An Impossibility Gap Between Width-4 and Width-5 Permutation Branching Programs

Alex Brodsky [1]

Department of Computer Science, University of Toronto,
10 King's College Rd., Toronto, ON, M5S 3G4, Canada

Key words: Computational Complexity, Formal Languages, Branching Programs

1 Introduction

The study of branching programs [10] yields insight into the structure of low-lying complexity classes—those that are contained within **P**. There has been significant progress towards deriving bounds for branching programs, particularly for bounded-width branching programs [8,12,5] and for polylogarithmic width branching programs [1]. Perhaps the best known result, which appeared in STOC '86, is that of Barrington [3] who showed that $\mathbf{NC}^1$ is exactly the class of functions computed by polynomial length width-w permutation branching programs (w-PBP).

One motivation for investigating permutation branching programs is to study the fine structure of $\mathbf{NC}^1$. Specifically, to determine whether the class **ACC** (the languages recognized by constant-depth polynomial-size families of circuits comprising unbounded fan-in AND, OR, and MOD-q gates for some fixed q) is strictly contained within $\mathbf{NC}^1$. Barrington and Thérien [7] showed that a language is in **ACC** if and only if it can be accepted by a polynomial length program over a solvable monoid—a monoid that contains no subsets that are nonsolvable groups. Consequently, showing that no polynomial length program over a solvable group can compute the conjunction would prove proper containment. While there has been some progress in showing this for programs over some solvable groups [6], the general problem remains open. Showing that polynomial length 4-PBPs cannot compute conjunction would be a major step towards solving this open problem.

Email address: `abrodsky@cs.utoronto.ca` (Alex Brodsky).
[1] This work was supported by an NSERC PGS B Fellowship and by a Killam Fellowship while the author was at the University of British Columbia, Canada.

Barrington [4] defined a w-PBP B as a sequence of instructions; each of which is a tuple (j_i, f_i, g_i), where $j_i \in \mathbb{Z}_n$ identifies one of the n inputs $\{x_1, x_2, \ldots, x_n\}$, and f_i and g_i are permutations on $\mathbb{Z}_w$. If x_{j_i} is true, the ith instruction yields permutation f_i, otherwise it yields g_i. The output, $B(x)$, of program B on input x, is the product of the permutations yielded by the instructions of B. Program B (strongly) accepts language L if there is a fixed permutation $\sigma \neq \varepsilon$ such that $B(x) = \sigma$ if $x \in L$ and $B(x) = \varepsilon$, otherwise. There is also a notion of weak acceptance, which we also consider in this paper.

Using permutations from the alternating group A_5, Barrington's [4] construction simulates an AND operation using a commutator and inductively composes a polynomial length 5-PBP that realizes any function in $\mathbf{NC}^1$. The other direction follows from the fact that computing the product of a polynomial number of elements from a fixed finite group is in $\mathbf{NC}^1$.

A corollary of a theorem of Maurer and Rhodes [11] is that 3-PBPs can compute any Boolean function. However, we know that 3-PBPs are not as powerful as 5-PBPs; computing the conjunction requires an exponentially long program [2]. Not surprisingly, there has been significant work done to investigate the reasons for the disparity between 3-PBPs and 5-PBPs, and to generalize the lower bounds for bounded-width permutation branching programs and programs over groups [7,6,5]. One deceptively difficult problem is determining if there exists a polynomial length 4-PBP that can compute the conjunction of n variables. Barrington et al. [6] have proven that the answer is negative if the instructions of the program are restricted to the alternating group A_4. Intriguingly, the problem for general 4-PBPs remains open!

In [4,6] it is conjectured that no program over a solvable group can compute the conjunction on n variables—all finite groups of order less than 60 are solvable [9]. If the conjecture is true, then the alternating group A_5 is the smallest group over which polynomial length programs can compute the conjunction on n variables. Otherwise, there may yet be a clever construction that realizes a polynomial length program over S_4 that computes the conjunction.

