function f is defined from g, h_0, h_1, by weak k-BRN (*weak, k-bounded recursion on notation*) if $f(x, \overrightarrow{u}) = F(|x|, \overrightarrow{u})$ and $F(x, \overrightarrow{u})$ is definable from g, h_0, h_1 by k-BRN.

Note that the number of steps in iterated recursions of k-BRN and weak k-BRN are $|x| (= n)$ and $||x|| (= \log n)$ respectively.

Lemma 2.13. *A_0 is closed under weak k-BRN.*

We now use the following notation for extension of function algebras: Consider a function algebra A, a function f, and a formation rule $\mathcal{R}$, then $A(f)$ denote the function algebra which has basic functions of A and a new basic function f, and its formation rules are the same as A. Similarly $A(\mathcal{R})$ denote the function algebra which has basic functions of A, and its formation rules are the formation rules of A and a new formation rule $\mathcal{R}$.

Definition 2.14. $count(x)$ is the number of 1's in the binary expression of x, i.e.,

$$count(0) = 0,$$

$$count(s_0(x)) = count(x), \text{ provided } x > 0,$$

$$count(s_1(x)) = count(x) + 1.$$

And T_0 is the smallest class containing basic functions $zero$, s_0, s_1, i_k^n, $|x|$, $\#$, $Bit(i,x)$, $count$, and closed under composition and CRN. (That is, $T_0 = A_0(count)$.)

Definition 2.15. *Sharply bounded counting function* $sbcount(x,y)$ is defined as follows:

$$sbcount(x,y) = \begin{cases} count(x) & \text{if } x \leq |y|; \\ 0 & \text{else.} \end{cases}$$

$sbcount(x,y)$ means $count(x)$ for small x.

Lemma 2.16. $sbcount(x,y) \in A_0$.

Definition 2.17. The modular counting function is defined as

$$Mod_k(x) = \begin{cases} 1 & \text{if } k|count(x); \\ 0 & \text{else.} \end{cases}$$

Function algebra $A_0(k)$ is $A_0(Mod_k)$, i.e., the smallest class of functions containing the basic functions $zero, s_0, s_1, i_k^n, |x|, \#, Bit(i,x), Mod_k$, and closed under composition and CRN.

Note that $A_0(k)$ is the uniform $AC^0(Mod_k)$. For any two distinct primes p,q, $AC^0(Mod_{pq}) = AC^0(Mod_p)(Mod_q)$ and $AC^0(Mod_p), AC^0(Mod_q) \subset AC^0(Mod_{pq})$.

Definition 2.18. The function $tree(x)$ taking values $0,1$ is defined from the auxiliary functions $and(x), or(x)$ as follows:

$$\begin{aligned} and(0) &= 0, \\ and(1) &= 1, \\ and(2) &= 1, \\ and(3) &= 1, \\ and(s_0(s_0(x))) &= s_0(and(x)) \text{ if } x > 0, \\ and(s_0(s_1(x))) &= s_0(and(x)) \text{ if } x > 0, \\ and(s_1(s_0(x))) &= s_0(and(x)) \text{ if } x > 0, \\ and(s_1(s_1(x))) &= s_1(and(x)) \text{ if } x > 0, \end{aligned}$$

$$\begin{aligned} or(0) &= 0, \\ or(1) &= 1, \\ or(2) &= 1, \\ or(3) &= 1, \\ or(s_0(s_0(x))) &= s_0(or(x)) \text{ if } x > 0, \\ or(s_0(s_1(x))) &= s_1(or(x)) \text{ if } x > 0, \\ or(s_1(s_0(x))) &= s_1(or(x)) \text{ if } x > 0, \\ or(s_1(s_1(x))) &= s_1(or(x)) \text{ if } x > 0, \end{aligned}$$

$$tree(x) = \begin{cases} x - 2 \cdot \lfloor x/2 \rfloor & \text{if } x < 16, \\ tree(or(and(x))) & \text{else.} \end{cases}$$

The function $tree(x)$ actually does the following computation. Suppose that $x = \sum_{i=0}^{n-1} 2^i \cdot x_i$, $x_i \in \{0,1\}$ for $0 \leq i \leq n-1$. Let $4^m < n \leq 4^{m+1}$. Then we compute AND, OR alternatively on those 4^m bits $x_{4^m-1}, \ldots, x_0$. Consider that there are $2m+1$ levels. At level 0 we have $2^{2m}(=4^m)$ many bits: $L(0, 4^m-1) = x_{4^m-1}$, $L(0, 4^m-2) = x_{4^m-2}$, $\ldots$, $L(0,1) = x_1$, $L(0,0) = x_0$. At level 1 we define $L(1,j) = L(0, 2j+1) \wedge L(0, 2j)$, the conjunction of $L(0, 2j+1)$ and $L(0, 2j)$. Therefore we have 2^{2m-1} many bits at level 1. At level 2 we use OR: $L(2,j) = L(1, 2j+1) \vee L(1, 2j)$. In general, $L(2k+1, j) = L(2k, 2j+1) \wedge L(2k, 2j)$ and $L(2k, j) = L(2k-1, 2j+1) \vee L(2k-1, 2j)$. At level $2m$ there is only one bit, and we denote this by $tree(x)$.

For example, if $m = 1$, then $4 < |x| \leq 4^2$ and $tree(x) = (x_3 \wedge x_2) \vee (x_1 \wedge x_0)$. If $m = 2$, $4^2 < |x| \leq 4^3$, then first compute

$$\begin{aligned} y_3 &= (x_{15} \wedge x_{14}) \vee (x_{13} \wedge x_{12}), \\ y_2 &= (x_{11} \wedge x_{10}) \vee (x_9 \wedge x_8), \\ y_1 &= (x_7 \wedge x_6) \vee (x_5 \wedge x_4), \\ y_0 &= (x_3 \wedge x_2) \vee (x_1 \wedge x_0), \end{aligned}$$

and $tree(x) = (y_3 \wedge y_2) \vee (y_1 \wedge y_0)$.

Definition 2.19. N_0 is the smallest class of functions which contains the basic functions $zero, s_0, s_1, i_k^n, |x|, \#, Bit(i,x), tree$, and is closed under composition and CRN.

Definition 2.20. N_0' is the smallest class of functions containing the basic functions $zero, s_0, s_1, i_k^n, |x|, \#, Bit(i,x)$ and closed under composition, CRN, and k-BRN for any constant $k \geq 0$.

3. Expressibility

Remark 3.1. The concept "expressibility" is similar to Karp reduction, and actually the same as "projection" (developed by Valiant). To show that $tree$ is complete in NC^1, it suffices to show that for any Boolean function g in NC^1 there is an input sequence $\overrightarrow{b}$ (constructed from the original input x) of polynomial size such that $tree(\overrightarrow{b}) = g(x)$.

Definition 3.2. Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. f is called *m-tree expressible* if there exists an input sequence $G_{4^m-1}, G_{4^m-2}, \ldots, G_0 (= \overrightarrow{G})$ such that

$$f(x_1, \ldots, x_n) = tree(2^{4^m} + \sum_{i=0}^{4^m-1} 2^i \cdot G_i)$$

where the input sequence $G_{4^m-1}, G_{4^m-2}, \ldots, G_0$ is a sequence with elements from $\{0,1\} \cup \{x_j, \neg x_j : 1 \leq j \leq n\}$. Note that if $G_i = \neg x_j$, then the value of G_i is 0 if and only if x_j is assigned to 1. We call G_i as the i-th position of the input sequence $\overrightarrow{G}$. The length of $\overrightarrow{G}$ is 4^m.

We call x_j a positive atom, $\neg x_j$ a negative atom, and we identify $\neg\neg x$ with x. We also use the following notation

$$f(x_1, \ldots, x_n) = (\bigvee \bigwedge)^m (G_{4^m-1}, G_{4^m-2}, \ldots, G_0)$$

to denote that f is *m-tree* expressible with input sequence $G_{4^m-1}, G_{4^m-2}, \ldots, G_0$. For example,

$$\begin{aligned}(\bigvee\bigwedge)^1(x_1,x_2,x_3,x_4) &= \bigvee(\bigwedge(x_1,x_2,x_3,x_4)) \\ &= \bigvee((x_1\wedge x_2),(x_3\wedge x_4)) \\ &= ((x_1\wedge x_2)\vee(x_3\wedge x_4)).\end{aligned}$$

If f is *m-tree* expressible with input sequence $\overrightarrow{G}$, then we may denote the input sequence $\overrightarrow{G}$ by $sqtree[f]$ (sequence of *tree* of f). Note that $\overrightarrow{G}$ is not unique.

The following remark shows that we may describe $\overrightarrow{G}$ by some functions h_1, h_2, h_3.

Remark 3.3. Suppose that $f : \{0,1\}^n \to \{0,1\}$ is *m-tree* expressible, then there exist h_1, h_2, h_3 with domain $\{0,1\}^{2m}$ such that these functions characterize the behavior of the input sequence $G_{4^m-1}, G_{4^m-2}, \ldots, G_0 (= sqtree[f])$. For example,

$$\begin{aligned} h_1(l) &= \begin{cases} 0 & \text{if the } l\text{-th position of } sqtree[f] \text{ is a constant } 0; \\ 1 & \text{if the } l\text{-th position of } sqtree[f] \text{ is a constant } 1; \\ 2 & \text{else.} \end{cases} \\ h_2(l) &= \begin{cases} \neg & \text{if the } l\text{-th position of } sqtree[f] \text{ is a negative atom;} \\ \neg\neg & \text{if the } l\text{-th position of } sqtree[f] \text{ is a positive atom;} \\ 0 & \text{else.} \end{cases} \\ h_3(l) &= \begin{cases} j & \text{if the } l\text{-th position of } sqtree[f] \text{ is either a positive atom } x_j \\ & \text{or a negative atom } \neg x_j; \\ -1 & \text{else.} \end{cases} \end{aligned}$$

Then h_1, h_2, h_3 express the constant, sign, index condition of $sqtree[f]$ respectively. When n is fixed, h_1, h_2, h_3 are obviously in A_0. Later we will consider the case $f : \{0,1\}^* \to \{0,1\}$. In such case we expect that $m(x) \le p(|x|)$ for some polynomial p and $h_1, h_2, h_3 \in A_0$. Here we abuse the definition of A_0 by allowing $\neg, \neg\neg, -1$ as outputs.

