Amortized Circuit Complexity, Formal Complexity Measures, and Catalytic Algorithms

Robert Robere[†]
McGill University
robere@cs.mcgill.ca

Jeroen Zuiddam[†]
Courant Institute, NYU and U. of Amsterdam
jzuiddam@nyu.edu

October 1, 2021

Abstract

We study the *amortized circuit complexity* of boolean functions. Given a circuit model $\mathcal{F}$ and a boolean function $f : \{0,1\}^n \to \{0,1\}$, the $\mathcal{F}$-amortized circuit complexity is defined to be the size of the smallest circuit that outputs m copies of f (evaluated on the *same* input), divided by m, as $m \to \infty$. We prove a general duality theorem that characterizes the amortized circuit complexity in terms of "formal complexity measures". More precisely, we prove that the amortized circuit complexity in any circuit model composed out of gates from a finite set is equal to the *pointwise maximum* of the family of "formal complexity measures" associated with $\mathcal{F}$. Our duality theorem captures many of the formal complexity measures that have been previously studied in the literature for proving lower bounds (such as formula complexity measures, submodular complexity measures, and branching program complexity measures), and thus gives a characterization of formal complexity measures in terms of circuit complexity. We also introduce and investigate a related notion of *catalytic circuit complexity*, which we show is "intermediate" between amortized circuit complexity and standard circuit complexity, and which we also characterize (now, as the best integer solution to a linear program).

Finally, using our new duality theorem as a guide, we strengthen the known upper bounds for *non-uniform catalytic space*, introduced by Buhrman et. al [BCK$^+$14] (this is related to, but not the same as, our notion of catalytic circuit size). Potechin [Pot17] proved that for any boolean function $f : \{0,1\}^n \to \{0,1\}$, there is a *catalytic branching program* computing $m = 2^{2^n - 1}$ copies of f with total size $O(mn)$ — that is, linear size per copy — refuting a conjecture of Girard, Koucký and McKenzie [GKM15]. Potechin then asked if the number of copies m can be reduced while retaining the amortized upper bound. We make progress on this question by showing that if f has degree d when represented as polynomial over $\mathbb{F}_2$, then there is a catalytic branching program computing $m = 2^{\binom{n}{\leq d}}$ copies of f with total size $O(mn)$.

[†]Part of this work was done while the authors were at the Institute for Advanced Study.

Contents

1. Introduction

One of the long-standing frontier problems in circuit complexity is to prove super-polynomial (indeed, even super-*cubic* [Hås98, Tal14]) lower bounds on the size of boolean formulas computing any explicitly given boolean function. Recall that a *boolean formula* is a simple type of boolean circuit that, starting with a list of input literals $x_1, x_2, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n$, applies a sequence of *fan-out-1* AND and OR gates to compute some target function $f : \{0,1\}^n \to \{0,1\}$. For any boolean function f we let $\mathsf{F}(f)$ be the size of the smallest boolean formula computing f, measured by the number of leaves (input literals) in the formula.

One of the classic techniques for proving lower bounds on boolean formula size is the use of *formal complexity measures* [Weg87, Khr72]. A formal complexity measure is a function

$$\mu : \{n\text{-bit boolean functions } f\} \to \mathbb{R}_{\geq 0}$$

satisfying the following two properties:

- μ is *monotone*[1] with respect to AND and OR, that is, for any boolean functions f, g,
$$\mu(f \wedge g) \leq \mu(f) + \mu(g),$$
$$\mu(f \vee g) \leq \mu(f) + \mu(g);$$
- μ is *normalized*, that is, for every input literal ℓ,
$$\mu(\ell) \leq 1.$$

An easy induction on formulas shows that $\mu(f) \leq \mathsf{F}(f)$ for any formal complexity measure μ. Moreover, formula complexity $\mathsf{F}(f)$ is *itself* a formal complexity measure, since the cost of building a minimal formula for $f \wedge g$ or $f \vee g$ is never any more than the cost of building f and g separately and then using the appropriate gate. Thus, if we let M_F denote the family of all formal complexity measures, then we have

$$\mathsf{F}(f) = \max_{\mu \in M_\mathsf{F}} \mu(f), \tag{1}$$

and so formal complexity measures are a *complete* method for proving lower bounds against boolean formulas, for the simple reason that F is an element of M_F.