To further our insight into this problem we consider two modes of acceptance, strong and weak, and differentiate programs based on their acceptance mode. We then pose the following problem: Can a weakly accepting program be transformed into strongly accepting one. Under a relatively general transformation model, we show that this is possible for width-5 permutation branching programs and impossible for width-4 permutation branching programs. This defines a natural computational gap between width-4 and width-5 permutation branching programs.

2 Definitions

For the most part, we will use Barrington's [2,4] notation and definitions. However, we will use a slightly simpler notation in several instances. A **width-w permutation branching program** (w-PBP) P on n variables $x_1,\ldots,x_n$ is a sequence of instructions of the form σ^{x_i} or σ^1. An **instruction σ^{x_i} yields** permutation $\sigma \in S_w$ if $x_i = 1$ and yields the identity, ε, otherwise; the **oblivious instruction** σ^1 always yields σ. On input x, program P yields a permutation $P(x) \in S_w$, where $P(x)$ is the product of the yields of the instructions of P. A program P over a group G is defined in a similar manner [6]: each instruction σ^{x_i} yields either a group element $\sigma \in G$ or the identity, and the yield of the program, $P(x)$, is also an element of G. Thus, w-PBPs are programs over the symmetric group S_w. The **length** of program P, denoted $|P|$, is the number of instructions in P.

A program P over a group G **strongly accepts** a language $L \subseteq B_n$ if there exists a $\sigma \in G$, $\sigma \neq \varepsilon$, such that for all $x \in B_n$, $P(x) = \sigma$ if $x \in L$ and $P(x) = \varepsilon$ if $x \notin L$. Similarly, a program P over a group G **weakly accepts** a language $L \subseteq B_n$ if for all $x \in B_n$, $P(x) \neq \varepsilon$ if and only if $x \in L$. Observe that all strongly accepting programs are also weakly accepting programs. Additionally, if f is the characteristic Boolean function of L, i.e., $f(x) = 1$ if and only if $x \in L$, then we say that program P **strongly computes** or **weakly computes** function f.

2.1 Program Transformations

Branching programs may be composed in several ways. The concatenation of two programs P and P', denoted PP', is the concatenation of the instruction sequences of P and P'. Since the inverse of a concatenation $(PP')^{-1}$ is $P'^{-1}P^{-1}$, the inverse of a program P is the reversed sequence of the inverted instructions of P. A **commutator** of programs P and P', $[P,P'] = PP'P^{-1}P'^{-1}$, mirrors the group-theoretic definition of a commutator. Finally, the kth power of P is the concatenation of k copies of P, i.e., $P^k = PP\ldots P$.

A **program transformation** Ψ takes a program P (over G) as an argument and yields a new program via a basic set of operations. A transformation is either

(1) a constant: $\Psi(P) = \tau^1$ for some $\tau \in G$,
(2) a projection: $\Psi(P) = P$,
(3) an inverse: $\Psi(P) = P^{-1}$, or
(4) a concatenation: $\Psi(P) = \Phi(P)\Upsilon(P)$ of two transformations.

For conciseness, the composition of transformations, $\Psi(\Phi(P))$, is commonly used: however it is not a basic operation because it is constructible from the preceding rules. Similarly, P^k denotes the concatenation of k copies of program P.

Concretely, a transformation is a word comprising placeholders and oblivious instructions, e.g., τ^1, where each placeholder is replaced by an instance of program P, or its inverse, when the transformation is applied to P. One important feature is that the resulting program $\Psi(P)$ has length that is only a constant factor larger than the original program.

Applying a transformation to an oblivious program $P = \tau^1$, which is a permutation, yields a permutation. Hence, a program transformation is also a map from G to itself. This concise characterization, called a **characteristic map**, is used extensively in our discussion; however, while each transformation is a map from G to itself, not all maps are valid transformations.

3 Main Result

The acceptance mode of a permutation branching program significantly affects the computational power of the model. First, we show that weakly accepting width-4 branching programs cannot necessarily be transformed into strongly accepting width-4 branching programs.