Remark 3.4. For $l = a_{m-1}a_{m-2}\ldots a_0 \in \{0,1\}^m$, we identify the binary sequence l with the number $n = \sum_{j=0}^{m-1} a_j \cdot 2^j$. When we say "the l-th position" we actually mean "the n-th position where $n = \sum_{j=0}^{m-1} a_j \cdot 2^j$ and $l = a_{m-1}a_{m-2}\ldots a_0 \in \{0,1\}^m$." Also "$n = l$" means "$n = \sum_{j=0}^{m-1} a_j \cdot 2^j$ and $l = a_{m-1}a_{m-2}\ldots a_0$."

Later on, we will express function $f : \{0,1\}^* \to \{0,1\}$ by *tree* and investigate the constant function, sign function, and index function for f. Note that these functions are not obviously in A_0.

We need to build up some basic tools.

Lemma 3.5. *$x_1 \oplus x_2$ is 1-tree expressible.*

Proof. $x_1 \oplus x_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) = (\bigvee\bigwedge)^1(x_1, \neg x_2, \neg x_1, x_2)$. $\square$

Convention. We use the following abbreviation for input sequences: Let $0^{(k)} = \overbrace{0,0,\ldots,0}^{k \text{ times}}$, $1^{(k)} = \overbrace{1,1,\ldots,1}^{k \text{ times}}$, $0^{[4^m]} = (\bigvee\bigwedge)^m(0^{(4^m)})$, and $1^{[4^m]} = (\bigvee\bigwedge)^m(1^{(4^m)})$.

Lemma 3.6. *If $f(x_1,\ldots,x_n)$ is m-tree expressible, then $\neg f(x_1,\ldots,x_n)$ is $(m+1)$-tree expressible.*

Proof. By De Morgan's law. $\square$

Lemma 3.7. *If f is m-tree expressible and $l > m$, then f is l-tree expressible.*

Proof. Induction on $(l - m)$. □

Lemma 3.8. *If $f : \{0,1\}^l \to \{0,1\}$ is m-tree expressible and $g_i : \{0,1\}^l \to \{0,1\}$ are all k-tree expressible for $1 \leq i \leq l$, then $f(g_1, \ldots, g_l)$ is $(m + k + 1)$-tree expressible.*

Proof. Since $g_1, \ldots, g_l, \neg g_1, \ldots, \neg g_l$ are all $(k+1)$-*tree* expressible, we may simply substitute input sequences $sqtree[g_1], \ldots, sqtree[g_l], sqtree[\neg g_1], \ldots, sqtree[\neg g_l]$ (each with size 4^{k+1}) into $sqtree[f]$. Then $f(g_1, \ldots, g_l)$ is $(m+k+1)$-*tree* expressible. □

Lemma 3.9. *If f, g are m-tree expressible, then $(f \wedge g), (f \vee g)$ are $(m+1)$-tree expressible.*

Proof. $(f \wedge g) = (f \wedge g) \vee (0^{[4^m]} \wedge 0^{[4^m]})$ and $(f \vee g) = (f \wedge 1^{[4^m]}) \vee (g \wedge 1^{[4^m]})$. □

Definition 3.10. Function f is called A_0*-tree expressible* if there exist a polynomial $p(x)$, a constant c, constant function $h_1 \in A_0$, sign function $h_2 \in A_0$, and index function $h_3 \in A_0$ such that the following conditions hold:

(1) $|f(x)| \leq p(|x|)$.
(2) For any $j \leq p(|x|)$ and x,

$$Bit(j, f(x)) = tree(2^{4^{c|x|_2}} + \sum_{i=0}^{4^{c|x|_2}-1} 2^i \cdot G(i,j,x)),$$

where the input sequence $G(i,j,x)$ has range $\subseteq \{0,1\}$, and it is characterized by functions $h_1(i,j,2^{4^{c|x|_2}})$, $h_2(i,j,2^{4^{c|x|_2}})$, $h_3(i,j,2^{4^{c|x|_2}})$.

Similarly we can define N_0-*tree* expressibility, etc.

Remark 3.11. Intuitively "f is A_0-*tree* expressible" means: to compute any bit of $f(x)$, it suffices to construct an input sequence, whose range is $\{\sim Bit(i,x) : 0 \leq i \leq |x|\} \cup \{0,1\}$, and then we substitute this sequence into *tree*. Note that the construction of the input sequence should be computable in A_0. We give this definition without rigorously restricting the form of h_1, h_2, h_3 for flexibility. (Remark 3.3 shows how we may define h_1,h_2, h_3.)

Remark 3.12. The denotation of input sequence $G(i,j,x)$ does not well characterize the way we use information from x. For h_1, h_2, h_3 we only need to know $|x|_2$ (we use $2^{4^{c|x|_2}}$ to preserve the size of input.)

Theorem 3.13. *If $f(x)$ is N_0-tree expressible, then $f \in N_0$.*

Proof. By CRN,

$$g(j,x) = 2^{4^{c|x|_2}} + \sum_{i=0}^{4^{c|x|_2}-1} 2^i \cdot G(i,j,x) \in N_0.$$

So $tree(g(j,x)) \in N_0$. Then by CRN we have

$$f(x) = \sum_{j=0}^{p(|x|)} 2^j \cdot tree(g(j,x)) \in N_0.$$

□

4. k-BRN IS A_0-*tree* EXPRESSIBLE

In this section we show how to convert k-BRN to *tree* by *tree* expressibility. Since any function $f : \{0, \ldots, k\} \to \{0, \ldots, k\}$ can be represented by a $(k+1) \times (k+1)$ matrix, the bounded recurrence k-BRN can be simulated by a multiple product of the corresponding Boolean matrices. Therefore to simulate k-BRN by *tree* it suffices to show that such a product is computable in N_0. Note that Boolean multiplication of matrices is associative (see Lemma 4.3).

Definition 4.1. A is called an $m \times n$ Boolean matrix if $A : \{0, 1, \ldots, m-1\} \times \{0, 1, \ldots, n-1\} \to \{0, 1\}$. We denote A by $(a_{ij})_{m \times n}$, where $a_{ij} = A(i, j) \in \{0, 1\}$. $\mathcal{B}_{m \times n} \stackrel{def}{=} \{A : A \text{ is an } m \times n \text{ Boolean matrix}\}$. $I_n \in \mathcal{B}_{n \times n}$, $I_n(i, j) = 1$ iff $i = j$.

Definition 4.2. Let $C = (c_{ij})_{m \times n}, D = (d_{ij})_{n \times l}$, then the Boolean multiplication of C and D is defined as follows:

$$C \circ_{\mathcal{B}} D = (\bigvee_{k=0}^{n-1} c_{ik} \wedge d_{kj})_{m \times l}.$$

In this section, we will just write CD instead of $C \circ_{\mathcal{B}} D$.

The following lemma is obvious.

Lemma 4.3. *Boolean multiplication is associative.*

Let $A(t)$ be an $m_t \times n_t$ Boolean matrix for $1 \leq t \leq s$, and $n_{t-1} = m_t$ for $2 \leq t \leq s$. Then $\prod_{t=1}^{s} A(t)$ makes sense. If $\prod_{t=1}^{s} A(t)(i, j) = 1$, then there exists a path function g such that $g(0) = i$, $g(s) = j$, $1 \leq g(t) \leq n_t$, $A(t)(g(t-1), g(t)) = 1$ for $1 \leq t \leq s$, i.e., there is a path from i to j.

Now we work on the case 7-BRN. The other cases are similar to this.

Theorem 4.4. *If $A(\overrightarrow{x}, t) \in A_0$ is a function with range $\mathcal{B}_{8\times 8}$, then the Boolean product $[\prod_{t=0}^{|y|-1} A(\overrightarrow{x}, t)](i, j)$ is A_0-tree expressible for $0 \leq i, j \leq 7$.*

Proof. We may assume that $|y| = 2^m$ for some m. If not, we may define

$$A'(\overrightarrow{x}, t) = \begin{cases} A(\overrightarrow{x}, t) & \text{if } t < |y|; \\ I_8 & \text{else.} \end{cases}$$

Then

$$\prod_{t=0}^{|y|-1} A(\overrightarrow{x}, t) = \prod_{t=0}^{2^{||y||}-1} A'(\overrightarrow{x}, t).$$

Since Boolean multiplication is associative, we will use $DC(\circ_{\mathcal{B}})$ to compute the product.

Claim 1: $(\bigvee_{l=0}^{7} a_{il} \wedge b_{lj})_{8\times 8}$ is 3-*tree* expressible.