Given that these measures give a complete lower bound method for formulas (and also given the general lack of techniques for proving lower bounds against most boolean circuit models), it is natural to wonder about "formal complexity measures" for *other* boolean circuit models. Unfortunately, for many of these measures it is known that they do *not* characterize the complexity of computation in the corresponding models. For example:

[1] It will become clear soon why "monotone" is the appropriate terminology here rather than "subadditive".

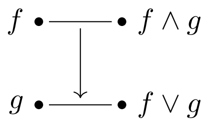


Figure 1: A comparator gate.

- The *submodular complexity measures*, introduced by Razborov [Raz92], map boolean functions to non-negative real numbers and satisfy the following constraints, for all boolean functions f, g and all input literals ℓ:

$$\mu(f \wedge g) + \mu(f \vee g) \leq \mu(f) + \mu(g)$$
$$\mu(\ell) \leq 1.$$

  These can be shown to lower bound the size of *comparator circuits* [RPRC16], which are circuits composed out of *comparator gates* that take in two bits a, b and output the bits in sorted order (see Figure 1). By a clever, though ad-hoc, proof Razborov showed that $\mu(f) = O(n)$ for every boolean function f depending on n variables and every submodular complexity measure μ. On the other hand, boolean functions with exponential comparator circuit complexity exist by counting arguments.

- The *branching complexity measures*, introduced by Potechin [Pot17], map boolean functions to non-negative real numbers and satisfy the following constraints[2], again for all boolean f, g and input variables x_i:

$$\mu(f \wedge x_i) + \mu(f \wedge \overline{x}_i) \leq \mu(f) + 2$$
$$\mu(f \vee g) \leq \mu(f) + \mu(g)$$
$$\mu(x_i) + \mu(\overline{x}_i) \leq 2$$

  An easy induction shows that these measures lower bound *deterministic branching program size*, although, in the same work Potechin also proved that $\mu(f) = O(n)$ for any branching complexity measure μ and any boolean function f depending on n variables by proving new upper bounds on *nonuniform catalytic space* (which we will discuss more later). On the other hand, boolean functions with exponential deterministic branching program complexity exist by counting arguments.

Thus we are at an impasse. We have several natural circuit families — formulas, comparator circuits, and deterministic branching programs — and natural formal complexity measures associated with them. For one family (formulas) we have that the measures completely

[2]Potechin used a slight variant of our definition (including $\mu(\ell) \leq 1$ for each literal ℓ). Our definition has the benefit of lower bounding the number of "wires" quite tightly.

characterize the complexity of computation in the family; for the others, only $O(n)$ bounds are possible, and the proofs of the $O(n)$ bounds are quite different from one another in character.

1.1. Our Results

In the present paper, we introduce an abstract framework that explains the above discrepancies. For essentially any boolean circuit model $\mathcal{F}$, we describe the family of "formal complexity measures" $M_{\mathcal{F}}$ associated with $\mathcal{F}$, and then characterize the quantity

$$\max_{\mu \in M_{\mathcal{F}}} \mu(f)$$

for any boolean function f. Looking ahead, it turns out that formal complexity measures μ (like those defined above for formulas, comparator circuits, and branching programs) do not capture standard circuit complexity, but rather capture what we call the *amortized* circuit complexity $\underset{\sim}{\mathcal{F}}(f)$: this is the quantity

$$\underset{\sim}{\mathcal{F}}(f) := \lim_{m \to \infty} \frac{\mathcal{F}(m \cdot f)}{m}$$

where $\mathcal{F}(m \cdot f)$ represents the cost of computing m copies of f (on the *same* input) by $\mathcal{F}$-circuits. Our main result, roughly speaking, is the following duality for amortized complexity:

Theorem 1.1 (Main Duality Theorem, Rough)**.** *Let G be any finite gate set, let $\mathcal{F}$ be any family of boolean circuits composed from G gates, and let $M_{\mathcal{F}}$ be the set of all $\mathcal{F}$-formal complexity measures. Then for any boolean function $f : \{0,1\}^n \to \{0,1\}$*

$$\underset{\sim}{\mathcal{F}}(f) = \max_{\mu \in M_{\mathcal{F}}} \mu(f).$$

Let us place Theorem 1.1 in context with the earlier results and models. First, note that it is easy to see that amortized formula complexity is exactly the same as formula complexity. Since all gates in a formula have fan-out 1, the only way to compute m copies of any function is to take m independent boolean formulas (i.e. a *forest* of formulas), and so it follows that $\mathsf{F}(m \cdot f) = m\mathsf{F}(f)$ and thus $\underset{\sim}{\mathsf{F}} = \mathsf{F}$. Indeed, for formulas, formula size is *already* a formal complexity measure, as discussed above.

However, in general, the amortized complexity $\underset{\sim}{\mathcal{F}}$ will not be an element of the feasible region $M_{\mathcal{F}}$. In other words, the maximizing choice of μ will generally depend on f. Indeed, this will be the case for the models that we will consider in detail (comparator circuits and deterministic branching programs).

We also note that if $\mathcal{F}$ is the model of *general* boolean circuits (i.e. circuits composed of $\wedge$ and $\vee$ gates with arbitrary fan-out) then it is easy to see that $\underset{\sim}{\mathcal{F}}(f) = O(1)$, since we can simply compute a single copy of f and fan it out arbitrarily many times. Thus, "formal boolean circuit measures", which have not been formally defined in the literature (although, it would be fairly easy to do so), are useless for proving lower bounds.

For comparator circuits, we have mentioned that Razborov proved that submodular complexity measures are all upper bounded by $O(n)$ [Raz92]. Thus, as an immediate corollary of our duality theorem, we have:

Corollary 1.2. *For any boolean function* $f : \{0,1\}^n \to \{0,1\}$,

$$\underset{\sim}{\mathsf{CC}}(f) = O(n),$$

where $\underset{\sim}{\mathsf{CC}}(f)$ *is the amortized comparator circuit size of computing* f.

This result is remarkable for two reasons. First, comparator circuits *cannot* copy intermediate computations [Sub90], and so there is no way to create many copies of f from one copy. Second, if we restrict to *monotone* comparator circuits (i.e. no negated input literals are allowed), then also an analogue of our duality theorem (Theorem 1.1) applies and in this case a submodular measure due to Razborov (the *rank measure* [Raz92]) *can* be used to prove strongly exponential lower bounds [RPRC16, PR17, PR18], and so these lower bounds already apply to amortized monotone comparator circuit size $\mathsf{m}\underset{\sim}{\mathsf{CC}}$. We believe that this new distinction between monotone and non-monotone computation interesting, and conjecture that *amortized* monotone comparator circuit size $\mathsf{m}\underset{\sim}{\mathsf{CC}}$ is simply equal to monotone comparator circuit size mCC.