Second, we show that in the case of width-w permutation branching programs, where $w > 4$, weak acceptance is equal to strong acceptance. We exhibit a transformation that transforms any weakly accepting program into a strongly accepting program, whose length is constant multiple of the original program that only depends on w.

Thus, we show that there is a natural gap between width-4 and width-5 permutations branching programs: what can easily be accomplished in the latter model, cannot be accomplished at all in the former.

3.1 Weak Acceptance Does not Equal Strong Acceptance for 4*-PBPs*

Theorem 3.1 *There does not exist a transformation Ψ such that for every program P over S_4, $\Psi(P)$ is a strongly accepting program computing the same Boolean function as P.*

PROOF. By contradiction, assume that there exists a transformation Ψ that converts weakly accepting programs into strongly accepting ones. Let $\sigma_1 = (1\,2)(3\,4)$, let $\sigma_2 = (1\,3)(2\,4)$, and let $Q = \sigma_1^{x_1}\sigma_2^{x_2}$ be a program. Now, consider the structure of $\Psi(Q)$. $\Psi(Q)$ is a word comprising oblivious instructions of the form τ^1 and nonoblivious instructions: either $\sigma_1^{x_1}$ or $\sigma_2^{x_2}$. Note, that σ_1, σ_2, and the yield of Q are all elements of the Klein group $K = \{\varepsilon, (1\,2)(3\,4), (1\,3)(2\,4), (1\,4)(2\,3)\} \subset S_4$.

Via conjugation, we re-arrange the program by clustering all oblivious instructions at the beginning of the program: if $\sigma' = \tau^{-1}\sigma\tau$, then the instruction sequence $\sigma^{x_i}\tau^1$ can be rewritten as $\tau^1\sigma'^{x_i}$. This rewriting does not change the program's yield and since K is closed under conjugation, i.e., $\sigma'^{x_i} \in K$.

First, since $\Psi(Q)$ must yield ε whenever Q yields ε, the composition of all oblivious instructions must yield ε. Second, since all nonoblivious instructions of Q, and hence of $\Psi(Q)$, belong to the Klein group, the instructions commute. Hence, $\Psi(Q)$ can be rewritten as $\Psi(Q) = \tau_1^{x_1}\tau_2^{x_2}$, where $\tau_1, \tau_2 \in K$. Unless $\tau_1 = \tau_2$, $\Psi(Q)$ is not strongly accepting, and if equality holds then $\Psi(Q)$ yields ε on assignment $x_1 = x_2 = 1$, contrary to what Q yields. Hence, Ψ cannot exist. $\square$

3.2 Weak Acceptance Equals Strong Acceptance for w-PBPs, $w > 4$

If $w > 4$ then every weakly accepting w-PBP can be transformed into a strongly accepting program with only a linear increase in length. We first outline the construction and then present the details. To begin, we construct a transformation Ψ_π that is characterized by the map $\Psi_\pi(\sigma) = \varepsilon$ if and only if $\sigma \neq \pi$, where $\Psi_\pi(\pi) = \rho$ for some $\rho \neq \varepsilon$ and $\pi, \rho, \sigma \in S_w$. Hence, given program P, if $Q = \Psi_\pi(P)$, then Q yields ρ on input x if and only if P yields π on input x, and yields ε otherwise. In fact, we need only exhibit the construction for a fixed π_0, say $\pi_0 = (1\,2\ldots w)$ since $\Psi_\pi(P) = \Psi_{\pi_0}(P\pi^{-1}\pi_0)$. Finally, concatenating $|S_w| - 1$ copies of $\Psi_\pi(P)$ for each $\pi \in S_w \setminus \varepsilon$, yields the desired transformation.

The construction of transformation Ψ_{π_0} is based on the observation that it is sufficient to check the order of the adjacent points of a permutation to verify that the yielded permutation is $(1\,2\ldots w)$, i.e., that 1 precedes 2, 2 precedes 3, up to w precedes 1. Each such check is performed by composing two transformations: a transformation whose characteristic map sends each permutation to a transposition (see Lemma 3.3), and a transformation Υ_τ that takes the transposition as input and has the characteristic map $\Upsilon_\tau(\upsilon) = \varepsilon$ if and only if $\upsilon \neq \tau$. These "checks" are combined into a single transformation Ψ_{π_0} using Barrington's [4] standard commutator construction. We first exhibit Υ_τ.