Proof of Claim 1.

$$\begin{aligned} \bigvee_{l=0}^{7} a_{il} \wedge b_{lj} \quad &= \quad (\bigvee \bigwedge)^1(a_{i0}, b_{0j}, a_{i1}, b_{1j}) \vee (\bigvee \bigwedge)^1(a_{i2}, b_{2j}, a_{i3}, b_{3j}) \\ &\quad\ \vee (\bigvee \bigwedge)^1(a_{i4}, b_{4j}, a_{i5}, b_{5j}) \vee (\bigvee \bigwedge)^1(a_{i6}, b_{6j}, a_{i7}, b_{7j}) \\ &= (\bigvee \bigwedge)^3(a_{i0}, b_{0j}, a_{i1}, b_{1j}, 1^{(4)}, a_{i2}, b_{2j}, a_{i3}, b_{3j}, 1^{(4)}, 1^{(4^2)}, \\ &\quad\ a_{i4}, b_{4j}, a_{i5}, b_{5j}, 1^{(4)}, a_{i6}, b_{6j}, a_{i7}, b_{7j}, 1^{(4)}, 1^{(4^2)}). \end{aligned}$$

We call this input sequence $sqtree[\circ_{\mathcal{B}}]$. End of Claim 1. $\square$

Now we define the Boolean products level by level. Let $S(0;\overrightarrow{x},t) = A(\overrightarrow{x},t) \in \mathcal{B}_{8\times 8}$ for $0 \le t < 2^m$. Define 8×8 Boolean matrices $S(k+1;\overrightarrow{x},t) = S(k;\overrightarrow{x},2t) \circ_{\mathcal{B}} S(k;\overrightarrow{x},2t+1)$ for $0 \le t < 2^{m-k-1}$, $0 \le k \le m$.

The following claim is proved by induction.

Claim 2: $S(k;\overrightarrow{x},t)(i,j)$ is $(3k)$-*tree* expressible for $0 \le i,j \le 7$, $0 \le k \le m$.

Since $S(k;\overrightarrow{x},t)(i,j)$ is $3k$-*tree* expressible, we define

$$G(S(k;\overrightarrow{x},t)(i,j);l) \stackrel{def}{=} \text{the } l\text{-th position of } sqtree[S(k;\overrightarrow{x},t)(i,j)]$$

for $0 \le l \le 4^{3k}-1$.

In order to handle the behavior of $DC(\circ_{\mathcal{B}})$, we define

$$\begin{array}{rcl} Con(\circ_{\mathcal{B}};\cdot) & : & \{0,1\}^6 \to \{0,1\}, \\ Con(\circ_{\mathcal{B}};l) & = & \begin{cases} 0 & \text{if the } l\text{-th position of } sqtree[\circ_{\mathcal{B}}] \text{ is an atom;} \\ 1 & \text{else.} \end{cases} \\ Ind(\circ_{\mathcal{B}};\cdot) & : & \{0,1\}^6 \to \{0,1\}^4, \\ Ind(\circ_{\mathcal{B}};l) & = & \begin{cases} 0000 & \text{if the } l\text{-th position of } sqtree[\circ_{\mathcal{B}}] \text{ is a constant;} \\ 0s_1s_2s_3 & \text{if the } l\text{-th position of } sqtree[\circ_{\mathcal{B}}] \text{ is } a_{is}, \\ & s = s_1\cdot 2^2 + s_2\cdot 2 + s_3, \text{ and } 0 \le s_1,s_2,s_3 \le 1; \\ 1s_1s_2s_3 & \text{if the } l\text{-th position of } sqtree[\circ_{\mathcal{B}}] \text{ is } b_{sj}, \\ & s = 2^3 + s_1\cdot 2^2 + s_2\cdot 2 + s_3, \text{ and } 0 \le s_1,s_2,s_3 \le 1. \end{cases} \end{array}$$

Note that in this case the sign function is useless and constant 0 does not appear.

Claim 3: The function

$$Con(S(k;\overrightarrow{x},t)(i,j);l) \stackrel{def}{=} \begin{cases} 0 & \text{if the } l\text{-th position of } sqtree[S(k;\overrightarrow{x},t)] \text{ is an atom} \\ 1 & \text{else} \end{cases}$$

is in A_0.

Proof of Claim 3. Given $l \in [0,4^{3k})$, it can be expressed by $\gamma_{k-1} * \gamma_{k-2} * \cdots * \gamma_0$, where $\gamma_v \in \{0,1\}^6$ for $0 \le v < k$. According to $Con(\circ_{\mathcal{B}};\cdot)$, if $Con(\circ_{\mathcal{B}};\gamma_{k-1}) * Con(\circ_{\mathcal{B}};\gamma_{k-2}) * \cdots * Con(\circ_{\mathcal{B}};\gamma_0)$ is zero, then $G(S(k;\overrightarrow{x},t)(i,j);l)$ is an atom. Else $G(S(k;\overrightarrow{x},t)(i,j);l)$ is constant 1. It is clear that to determine whether "$Con(\circ_{\mathcal{B}};\gamma_{k-1}) * Con(\circ_{\mathcal{B}};\gamma_{k-2}) * \cdots * Con(\circ_{\mathcal{B}};\gamma_0) = 0$" is computable in A_0. End of Claim 3. $\square$

Claim 4: The function

$$Ind(S(k;\overrightarrow{x},t)(i,j);l) \stackrel{def}{=} \begin{cases} (\tilde{t},\tilde{\imath},\tilde{\jmath}) & \text{if the } l\text{-th position of } sqtree[S(k;\overrightarrow{x},t)(i,j)] \\ & \text{is } A(\overrightarrow{x},\tilde{t})(\tilde{\imath},\tilde{\jmath}) \\ (0,0,0) & \text{else} \end{cases}$$

is in A_0 for $0 \le i,j \le 7$.

Proof of Claim 4. Given $l \in [0,4^{3k})$, $l = \gamma_{k-1} * \cdots * \gamma_0, \gamma_v \in \{0,2\}^6$ for $0 \le v < k$. Since $S(k;\overrightarrow{x},t) = S(k-1;\overrightarrow{x},2t) \circ_{\mathcal{B}} S(k-1;\overrightarrow{x},2t+1)$, $S(k;\overrightarrow{x},t)$ can be seen as $sqtree[\circ_{\mathcal{B}}]$ with elements from $1^{[4^{3(k-1)}]}, S(k-1;\overrightarrow{x},2t)(i,s), S(k-1;\overrightarrow{x},2t+1)(s,j)$ for some s with $0 \le s \le 7$. Note that all of these are $3(k-1)$-*tree* expressible. For simplicity, let $G_l = G(S(k;\overrightarrow{x},t)(i,j);l)$. We only need to consider the case $Con(S(k;\overrightarrow{x},t)(i,j);l) = 0$ (else $G_l = 1$).

Now assume that $Con(S(k; \overrightarrow{x}, t)(i,j); l) = 0$. First we consider $k > 1$. According to $sqtree[\circ_{\mathcal{B}}]$, if $Bit(3, Ind(\circ_{\mathcal{B}}; \gamma_{k-1})) = 0$, then G_l comes from a_{is} where $s = Ind(\circ_{\mathcal{B}}; \gamma_{k-1})$. In this case

$$G(S(k; \overrightarrow{x}, t)(i,j); l) = G(S(k-1; \overrightarrow{x}, 2t)(i,s); l')$$

where $l' = \gamma_{k-2} * \cdots * \gamma_0$.
If $Bit(3, Ind(\circ_{\mathcal{B}}; \gamma_{k-1})) = 1$, then G_l comes from b_{sj}, where $s = Ind(\circ_{\mathcal{B}}; \gamma_{k-1}) - 2^3$. In this case

$$G(S(k; \overrightarrow{x}, t)(i,j); l) = G(S(k-1; \overrightarrow{x}, 2t+1)(s,j); l')$$

where $l' = \gamma_{k-2} * \cdots * \gamma_0$.

The case $k = 1$ is similar. Let $l = \gamma_0 \in \{0,1\}^6$.
If $Bit(3, Ind(\circ_{\mathcal{B}}; \gamma_0)) = 0$, then

$$G(S(1; \overrightarrow{x}, t)(i,j); l) = S(0; \overrightarrow{x}, 2t)(i,s)$$

where $s = Ind(\circ_{\mathcal{B}}; \gamma_0)$.
If $Bit(3, Ind(\circ_{\mathcal{B}}; \gamma_0)) = 1$, then

$$G(S(1; \overrightarrow{x}, t)(i,j); l) = S(0; \overrightarrow{x}, 2t+1)(s,j)$$

where $s = Ind(\circ_{\mathcal{B}}; \gamma_0) - 2^3$.

By induction, $\tilde{t} = t * Bit(3, Ind(\circ_{\mathcal{B}}; \gamma_{k-1})) * \cdots * Bit(3, Ind(\circ_{\mathcal{B}}; \gamma_0))$. This is computable in A_0.

To determine $\tilde{i}, \tilde{j}$, we define the following A_0 function:

$$f : [0, 2^k] \to \{0,1,2,3,4,5,6,7\}$$

and $f(0) = i, f(2^k) = j$.

Let $\alpha_v = Ind(\circ_{\mathcal{B}}; \gamma_v) - Bit(3, Ind(\circ_{\mathcal{B}}; \gamma_v)) \cdot 2^3$ for $0 \le v < k$, then $0 \le \alpha_v \le 7$.

Define $f((2u+1) \cdot 2^v) = \alpha_v$ for $0 < (2u+1) \cdot 2^v < 2^k$. Then f is totally defined on $[0, 2^k]$ and $f \in A_0$.

Let $\tilde{t} = t * \bar{t}$, i.e., $\bar{t} = Bit(3, Ind(\circ_{\mathcal{B}}; \gamma_{k-1})) * \cdots * Bit(3, Ind(\circ_{\mathcal{B}}; \gamma_0))$.