Lemma 3.2 *Let $w > 4$ and let $\tau \in S_w$ be a transposition. There exists a transformation Υ_τ such that $\Upsilon_\tau(\varepsilon) = \varepsilon$ and for every transposition $\upsilon \in S_w$, $\Upsilon_\tau(\upsilon) \neq \varepsilon$ if and only if $\upsilon = \tau$. Furthermore, $\Upsilon_\tau(\tau)$ is represented by two disjoint transpositions.*

PROOF. Let $\tau = (r\,t) \in S_w$ and let $(s\,u)$ be a transposition such that $\{r,t\}$ and $\{s,u\}$ are disjoint. The straight-line composition of Υ_τ is:

$$\begin{aligned}
\Upsilon_0 &= [(rs), \upsilon] \\
\Upsilon_1 &= [(rus), \Upsilon_0] \\
\Upsilon_2 &= \Upsilon_1(su) \\
\Upsilon_3 &= (\Upsilon_2)^2 \\
\Upsilon_4 &= \Upsilon_3(tu) \\
\Upsilon_{(rt)} &= (\Upsilon_4)^6
\end{aligned}$$

If υ commutes with (rs)—υ is a disjoint transposition, the transposition (rs), or the identity—then $\Upsilon_0(\upsilon) = \varepsilon$, implying that $\Upsilon_1(\upsilon) = \Upsilon_3(\upsilon) = \Upsilon_{(rt)}(\upsilon) = \varepsilon$. Otherwise, there are two cases: either $\upsilon = (rv)$, $v \neq s$ or $\upsilon = (vs)$, $v \neq r$.

If $\upsilon = (rv)$, then $\Upsilon_0((rv)) = (rvs)$. If $v = u \neq t$ then $\Upsilon_1((rv)) = \varepsilon$, and so does $\Upsilon_{(rt)}((rv))$. Otherwise, $\Upsilon_1((rv)) = (rs)(uv)$, $\Upsilon_2((rv)) = (ruvs)$, and $\Upsilon_3((rv)) = (rv)(su)$. If $v \neq t$, then $\Upsilon_4((rv)) = (rv)(stu)$ and hence $\Upsilon_{(rt)}(rv) = \varepsilon$. On the other hand, if $v = t$, then $\Upsilon_4((rt)) = (rust)$ and hence, $\Upsilon_{(rt)}(rt) = (rs)(tu)$.

If $\upsilon = (vs)$, then $\Upsilon_0((vs)) = (rsv)$. If $v = u \neq t$. then $\Upsilon_1((vs)) = \varepsilon$, and so does $\Upsilon_{(rt)}((vs))$. Otherwise, $\Upsilon_1((vs)) = (rv)(us)$, $\Upsilon_2((vs)) = (rv)$, and $\Upsilon_3((vs)) = \varepsilon$, implying that $\Upsilon_{(rt)}(vs) = \varepsilon$.

Hence, $\Upsilon_{(rt)}(\upsilon) \neq \varepsilon$ if and only if $\upsilon = (rt)$, and $\Upsilon_{(rt)}((rt)) = (rs)(tu)$. □

Next, we characterize the behaviour of the conjugate operation, which is used to map permutations to transpositions.