The idea we design f is to assign $f(2^{k-1}) = s$. If $Bit(3, Ind(\circ_{\mathcal{B}}; \gamma_{k-1})) = 0$, then $(f(0), f(2^{k-1})) = (i,s)$. If $Bit(3, Ind(\circ_{\mathcal{B}}; \gamma_{k-1})) = 1$, then $(f(2^{k-1}), f(2^k)) = (s,j)$. In both cases $f(2^{k-1}) = s$ and the difference shrinks from 2^k to 2^{k-1}. Therefore it suffices to find the right interval $[w, w+1]$ such that $f(w) = \tilde{i}$ and $f(w+1) = \tilde{j}$. By induction $\tilde{i} = f(\bar{t}), \tilde{j} = f(\bar{t}+1)$ are computable in A_0. End of Claim 4. □

Now $S(m; \overrightarrow{x}, 0)(i,j) = [\prod_{t=0}^{|y|-1} A(\overrightarrow{x}, t)](i,j)$ is A_0-*tree* expressible since the constant function and index function for $S(k; \overrightarrow{x}, t)(i,j)$ are in A_0. □

Corollary 4.5. *If $A(\overrightarrow{x}, t) \in N_0$ is a function with range $\mathcal{B}_{8\times 8}$, then the Boolean product $[\prod_{t=0}^{|y|-1} A(\overrightarrow{x}, t)](i,j)$ is $A_0(tree)$-tree expressible for $0 \le i, j \le 7$.*

Proof. Same as Theorem 4.4 except that now all functions are in N_0. □

Theorem 4.6. *If $f : \mathbb{N}^m \to \mathbb{N}$ is N_0-tree expressible, then $f \in N_0$.*

Proof. Similar to Theorem 3.13. □

Theorem 4.7. *If $f \in A_0(7\text{-}BRN)$, then $f \in N_0$.*

Proof. By Theorems 4.4 and 4.6, 7-BRN can be translated into N_0 computation. Hence by induction and Theorem 4.6 any $f \in A_0$(7-BRN) is computable in N_0. □

By showing that k-BRN for $k = 2^m - 1$ are A_0-*tree* expressible, this implies $N_0' \subseteq N_0$.

Corollary 4.8. $N_0' \subseteq N_0$.

Now we can get more from above technique. Consider the Boolean product of m many Boolean matrices, each with size $s \times s$. Then the length of its input sequence is $4^{(\log s)\cdot m}$. Assume that the input x has $|x| = n$. Then we can set $s = O(\log n)$ and $m = O(\log n/(\log\log n))$ and we still have the input sequence with a polynomial size. Let $\circ_{\log n}$ be the corresponding Boolean product operator, then it can be easily checked that $Con(\circ_{\log n};\cdot)$ and $Ind(\circ_{\log n};\cdot)$ are in A_0. Furthermore, what we have checked in the claims of Theorem 4.4 still works in this case. We then have the following result.

Theorem 4.9. *If* $A(\overrightarrow{x}, t) \in A_0$ *is a function with range* $\mathcal{B}_{\log n \times \log n}$ *with* $|\overrightarrow{x}| = n$ *and* $z = (\log n/\log\log n)$, *then the Boolean product* $[\prod_{t=0}^{z} A(\overrightarrow{x}, t)](i,j)$ *is* A_0-*tree expressible for* $0 \le i, j \le \log n$.

Proof. *(Sketch)* Assume that $\log n = 2^r$. We define the Boolean products level by level in the same way. We have:

(1) $(\bigvee_{l=0}^{\log n} a_{il} \wedge b_{lj})_{\log n \times \log n}$ is $(\log\log n)$-*tree* expressible.

(2) $S(k; \overrightarrow{x}, t)(i,j)$ is $(k \cdot (\log\log n))$-*tree* expressible for $0 \le i, j < \log n$, $0 \le k \le (\log n/\log\log n)$.

What is different from the previous case is that $\circ_{\log n}$ is not a finite function. Therefore we need to verify that $Con(\circ_{\log n}, \cdot)$, $Ind(\circ_{\log n}, \cdot)$ are computable in A_0. We design the input sequence inductively:

$$\bigvee_{l=1}^{2^{r+1}} a_{il} \wedge b_{lj} = \bigvee \bigwedge (\bigvee_{l=1}^{2^r} a_{il} \wedge b_{lj}, 1^{[4^r]}, \bigvee_{l=2^r+1}^{2^{r+1}} a_{il} \wedge b_{lj}, 1^{[4^r]}).$$

Then $Con(\circ_{\log n}, \cdot)$, $Ind(\circ_{\log n}, \cdot)$ can be defined similarly. According to this construction, $Con(\circ_{\log n}, \cdot)$, $Ind(\circ_{\log n}, \cdot)$ are obviously computable in A_0.

Then one can easily run through the proofs of Claims 3,4. Note that the auxiliary function f is also computable in A_0. □

Since the Boolean product is associative, we can even multiply $(\log n)^k$ many Boolean matrices with size $p(\log n)$ in $k+1$ levels (each time with $(\log n/\log\log n)$ many matrices). Finally we have:

Theorem 4.10. *For any polynomials p, q, the product of $p(\log n)$ many Boolean matrices with size $q(\log n)$ is computable in N_0, where $n = |x|$ is the input size.*

Remark 4.11. While analyzing the uniformity of an input sequence, we can apply multiple products and powering. That is because multiple products and powering with sharply bounded values are computable in A_0. (For details see [11].)

Remark 4.12. The tree constructions in Theorems 4.4, 4.10 are monotone, i.e., no negation atoms are used in the construction. It seems possible to apply Theorem 4.10 for a monotone uniform construction of the majority gate in uniform NC^1. The known monotone construction of the majority gate is given in [12] by probabilistic method. But the uniform construction is either monotone but quite complicated (in [1]) or nonmonotone (in [5] or [10]).

5. Permutation branching program for *tree*

In this section first we introduce the concept of (permutation) branching programs. Then we prove that *tree* is computable in $A_0(k\text{-BRN})$ for $k \geq 4$. We follow the setting in [2]: For product in S_n we use the convention "from left to right," i.e., if $\sigma : x \mapsto y$ and $\tau : y \mapsto z$, then $\sigma\tau : x \mapsto z$. Let e be the identity element in S_n. We use $\sigma(i)$ to denote the image of i for $\sigma \in S_n$ and $1 \leq i \leq n$. Sometimes we use $\sigma \cdot \tau$ to denote the product $\sigma\tau$ in S_n.

Definition 5.1. Let $[w] = \{1, \ldots, w\}$, $\overrightarrow{x} = \langle x_0, \ldots, x_{n-1}\rangle, x_i \in \{0,1\}$ for $0 \leq i \leq n-1$. We abuse the notation $\overrightarrow{x}(i) = x_i$ and $\overrightarrow{x}(i,j) = \langle x_i, x_{i+1}, \ldots, x_j\rangle$ for $i < j$. An instruction is a triple $\langle j, f, g\rangle$ in which $f, g \in S_w$. The meaning of the instruction $\langle j, f, g\rangle$ is "evaluate to f if $Bit(j, \overrightarrow{x}) \stackrel{def}{=} x_j = 0$, else evaluate to g."

A width-w branching program (a w-BP) P of length l is a sequence $\langle j(i), f(i), g(i)\rangle$, for $1 \leq i \leq l$, such that $j : \{1, \ldots, l\} \to \{0, \ldots, n-1\}$, and $f, g : \{1, \ldots, l\} \to S_w$. Given $\overrightarrow{x} = \langle x_0, \ldots, x_{n-1}\rangle$, we define

$$h_{\overrightarrow{x}}(i) = \begin{cases} f(i) & \text{if } \overrightarrow{x}(j(i)) = 0; \\ g(i) & \text{else.} \end{cases}$$

Then the branching program P yields the function

$$P(\overrightarrow{x}) = \prod_{i=1}^{l} h_{\overrightarrow{x}}(i) \in S_w.$$

We use $P = P(j, f, g, l)$ or $P = P(j, f, g)$ to denote the corresponding j, f, g, l.

Actually the branching program we just defined is called permutation branching program. (For simplicity we abuse the term.) The general case is that f, g are functions, which may not be permutations. However, we shall see that they have the same computing power later.

Given $P = P(j, f, g, l)$, and $\sigma \in S_w$, we define $P^\sigma = \sigma P \sigma^{-1} = \langle j, \sigma f \sigma^{-1}, \sigma g \sigma^{-1}\rangle$, i.e., the i-th instruction is $\langle j(i), \sigma f(i)\sigma^{-1}, \sigma g(i)\sigma^{-1}\rangle$ for $1 \leq i \leq l$.

The concatenation of two branching programs can be naturally defined as follows: Given two branching program $P_1(j_1, f_1, g_1, l_1), P_2(j_2, f_2, g_2, l_2)$, we have a length $(l_1 + l_2)$ branching program

$$\begin{aligned} P_1 * P_2 \quad = \quad & \langle\langle j_1(1), f_1(1), g_1(1)\rangle, \ldots, \langle j_1(l_1), f_1(l_1), g_1(l_1)\rangle, \\ & \langle j_2(1), f_2(1), g_2(1)\rangle, \ldots, \langle j_2(l_2), f_2(l_2), g_2(l_2)\rangle\rangle. \end{aligned}$$

Definition 5.2. Consider $\alpha \in S_w$, $\alpha \neq e$, and a Boolean formula $c = c(\overrightarrow{x})$, we say that branching program P computes c by α iff

$$P(\overrightarrow{x}) = \begin{cases} \alpha & \text{if } c(\overrightarrow{x}) = 1; \\ e & \text{if } c(\overrightarrow{x}) = 0. \end{cases}$$

We denote this by $c \stackrel{\alpha}{\Leftrightarrow} P$.