Lemma 3.3 (Selection Lemma) *Let σ be a cycle of the form $(rs\ldots tu\ldots)$ where the ellipses $(\ldots)$ represent zero or more additional points within the cycle. Then $\sigma(su)\sigma^{-1} = (rt)$.*

PROOF. We can rewrite σ as:

$$\begin{aligned}
\sigma &= (rs\ldots tu\ldots) \\
&= (rs\ldots t)(ru\ldots) \\
&= (trs\ldots)(ru\ldots) \\
&= (trs)(t\ldots)(ru)(r\ldots) \\
&= (trs)(ru)(t\ldots)(r\ldots) \\
&= (turs)(t\ldots)(r\ldots).
\end{aligned}$$

Setting $\rho = (t\ldots)(r\ldots)$, and rewriting the conjugate

$$\sigma(su)\sigma^{-1} = (turs)\rho(su)\rho^{-1}(tsru) = (turs)(su)(tsru) = (rt),$$

yields the result. □

Corollary 3.4 *If σ is a permutation that is represented by the cycle $(r\,s\,t\ldots)$, then $\sigma(s\,t)\sigma^{-1} = (r\,s)$.*

To construct the mechanism for "checking" the adjacent order of the points in a permutation, we use Corollary 3.4 in conjunction with Lemma 3.2. To verify that the permutation is $(1\,2\ldots w)$, w "checks" must be performed, to ensure that 1 follows w, 2 follows 1, and so on. This is formalized in the following theorem.

Theorem 3.5 *Let $w > 4$ and let $\pi_0 = (1\,2\ldots w) \in S_w$. There exists a width-$w$ program transformation Ψ_{π_0} that is characterized by the map that sends every element except π_0 to ε and sends π_0 to some nonidentity element.*

PROOF. Let $\Xi_r = P(s\,t)P^{-1}$, $r = 1\ldots w$, be the conjugation transformation where $s = (r \bmod w) + 1$ and $t = (s \bmod w) + 1$, i.e., rst corresponds to a triple of points in permutation π_0.

Without loss of generality, assume that the characteristic map of $\Upsilon_{(r\,s)}$ maps $(r\,s)$ to $(r\,s)(t\,u)$, where $u = (t \bmod w) + 1$. Compose the Ξ_r transformation with $\Upsilon_{(r\,s)}$ to construct the adjacency checking transformation $\Phi_r(P) = \phi_r\left(\Upsilon_{(r\,s)}(\Xi_r(P))\right)\phi_r^{-1}$, where $\phi_r = (1\,r)(1\,2)(1\,s)(1\,4)(1\,t)(1\,5)(1\,u)$. Conjugating by ϕ_r is characterized by a map that sends ε to ε and sends $(r\,s)(t\,u)$ to $(1\,2)(4\,5)$.

Finally, Ψ_{π_0} is a combination of the transformations Φ_r, $r = 1\ldots w$. Define the recursive commutator of an ordered set of permutations to be

$$[\{\rho_1, \rho_2, \ldots, \rho_k\}] = [\rho_1, [\{\rho_2, \ldots, \rho_k\}]],$$

where $[\{\rho_{k-1}, \rho_k\}] = [\rho_{k-1}, \rho_k]$. Since $[(1\,2)(4\,5), (1\,2\,3)] = (1\,2\,3)$, the construction of Ψ_{π_0} is the recursive commutator of the set $\{\Phi_1, \Phi_2, \ldots, \Phi_w, (1\,2\,3)\}$,

$$\Psi_{\pi_0} = [\{\Phi_1(P), \Phi_2(P), \ldots, \Phi_w(P), (1\,2\,3)\}].$$

The characteristic map of Ψ_{π_0} sends π_0 to $(1\,2\,3)$ and sends all other permutations to ε. If the permutation is not π_0 then the characteristic maps of at least one Φ_i, $i = 1\ldots w$ will send the permutation to the identity—the recursive commutator of the yields of the Φ_is yields the identity. On the other hand, if all Φ_is yield $(1\,2)(4\,5)$, the yield of the recursive commutator is $(1\,2\,3)$. Finally, note that the size of the transformation Ψ_{π_0} only depends on w. Hence, the transformation increases the program length by a constant factor. $\square$

Corollary 3.6 *The transformation $\Psi_\pi = \Psi_{\pi_0}(P\pi^{-1}\pi_0)$ of width-w programs, $w > 4$, is characterized by the map that sends every element except π to ε and sends π to $(1\,2\,3)$.*

Using the preceding corollary we can now construct the transformation Ψ that transforms every weakly accepting program into a strongly accepting one.