Let $P = P(j, f, g, l)$, define

$P_{t(\alpha)} = \langle\langle j(1), f(1), g(1)\rangle, \ldots, \langle j(l-1), f(l-1), g(l-1)\rangle, \langle j(l), f(l)\alpha, g(l)\alpha\rangle\rangle.$

Note that $P_{t(\alpha)}(\overrightarrow{x}) = P(\overrightarrow{x}) \cdot \alpha$. (This means "add a tail α to P.")

Lemma 5.3. *If P computes c by α, then $P_{t(\alpha^{-1})}$ computes $\neg c$ by α^{-1}.*

Proof. Since $c(\overrightarrow{x}) = 1 \leftrightarrow \neg c(\overrightarrow{x}) = 0$,

$$P_{t(\alpha^{-1})}(\overrightarrow{x}) = P(\overrightarrow{x}) \cdot \alpha^{-1} = \begin{cases} \alpha \cdot \alpha^{-1} = e & \text{if } \neg c(\overrightarrow{x}) = 0; \\ e \cdot \alpha^{-1} = \alpha^{-1} & \text{else.} \end{cases}$$

□

Lemma 5.4. *Let P be a w-BP, $\alpha, \beta \in S_w \backslash \{e\}$, then*

(1) $(P_{t(\alpha)})^{\beta} = (P^{\beta})_{t(\beta\alpha\beta^{-1})}$,
(2) $(P^{\alpha})^{\beta} = P^{\beta\alpha}$,
(3) $(P_1 * P_2)^{\alpha} = P_1^{\alpha} * P_2^{\alpha}$,
(4) $(P_1 * P_2)_{t(\alpha)} = P_1 * (P_2)_{t(\alpha)}$.

Proof. By definition. □

In the rest of this section we will focus on branching programs with width$= 5$. To deal with this we need to name some elements in S_5.

Let $\sigma_1 = (1\,2\,3\,4\,5)$, $\sigma_2 = (1\,3\,5\,4\,2)$, then

$$\theta \stackrel{def}{=} \sigma_1 \sigma_2 \sigma_1^{-1} \sigma_2^{-1} = (1\,3\,2\,5\,4).$$

Define τ,γ_1,γ_2, δ_1,δ_2 which satisfy the following equalities: $\tau\theta^{-1}\tau^{-1} = \theta$, $\gamma_1\theta\gamma_1^{-1} = \sigma_1$, $\gamma_2\theta\gamma_2^{-1} = \sigma_2$, $\delta_1\theta\delta_1^{-1} = \sigma_1^{-1}$, $\delta_2\theta\delta_2^{-1} = \sigma_2^{-1}$. (There are more than one τ satisfying above condition. Anyway we just choose one and name it τ. In the same way we choose γ_1,γ_2,δ_1,δ_2. Note that the existence of θ plays a key role in [2].)

Remark 5.5. Note that $\sigma_1, \sigma_2, \theta \in A_5$. We may choose $\tau, \gamma_1, \gamma_2, \delta_1, \delta_2 \in A_5$:

$$\begin{aligned} \tau &= (2\,5)(3\,4), \\ \gamma_1 &= (2\,3)(4\,5), \\ \gamma_2 &= (2\,4\,5), \\ \delta_1 &= (3\,5)(2\,4), \\ \delta_2 &= (2\,3\,4). \end{aligned}$$

Since *tree* is complete in NC^1, this implies that "the word problem for A_5 is complete in NC^1."

Lemma 5.6. *If $c_1 \stackrel{\theta}{\leftrightarrow} P_1$, $c_2 \stackrel{\theta}{\leftrightarrow} P_2$, then*

$$c_1 \wedge c_2 \stackrel{\theta}{\leftrightarrow} P_1^{\gamma_1} * P_2^{\gamma_2} * P_1^{\delta_1} * P_2^{\delta_2}.$$

Proof. Let $P' = P_1^{\gamma_1} * P_2^{\gamma_2} * P_1^{\delta_1} * P_2^{\delta_2}$. If $c_1(\overrightarrow{x}) = 1$ and $c_2(\overrightarrow{x}) = 1$, then

$$P'(\overrightarrow{x}) = (\gamma_1\theta\gamma_1^{-1})(\gamma_2\theta\gamma_2^{-1})(\delta_1\theta\delta_1^{-1})(\delta_2\theta\delta_2^{-1}) = \sigma_1\sigma_2\sigma_1^{-1}\sigma_2^{-1} = \theta.$$

Otherwise, $P'(\overrightarrow{x}) = e$. □

Lemma 5.6 is the key part of Theorem 1 in [2], Section 3.

Example 5.7. Let $c_1 = x_i$, then $P = \langle i, \theta, e \rangle$ and $c_1 \stackrel{\theta}{\leftrightarrow} P$.

Example 5.8. Let $c_1 = x_i$ and $c_2 = x_j$, then

$$\begin{aligned} c_1 &\stackrel{\theta}{\leftrightarrow} P = \langle i, \theta, e \rangle, \\ c_2 &\stackrel{\theta}{\leftrightarrow} Q = \langle j, \theta, e \rangle. \end{aligned}$$

By Lemma 5.6, $c_1 \wedge c_2 \stackrel{\theta}{\leftrightarrow} P^{\gamma_1} * Q^{\gamma_2} * P^{\delta_1} * Q^{\delta_2}$.

Example 5.9. Let $c_0 \overset{\theta}{\leftrightarrow} P_0$, $c_1 \overset{\theta}{\leftrightarrow} P_1$, $c_2 \overset{\theta}{\leftrightarrow} P_2$, and $c_3 \overset{\theta}{\leftrightarrow} P_3$. If $c' = (c_0 \wedge c_1) \vee (c_2 \wedge c_3)$, then $c' \equiv \neg(\neg(c_0 \wedge c_1) \wedge \neg(c_2 \wedge c_3))$. From Lemma 5.3 and Lemma 5.6,

$$\begin{array}{rcl} c_0 \wedge c_1 & \overset{\theta}{\leftrightarrow} & P_0^{\gamma_1} * P_1^{\gamma_2} * P_0^{\delta_1} * P_1^{\delta_2}, \\ c_2 \wedge c_3 & \overset{\theta}{\leftrightarrow} & P_2^{\gamma_1} * P_3^{\gamma_2} * P_2^{\delta_1} * P_3^{\delta_2}. \end{array}$$

$\Longrightarrow$

$$\begin{array}{rcl} \neg(c_0 \wedge c_1) & \overset{\theta^{-1}}{\longleftrightarrow} & (P_0^{\gamma_1} * P_1^{\gamma_2} * P_0^{\delta_1} * P_1^{\delta_2})_{t(\theta^{-1})}, \\ \neg(c_2 \wedge c_3) & \overset{\theta^{-1}}{\longleftrightarrow} & (P_2^{\gamma_1} * P_3^{\gamma_2} * P_2^{\delta_1} * P_3^{\delta_2})_{t(\theta^{-1})}. \end{array}$$

$\Longrightarrow$

$$\begin{array}{rcl} \neg(c_0 \wedge c_1) & \overset{\theta}{\leftrightarrow} & [(P_0^{\gamma_1} * P_1^{\gamma_2} * P_0^{\delta_1} * P_1^{\delta_2})_{t(\theta^{-1})}]^{\tau}, \\ \neg(c_2 \wedge c_3) & \overset{\theta}{\leftrightarrow} & [(P_2^{\gamma_1} * P_3^{\gamma_2} * P_2^{\delta_1} * P_3^{\delta_2})_{t(\theta^{-1})}]^{\tau}. \end{array}$$

$\Longrightarrow \neg(c_0 \wedge c_1) \wedge \neg(c_2 \wedge c_3) \overset{\theta}{\leftrightarrow}$

$$\begin{array}{llll} & [(P_0^{\gamma_1} * P_1^{\gamma_2} * P_0^{\delta_1} * P_1^{\delta_2})_{t(\theta^{-1})}]^{\gamma_1\tau} & * & [(P_2^{\gamma_1} * P_3^{\gamma_2} * P_2^{\delta_1} * P_3^{\delta_2})_{t(\theta^{-1})}]^{\gamma_2\tau} \\ * & [(P_0^{\gamma_1} * P_1^{\gamma_2} * P_0^{\delta_1} * P_1^{\delta_2})_{t(\theta^{-1})}]^{\delta_1\tau} & * & [(P_2^{\gamma_1} * P_3^{\gamma_2} * P_2^{\delta_1} * P_3^{\delta_2})_{t(\theta^{-1})}]^{\delta_2\tau} \\ & (= P'). & & \end{array}$$

$\Longrightarrow \neg(\neg(c_0 \wedge c_1) \wedge \neg(c_2 \wedge c_3)) \overset{\theta}{\leftrightarrow} (P'_{t(\theta^{-1})})^{\tau}$.
We may rewrite this by the following way:

$$\begin{array}{rllll} c' \overset{\theta}{\leftrightarrow} & P_0^{h(0)} & *P_1^{h(1)} & *P_0^{h(2)} & *(P_1^{h(3)})_{t(u(0))} \\ * & P_2^{h(4)} & *P_3^{h(5)} & *P_2^{h(6)} & *(P_3^{h(7)})_{t(u(1))} \\ * & P_0^{h(8)} & *P_1^{h(9)} & *P_0^{h(10)} & *(P_1^{h(11)})_{t(u(2))} \\ * & P_2^{h(12)} & *P_3^{h(13)} & *P_2^{h(14)} & *(P_3^{h(15)})_{t(u(3))}, \end{array}$$

where

$$\begin{array}{llllllll} h(0) & = \tau\gamma_1\tau\gamma_1, & h(1) & = \tau\gamma_1\tau\gamma_2, & h(2) & = \tau\gamma_1\tau\delta_1, & h(3) & = \tau\gamma_1\tau\delta_2, \\ h(4) & = \tau\gamma_2\tau\gamma_1, & h(5) & = \tau\gamma_2\tau\gamma_2, & h(6) & = \tau\gamma_2\tau\delta_1, & h(7) & = \tau\gamma_2\tau\delta_2, \\ h(8) & = \tau\delta_1\tau\gamma_1, & h(9) & = \tau\delta_1\tau\gamma_2, & h(10) & = \tau\delta_1\tau\delta_1, & h(11) & = \tau\delta_1\tau\delta_2, \\ h(12) & = \tau\delta_2\tau\gamma_1, & h(13) & = \tau\delta_2\tau\gamma_2, & h(14) & = \tau\delta_2\tau\delta_1, & h(15) & = \tau\delta_2\tau\delta_2, \end{array}$$

and

$$\begin{array}{ll} u(0) = (\tau\gamma_1\tau)\theta^{-1}(\tau\gamma_1\tau)^{-1}, & u(1) = (\tau\gamma_2\tau)\theta^{-1}(\tau\gamma_2\tau)^{-1}, \\ u(2) = (\tau\delta_1\tau)\theta^{-1}(\tau\delta_1\tau)^{-1}, & u(3) = (\tau\delta_2\tau)\theta^{-1}(\tau\delta_2\tau)^{-1}\tau\theta^{-1}\tau^{-1}. \end{array}$$

Note that in Example 5.8 we always convert the permutation to θ at each stage of construction. This will simplify the construction of branching program for *tree*.

Remark 5.10. In order to be consistent with the convention in branching program, we reverse the order of input sequence. Anyway they are equivalent:

$$(\bigvee\bigwedge)^m(x_0, \ldots, x_{4^m-1}) = (\bigvee\bigwedge)^m(x_{4^m-1}, \ldots, x_0).$$

Example 5.11. **(Branching Program for $(\bigvee\bigwedge)^m$)**

Now we can use Example 5.8 to construct $(\bigvee\bigwedge)^m(x_0,\ldots,x_{4^m-1})$ recursively. The branching program $P((\bigvee\bigwedge)^m;\overrightarrow{x}(0,4^m-1))$ is defined as follows:

$$\begin{array}{lllll} P((\bigvee\bigwedge)^0;x_i)=\langle i,\theta,e\rangle, & & & & \\ P((\bigvee\bigwedge)^{n+1};\overrightarrow{x}(i,i+4^{n+1}-1))= & P_0^{h(0)} & *P_1^{h(1)} & *P_0^{h(2)} & *(P_1^{h(3)})_{t(u(0))} \\ & *P_2^{h(4)} & *P_3^{h(5)} & *P_2^{h(6)} & *(P_3^{h(7)})_{t(u(1))} \\ & *P_0^{h(8)} & *P_1^{h(9)} & *P_0^{h(10)} & *(P_1^{h(11)})_{t(u(2))} \\ & *P_2^{h(12)} & *P_3^{h(13)} & *P_2^{h(14)} & *(P_3^{h(15)})_{t(u(3))}, \end{array}$$

where $P_0=P((\bigvee\bigwedge)^n;\overrightarrow{x}(i,i+4^n-1))$, $P_1=P((\bigvee\bigwedge)^n;\overrightarrow{x}(i+4^n,i+2\cdot4^n-1))$, $P_2=P((\bigvee\bigwedge)^n;\overrightarrow{x}(i+2\cdot4^n,i+3\cdot4^n-1))$, $P_3=P((\bigvee\bigwedge)^n;\overrightarrow{x}(i+3\cdot4^n,i+4\cdot4^n-1))$, and $h(0),\ldots,h(15)$,$u(0),\ldots,u(3)$ are defined in Example 5.8. (To avoid ambiguity, we may define $\overrightarrow{x}$ as an infinite sequence $Bit(0,x),Bit(1,x),Bit(2,x),\ldots,Bit(i,x),\ldots$)

In $P((\bigvee\bigwedge)^m;\overrightarrow{x}(i,i+4^m-1))$, the number i is called the starting point of $\overrightarrow{x}$.

Lemma 5.12. *The length of* $P((\bigvee\bigwedge)^m;\overrightarrow{x}(i+4^m,i+2\cdot4^m-1))$ *is* 4^{2m}.

Proof. By induction. □

Recall that F is definable from p,q_0,q_1 by k-BRN (k-bounded recursion on notation) for $k\in\mathbb{N}$ if

$$\begin{array}{rcll} F(0,\overrightarrow{x}) &=& p(\overrightarrow{x}), & \\ F(2n,\overrightarrow{x}) &=& q_0(n,\overrightarrow{x},F(n,\overrightarrow{x})) & \text{if } n>0, \\ F(2n+1,\overrightarrow{x}) &=& q_1(n,\overrightarrow{x},F(n,\overrightarrow{x})), & \end{array}$$

and $0\le F(n,\overrightarrow{x})\le k$ for all $n,\overrightarrow{x}$.

We will use a variation of k-BRN by restricting $1\le F(n,\overrightarrow{x})\le k+1$ instead. Note that the variation is equivalent to the original one. Since we treat $\sigma\in S_n$ as a bijective map from $\{1,\ldots,n\}$ to $\{1,\ldots,n\}$, it would be better to use the variation.

Lemma 5.13. *If* $q\in A_0(k\text{-}BRN)$, $q:\{0,1\}\times\mathbb{N}\times\mathbb{N}\to S_{k+1}$, *then for* $1\le s\le k+1$,

$$F(n,j,\overrightarrow{x})=[\prod_{i=0}^{|n|-1}q(Bit(i,n),\lfloor n/2^i\rfloor,\overrightarrow{x})](s)\ \ (\in\{1,2,\ldots,k+1\})$$

is in $A_0(k\text{-}BRN)$.

Proof. Note that $[\cdots](s)$ means the image of s by the product $[\cdots]$ in S_{k+1}. Define

$$\begin{array}{rcll} F(0,s,\overrightarrow{x}) &=& s, & \\ F(2n,s,\overrightarrow{x}) &=& q(0,n,\overrightarrow{x})(F(n,s,\overrightarrow{x})) & \text{if } n>0, \\ F(2n+1,s,\overrightarrow{x}) &=& q(1,n,\overrightarrow{x})(F(n,s,\overrightarrow{x})). & \end{array}$$

Then $1\le F(n,s,\overrightarrow{x})\le k+1$ and $F(n,s,\overrightarrow{x})\in A_0(k\text{-BRN})$. The equality $F(n,s,\overrightarrow{x})=[\prod_{i=0}^{|n|-1}g(Bit(i,n),\lfloor n/2^i\rfloor,\overrightarrow{x})](s)$ can be easily verified by induction. □

We encode $x\in\mathbb{N}$ into $\overrightarrow{x}$ as follows:

$$\overrightarrow{x}=\langle x_0,\ldots,x_{n-1}\rangle=\langle Bit(0,x),\ldots,Bit(n-1,x)\rangle$$

where $n=|x|$. Note that $tree(x)=(\bigvee\bigwedge)^m(\overrightarrow{x}(0,4^m-1))$ where $4^m\le|x|<4^{m+1}$. From Lemma 5.12, we may assume that the branching program $P((\bigvee\bigwedge)^m;\overrightarrow{x}(0,4^m-1))=\langle j(i,\overrightarrow{x}),f(i,\overrightarrow{x}),g(i,\overrightarrow{x})\rangle$ for $1\le i\le4^{2m}$.

Lemma 5.14. *Suppose that* $P((\bigvee\bigwedge)^m; \overrightarrow{x}(0,4^m-1)) = \langle j(i,\overrightarrow{x}), f(i,\overrightarrow{x}), g(i,\overrightarrow{x})\rangle$ *for* $1 \le i \le 4^{2m}$. *If* $j(i,\overrightarrow{x})$,$f(i,\overrightarrow{x})$, $g(i,\overrightarrow{x}) \in A_0$, *then* $tree(x) \in A_0(4\text{-}BRN)$.

Proof. Define $q(0,i,\overrightarrow{x}) = f(i,\overrightarrow{x})$ and $q(1,i,\overrightarrow{x}) = g(i,\overrightarrow{x})$. By Lemma 5.13, we have $F \in A_0$(4-BRN). Now we use CRN and $j(i,\overrightarrow{x})$ to construct $n(\overrightarrow{x})$:

$$n(\overrightarrow{x}) = \overrightarrow{x}(j(1,\overrightarrow{x})) * \overrightarrow{x}(j(2,\overrightarrow{x})) * \cdots * \overrightarrow{x}(j(4^m,\overrightarrow{x})).$$

(Note that $4^m \le |x|$ is sharply bounded.) So $n(\overrightarrow{x}) \in A_0$. We can compute $tree(\overrightarrow{x})$ as follows:

$$tree(\overrightarrow{x}) = \begin{cases} 1 & \text{if } F(n(\overrightarrow{x}), s, \overrightarrow{x}) = \theta(s) \text{ for all } s \in \{1,2,3,4,5\}; \\ 0 & \text{if } F(n(\overrightarrow{x}), s, \overrightarrow{x}) = s \text{ for all } s \in \{1,2,3,4,5\}. \end{cases}$$

By Lemma 5.13, $tree \in A_0$(4-BRN). □

We need two technical lemmas to complete this work.

Lemma 5.15. $j(i,\overrightarrow{x}) \in A_0$.