Theorem 3.7 *If P is a weakly accepting program over S_w, $w > 4$, which computes a Boolean function f, then $\Psi(P)$ strongly computes Boolean function f, where transformation $\Psi = \prod_{\pi \in S_w \setminus \varepsilon} \Psi_\pi(P)$.*

PROOF. Each of the Ψ_π transformations transforms P into a program that yields $(1\,2\,3)$ if and only if P yields π. Hence, if P yields a nonidentity, exactly one of the transformed programs will yield $(1\,2\,3)$ and the rest will yield ε. Thus, for any input x, $\Psi(P)$ yields $(1\,2\,3)$ if and only if P does not yield ε; $\Psi(P)$ yields ε otherwise. The length of $\Psi(P)$ is $O(|P|)$ and the constant depends only on w. $\square$

Thus, the computational power of weakly accepting and strongly accepting programs of width-5 or greater is equivalent. This implies that there is a natural distinction between the power of 4-PBPs and w-PBPs, $w > 4$.

Corollary 3.8 *If P is a weakly accepting w-PBP that cannot be transformed into a strongly accepting w-PBP, then $w < 5$.*

4 Discussion

In this paper we demonstrated a natural gap between the computational power of width-4 and width-5 permutation branching programs. We defined a transformation that comprises a natural set of operations on PBPs, treating the programs as a black box. We showed that all weakly accepting programs of width-w, $w > 4$, can be transformed into strongly accepting width-w programs, while some weakly accepting width-4 programs cannot be transformed into strongly accepting width-4 programs.

This result is in line with the result of Barrington et al. [6]. If all weakly accepting programs could be transformed into strongly accepting ones, then the conjunction on n variables could be computed by a polynomial length program over the group A_4, which is a contradiction.

In fact, this impossibility holds even if we consider an extended transformation that has an extraction operation—defined as an oracle that given a program P, (nondeterministically) returns a subsequence of instructions from P. Namely, Theorem 3.1 holds even if we use this extended transformation in place of our standard one. Thus, without adding nonoblivious instructions, there are weakly accepting programs that cannot be transformed into strongly accepting programs.

A direct implication of this is that if there exists a polynomial length 4-PBP that computes the conjunction, then its construction can not be a recursive—black box like—composition because such compositions can be mimicked by a transformation. While our result does not preclude some clever construction, which would have to be unique for each n, it certainly makes such a possibility seem even less likely.

References

[1] M. Ajtai, L. Babai, P. Hajnal, J. Komlós, P. Pudlák, V. Rödl, E. Szemerédi, and G. Turán. Two lower bounds for branching programs. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 30–38, May 1986.

[2] D. Barrington. Width-3 permutation branching programs. Technical Memo MIT/LCS/TM-293, Massachusetts Institute of Technology, Laboratory for Computer Science, 1985.

[3] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in $\mathbf{NC}^1$. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 1–5, 1986.

[4] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in $\mathbf{NC}^1$. *Journal of Computer System Sciences*, pages 150–164, 1989.

[5] D. Barrington and H. Straubing. Superlinear lower bounds for bounded-width branching programs. *Journal of Computer and System Sciences*, 50(3):374–381, June 1995.

[6] D. Barrington, H. Straubing, and D. Thérien. Non-uniform automata over groups. *Information and Computation*, 89(2):109–132, December 1990.

[7] D. Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *Journal of the ACM*, 35(4):941–952, October 1988.

[8] A. Chandra, M. Furst, and R. Lipton. Multi-party protocols. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 94–99, April 1983.

[9] K. Doerk and T. Hawkes. *Finite Soluble Groups*. Berlin: de Gruyter, 1992.

[10] C. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38:985–999, July 1959.

[11] W. Mauer and J. Rhodes. A property of finite simple non-abelian groups. *Proceedings of the AMS*, 16(3):552–554, June 1965.

[12] P. Pudlák. A lower bound on complexity of branching programs. In *Proceedings of the 11th Symposium on Mathematical Foundations of Computer Science*, volume 176 of *LNCS*, pages 480–489, September 1984.