Proof. Assume that $1 \le i \le 4^{2m}$, then $0 \le i-1 \le 4^{2m}-1$. Let

$$i - 1 = v_{4m-1} * v_{4m-2} * v_{4m-3} * v_{4m-4} * \cdots * v_3 * v_2 * v_1 * v_0,$$

where $v_l \in \{0,1\}$. According to Example 5.11, the branching program for $(\bigvee\bigwedge)^m$ with input $\overrightarrow{x}(0,4^m-1)$ can be constructed by concatenating 16 branching programs for $(\bigvee\bigwedge)^{m-1}$. The indices of these 16 branching programs are: 0, 1, 0, 1, 2, 3, 2, 3, 0, 1, 0, 1, 2, 3, 2, 3. Hence the i-th instruction is from $P_{v_{4m-2}*v_{4m-4}}$. Since the starting point of $\overrightarrow{x}(0,4^m-1)$ is 0, inductively

$$j(i) = v_{4m-2} * v_{4m-4} * v_{4m-6} * v_{4m-8} * \cdots * v_6 * v_4 * v_2 * v_0.$$

Hence $j(i)$ is computable in A_0. □

It is not necessary to find the explicit expression for $j(i,\overrightarrow{x})$.

Lemma 5.16. $f(i,\overrightarrow{x}), g(i,\overrightarrow{x}) \in A_0$.

Proof. We use the same assumption in Lemma 5.15. Let $0 \le i-1 \le 4^{2m}-1$ and

$$i - 1 = v_{4m-1} * v_{4m-2} * v_{4m-3} * v_{4m-4} * \cdots * v_3 * v_2 * v_1 * v_0,$$

where $v_l \in \{0,1\}$. According to Example 5.11, when we trace $f(i), g(i)$ inductively, each time there are two parts added: the conjugate part $h(\cdot)$ and the tail part $u(\cdot)$. First we compute the conjugate part inductively. Define

$$\begin{aligned} C(i,1) &= h(v_{4m-1} * v_{4m-2} * v_{4m-3} * v_{4m-4}) \cdot h(v_{4m-5} * v_{4m-6} * v_{4m-7} * v_{4m-8}) \\ &\quad \cdot \ldots \cdot h(v_3 * v_2 * v_1 * v_0) \\ &= \prod_{l=1}^{m} h(v_{4l-1} * v_{4l-2} * v_{4l-3} * v_{4l-4}). \end{aligned}$$

Because $P((\bigvee\bigwedge)^0; x_i) = \langle i, \theta, e\rangle$, the conjugate parts for $f(i), g(i)$ are $\theta^{C(i,1)}, e^{C(i,1)}$ respectively. Since $m \le |x|_2$, $C(i,1)$ is computable by weak 5-BRN. Therefore $C(i,1)$ is computable in A_0.

Now consider the tail part. By the defining construction in Example 5.11, the tail part will be added only when $v_{4m-1} * v_{4m-2} * v_{4m-3} * v_{4m-4} =$ 0011,0111,1011, or 1111. We may think of the other cases as adding a trivial tail e. For $0 < j \le m$, define

$$T(i,j) = \begin{cases} u(0) & \text{if } v_{4j-1} * v_{4j-2} * v_{4j-3} * v_{4j-4} = 0011; \\ u(1) & \text{if } v_{4j-1} * v_{4j-2} * v_{4j-3} * v_{4j-4} = 0111; \\ u(2) & \text{if } v_{4j-1} * v_{4j-2} * v_{4j-3} * v_{4j-4} = 1011; \\ u(3) & \text{if } v_{4j-1} * v_{4j-2} * v_{4j-3} * v_{4j-4} = 1111; \\ e & \text{else.} \end{cases}$$

By Lemma 5.4, once a tail part is added, in the rest of the whole computation it will not be changed. For this we define the conjugate function

$$\begin{aligned} C(i,r) &= h(v_{4m-1} * v_{4m-2} * v_{4m-3} * v_{4m-4}) \cdot h(v_{4m-5} * v_{4m-6} * v_{4m-7} * v_{4m-8}) \\ &\quad \cdot \ldots \cdot h(v_{4r-1} * v_{4r-2} * v_{4r-3} * v_{4r-4}) \\ &= \prod_{l=r}^{m} h(v_{4l-1} * v_{4l-2} * v_{4l-3} * v_{4l-4}) \end{aligned}$$

for $r \le m$. Now we can compute $f(i), g(i)$:

$$\begin{aligned} f(i) &= \theta^{C(i,1)} \cdot T(i,1)^{C(i,2)} \cdot T(i,2)^{C(i,3)} \cdot \ldots \cdot T(i,m-1)^{C(i,m)} \cdot T(i,m) \\ &= \theta^{C(i,1)} \cdot \prod_{l=1}^{m-1} T(i,l)^{C(i,l+1)} \cdot T(i,m). \end{aligned}$$

Since $m \le |x|_2$, $C(i,r), f(i)$ are computable by weak 4-BRN. Hence $f(i) \in A_0$. Similarly $g(i) \in A_0$. Actually,

$$g(i) = e^{C(i,1)} \cdot \prod_{l=1}^{m-1} T(i,l)^{C(i,l+1)} = \prod_{l=1}^{m-1} T(i,l)^{C(i,l+1)}.$$

□

Theorem 5.17. *tree* $\in A_0$*(4-BRN). (Hence* $N_0 \subseteq N_0'$*.)*

Proof. Apply Lemmas 5.14, 5.15, 5.16. □

6. The case $LR(f)$ with $|B| \le 4$

Let $|B| = 4$ and $f : B \times B \to B$. To prove that $LR(f)[b_0 b_1 \ldots b_n]$ is computable in $A_0(6)$, it suffices to show that $f_n \circ f_{n-1} \circ \cdots \circ f_1(b_0)$ is computable in $A_0(6)$, where $f_i : B \to B$ for $1 \le i \le n$ and b_0 are inputs. It is because we may set $f_i(x) = f(x, b_i)$.

We first consider that case that $|Im(f_i)| = 4$ for $1 \le i \le n$. In this case each f_i is a permutation on B. Then it suffices to prove that the group multiplication $\prod_{i=1}^n g_i$ on the permutation group S_4 is computable in $A_0(6)$.

Lemma 6.1. *The group multiplication* $\prod_{i=1}^n g_i$ *in* S_4 *is computable in the function algebra* $A_0(6)$.

Proof. Let $V = \{e, (1\,2)(3\,4), (1\,3)(2\,4), (1\,4)(2\,3)\}$ be the four group. Then V is abelian and $V \lhd A_4 \lhd S_4$.

Let $a = (1\,2) \in S_4$ and $b = (1\,2\,3) \in A_4$. Then we can express S_4 and A_4 by the unions of disjoint cosets: $S_4 = A_4 \cup (a \cdot A_4)$ and $A_4 = V \cup (b \cdot V) \cup (b^2 \cdot V)$.

First we express each $g_i = a^{m_i} h_i$ for some $h_i \in A_4$ and $m_i \in \{0, 1\}$. This can be done in parallel, i.e., by CRN. Then

$$\begin{array}{rcl}\prod_{i=1}^n g_i & = & a^{m_1} h_1 \cdot a^{m_2} h_2 \cdot \ldots \cdot a^{m_n} h_n \\ & = & (a^{m_1} h_1 a^{-m_1}) \cdot \ldots \cdot (a^{\sum_{j=1}^i m_j} h_i a^{-\sum_{j=1}^i m_j}) \cdot \\ & & \quad \ldots \cdot (a^{\sum_{j=1}^n m_j} h_n a^{-\sum_{j=1}^n m_j}) \cdot a^{\sum_{j=1}^n m_j} \\ & = & \prod_{i=1}^n h_i' \cdot a^{\sum_{j=1}^n m_j},\end{array}$$

where $h_i' = a^{\sum_{j=1}^i m_j} h_i a^{-\sum_{j=1}^i m_j}$ and $a^{\sum_{j=1}^n m_j}$ are computable in $A_0(2)$. And the problem is reduced to a product $\prod_{i=1}^n h_i'$ in the group A_4.

Similarly, we can reduce $\prod_{i=1}^n h_i'$ into a product in V by an $A_0(3)$ computation. Since V is isomorphic to $Z_2 \oplus Z_2$, the product in V is computable in $A_0(2)$. $\square$

Example 6.2. Suppose that we have $f_1, f_2, \ldots, f_n : [k] \to [k]$ and $|Im(f_i)| = k - 1$, and we want to compute $f_n \circ f_{n-1} \circ \cdots \circ f_2 \circ f_1$, then we can reduce this to a composition of functions all of which has domain and range $[k-1]$. First for each f_i there is a permutation g_i on $[k]$ such that $Im(g_i f_i) = [k-1]$. Then

$$f_n \circ f_{n-1} \circ \cdots \circ f_2 \circ f_1 = g_n^{-1} \circ g_n f_n g_{n-1}^{-1} \circ g_{n-1} f_{n-1} g_{n-2}^{-1} \circ \cdots \circ g_2 f_2 g_1^{-1} \circ g_1 f_1.$$

Since $Im(f_i g_{i-1}^{-1}) = Im(f_i)$, $Im(g_i f_i g_{i-1}^{-1}) = Im(g_i f_i) = [k-1]$. Define $h_i = g_i f_i g_{i-1}^{-1}$ for $2 \le i \le n$. Note that h_i only depends on g_i, g_{i-1}, and then this can be determined from f_i, f_{i-1}. (Hence this can be computed in A_0.) After the first mapping $g_1 f_1$ the image is restricted to $[k-1]$. Then

$$\begin{aligned} f_n \circ f_{n-1} \circ \cdots \circ f_2 \circ f_1 &= g_n^{-1} \circ h_n \circ h_{n-1} \circ \cdots \circ h_2 \circ g_1 f_1 \\ &= g_n^{-1} \circ \overbrace{h_n \restriction_{[k-1]} \circ h_{n-1} \restriction_{[k-1]} \circ \cdots \circ h_2 \restriction_{[k-1]}}^{(*)} \circ g_1 f_1. \end{aligned}$$

The part $(*)$ is a composition of functions with domain and range $[k-1]$.

Example 6.3. If $min_{1 \le i \le n} |Im(f_i)| = 3$ and $i_1 < i_2 < \cdots < i_l$ are all i such that $|Im(f_i)| = 3$, then

$$f_n \circ \cdots \circ f_1 = (f_n \circ \cdots \circ f_{i_l+1}) \circ (f_{i_l} \circ \cdots \circ f_{i_{l-1}+1}) \circ \cdots \circ (f_{i_2} \circ \cdots \circ f_{i_1+1}) \circ (f_{i_1} \circ \cdots \circ f_1).$$

Each $h_j = (f_{i_{j+1}} \circ \cdots \circ f_{i_j+1})$ is computable in $A_0(6)$. Then we can apply the trick in Example 6.2 to compute $h_l \circ \cdots \circ h_1$.

We can then consider the more general case as in the following example.

Example 6.4. Consider the following composition $f_n \circ \cdots \circ f_1$ with each function $f_i : B \to B$.

$$|Im(f_i)| = \overbrace{\overbrace{\underset{2}{f_{22}} \overbrace{\underset{3}{f_{21}} \underset{4}{f_{20}} \underset{4}{f_{19}}} \overbrace{\underset{3}{f_{18}} \underset{4}{f_{17}}}} \overbrace{\underset{1}{f_{16}} \overbrace{\underset{2}{f_{15}} \overbrace{\underset{3}{f_{14}} \underset{4}{f_{13}}} \overbrace{\underset{3}{f_{12}} \underset{4}{f_{11}}}} \overbrace{\underset{2}{f_{10}} \overbrace{\underset{3}{f_9} \underset{4}{f_8}} \underset{3}{f_7} \overbrace{\underset{3}{f_6} \underset{4}{f_5} \underset{4}{f_4}} \overbrace{\underset{3}{f_3} \underset{4}{f_2} \underset{4}{f_1}}}}}$$

In this example, $n = 22$ and every $|Im(f_i)|$ is written right below f_i. Then according to the tricks in Examples 6.3,6.2 and Lemma 6.1, we can compute the composition "block by block" (as the way we brace them) from bottom to top in at most 4 levels.

Now the following theorem is obvious.

Theorem 6.5. *The composition of functions $f_n \circ \cdots \circ f_1$ with each $f_i : B \to B$ and $|B| = 4$ is computable in $A_0(6)$.*

Corollary 6.6. *If f is definable from g, h_0, h_1 by k-BRN with $g, h_0, h_1 \in A_0$, then $f \in A_0(2)$ while $k = 1$, and $f \in A_0(6)$ while $k = 2, 3$.*

This means: "1-BRN $\Leftrightarrow AC^0(Mod_2)$", "2-BRN $\Leftrightarrow AC^0(Mod_2 + Mod_3) \Leftrightarrow$ 3-BRN."

Question. For $|B| = k > 4$ (here k may be equal to $\log n$ or $(\log\log n)$), can the composition of functions $f_n \circ \cdots \circ f_1$ be computable in ACC if (1) in each block the computation is computable in ACC and (2) the braces are nested with depth $\le \log\log n$?

7. The associative case with $|B| < 60$

In this section we prove that $LR(f)$ for associative f with $|B| \le 59$ is computable in ACC.

Recall that a monoid $(M, \circ)$ consists of a nonempty set M and an associative binary operation $\circ : M \times M \to M$ and there is an identity element $e \in M$, i.e., $e \circ x = x \circ e$ for all $x \in M$.

A monoid is solvable if any of its subsets which form groups under the monoid operation are solvable groups.

Theorem 7.1. *(from [4]) A language is recognizable by polynomial length NUDFA's over a solvable monoid iff it is in ACC.*

Here we only use the $\Rightarrow$ direction. Since our concern is the word problem over a monoid, it suffices to show that the monoids in consideration are solvable.

Theorem 7.2. *$LR(f)$ for associative f with $|B| \le 59$ is computable in ACC.*

Proof. Since the smallest nontrivial simple group is A_5 (with 60 elements), (B, f) is solvable if it is a monoid.

Consider the case that $f : B \times B \to B$ is associative. If every element in B is not an identity element, we can add a new identity element e to B and expand f to $f : (B \cup \{e\}) \times (B \cup \{e\}) \to (B \cup \{e\})$: $f(e, x) = f(x, e) = x$ for any $x \in (B \cup \{e\})$.

If $|B| < 59$, then $|B \cup \{e\}| \le 59$ and $B \cup \{e\}$ is solvable.

If $|B| = 59$, then $B \cup \{e\}$ can not be isomorphic to A_5: if it were, then $e = f(a, b)$ for some $a, b \in B$ (that is because A_5 is a group), a contradiction (for $e \notin B$). Then $B \cup \{e\}$ must be solvable (otherwise it is isomorphic to A_5). □

8. Commutator and icosahedron

Is the discovery of the existence of three 5-cycles σ, δ, τ with $\sigma\delta\sigma^{-1}\delta^{-1} = \tau$ a big surprise?

One may start the investigation as what is presented in Section 6 and then start investigating A_5. Let $\alpha *_c \beta = \alpha\beta\alpha^{-1}\beta^{-1}$ be the commutator operation. By calculating $*_c$ over 5-cycles in S_5, we find an interesting structure: There are 2 disjoint subsets of 5-cycles (in S_5), say, A, A', which satisfy the following conditions: (1) $|A| = |A'| = 12$. (2) $\alpha \in A$ (or A') implies $\alpha^{-1} \in A$ (or A'). (3) For any $\alpha, \beta \in A$ (or A'), if $\alpha *_c \beta \ne e$, the identity element in S_5, then $\alpha *_c \beta \in A$ (or A'). By $*_c$ and such α, β, $A \cup \{e\}$ (or $A' \cup \{e\}$) can be generated.

By conjugate operation it is clear that A, A' are isomorphic. (Note that $A' = \{\alpha^2 : \alpha \in A\}$.) We may restrict $*_c$ on $A \cup \{e\}$. (Therefore, if one chooses three different 5-cycles a, b, c such that none of them is an inverse of the rest, then two of them must be in A or A' and these two will generate another 5-cycle by commutator. Hence it seems quite natural to discover this fact!) We are interested in the computational power of $LR(*_c), RL(*_c), DC(*_c)$. In Part II we will prove that $AC^0(DC(*_c)) = NC^1$ and $AC^0(LR(*_c)) = AC^0(Mod_{10})$. To prove the second statement, we need to visualize $*_c$ geometrically, by assigning the twelve 5-cycles in A to the vertices of icosahedron so that $\alpha *_c \beta = \gamma$ is geometrically invariant, i.e., for any rotation R on the icosahedron, $R(\alpha) *_c R(\beta) = R(\gamma)$. In the proof we will first show that $AC^0(Mod_{10}) \subseteq AC^0(LR(*_c)) \subseteq AC^0(Mod_{30})$, and then remove the Mod_3 gates.

Although Theorem 7.1 (from [4]) characterizes the solvable monoid case, it seems not clear that "If $f \in AC^0(Mod_k)$, then $AC^0(f) = AC^0(Mod_m)$ for some $m \mid k$." is always true. Our case "$AC^0(Mod_{10}) = AC^0(LR(*_c))$" somehow suggests that "If $f \in AC^0(Mod_k)$, then $AC^0(f) = AC^0(Mod_m)$ for some $m \mid k$." may not be proved easily. On searching any counterexample of it, what may deserve to investigate are the case $|B| \leq 4$ and the case $DC(f)$ for associative f with $|B| < 60$ (especially $|B| = 5$).

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 1–9, Boston, Massachusetts, 25–27 April 1983.

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

[3] D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. In *SCT: Annual Conference on Structure in Complexity Theory*, pages 47–59, 1988.

[4] David A. Mix Barrington and Denis Thérien. Finite monoids and the fine structure of NC^1. *J. Assoc. Comput. Mach.*, 35(4):941–952, 1988.

[5] S. R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic*, 52:916–927, 1987.

[6] P. Clote. Sequential machine independent characterizations of the parallel complexity classes AlogTIME, AC^k, NC^k, and NC. In *Feasible Mathematics: A Mathematical Sciences Institute Workshop held in Ithaca, New York, June 1989*, pages 49–69. Birkhäuser, 1990.

[7] P. Clote. On polynomial size Frege proofs of certain combinatorial principles. In *Clote & Krajíček (Eds.), Arithmetic, Proof Theory, and Computational Complexity*, pages 162–184. Clarendon Press, 1993.

[8] P. Clote and G. Takeuti. First order bounded arithmetic and small boolean circuit complexity classes. In *Feasible Mathematics II: A Mathematical Sciences Institute Workshop*, pages 154–218. Birkhauser, 1995.

[9] N. Immerman. Expressibility and parallel complexity. *SIAM Journal of Computing*, 18(3):625–638, June 1989.

[10] J.-L. Lee. *Count and tree in uniform NC^1*. PhD thesis, Department of Mathematics, University of Illinois at Urbana-Champaign, 1997.

[11] J.-L. Lee. Counting in uniform TC^0. Technical Report TR97–034, Electronic Colloquium on Computational Complexity, 1997.

[12] L. G. Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5:363–366, 1984.

Philosophy department, National Chung-Cheng University, Taiwan
E-mail address: jllee@phil.ccu.edu.tw

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'