Conjecture 1.3. *For any monotone boolean function* $f : \{0,1\}^n \to \{0,1\}$,

$$\mathsf{mCC}(f) = \mathsf{m}\underset{\sim}{\mathsf{CC}}(f).$$

Finally, we can apply our duality theorem (Theorem 1.1) to branching program measures and obtain a linear upper bound on amortized branching program complexity similar to Corollary 1.2. However, in this setting the story becomes more interesting. Namely, Potechin [Pot17] introduced a notion of "amortized branching program complexity" that is more restrictive than our own (in that, upper bounds on Potechin's version imply upper bounds on our version, but not *vice versa*). Indeed, we are unable to apply the main duality theorem (Theorem 1.1) to Potechin's definition of "amortized branching programs", and since our definition is slightly more general and satisfies the duality theorem we believe it is the "right" one in our context. However, Potechin's definition is very closely related to the notion of *non-uniform catalytic space* [BCK$^+$14] and, using our duality theorem as a guide, we can make some significant improvements in that area. We describe these improvements next.

Improved Upper Bounds for Catalytic Space. Let us recall the notion of catalytic space, introduced by Buhrman, Cleve, Koucký, Loff and Speelman [BCK$^+$14]. A *catalytic space Turing Machine* is a TM that comes equipped with a special auxiliary "catalytic tape" that, at the beginning of the computation, is filled with some "junk bits". The Turing Machine is then free to use this auxiliary tape for free, as long as it restores the catalytic tape to its original configuration at the end of the computation. Notably, the Turing Machine does *not* know the contents of the catalytic tape before it starts its computation, and thus it is quite surprising that Buhrman et al. [BCK+14] could show that a catalytic tape *can* actually help in computation.

We will be interested in the *non-uniform* version of catalytic space Turing Machines, introduced by Buhrman et al. [BCK+14] and further studied by Girard, Koucký, and Mckenzie [GKM15] and Potechin [Pot17]. These are called *m-catalytic branching programs*, and are defined as follows. Given a boolean function $f : \{0,1\}^n \to \{0,1\}$, we construct a branching program with m start nodes $s_1, s_2, \ldots, s_m$, m accept nodes $a_1, a_2, \ldots, a_m$, and m reject nodes $r_1, r_2, \ldots, r_m$, with the following special property. On any input $x \in \{0,1\}^n$, if $f(x) = 1$ then for each $i \in [m]$, the computation path starting at s_i ends at a_i. On the other hand, if $f(x) = 0$, then for each $i \in [m]$, the computation path starting at s_i ends at r_i. Clearly, if we have a branching program of size s computing f once, then we can create an m-catalytic branching program for f with size $O(ms)$, simply by repeating the branching program independently m times.

Potechin's definition of amortized branching program complexity used m-catalytic branching programs, and measured the average size of an m-catalytic program for f as $m \to \infty$. We emphasize that our above duality theorem (Theorem 1.1) *does not* apply to catalytic branching programs directly. It does indeed apply to branching programs, however, it turns out that the "right" model of amortized branching programs for proving the duality theorem is slightly less restrictive in that it does not require *pairing property* between source nodes and sink nodes. Rather, in our notion, on an input x a computation path starting at s_i can end up at any sink node.

For m-catalytic branching programs, Girard, Koucký, and McKenzie [GKM15] asked the following question:

Question. For which boolean functions $f : \{0,1\}^n \to \{0,1\}$ is the size of the smallest m-catalytic branching program for f much smaller than $O(ms)$, where s is the size of the smallest branching program for f?

In a surprising result, Potechin [Pot17] proved that *any* function $f : \{0,1\}^n \to \{0,1\}$ is computable by an m-catalytic branching program of size $O(mn)$ — that is, *linear* size per copy — with the caveat that m is enormous (doubly exponential in n). Potechin then asked whether or not it is possible to reduce the number of copies m in the construction. In our second major contribution, we show that is possible to reduce the number of copies significantly whenever f has low degree as an $\mathbb{F}_2$-polynomial.

Theorem 1.4. *Let $f = \{0,1\}^n \to \{0,1\}$, and let $d = \deg_2(f)$ be the degree of f as a polynomial over $\mathbb{F}_2$. Then there is an m-catalytic branching program computing f in total size $O(mn)$ with $m = 2^{\binom{n}{\leq d}}$.*

While our duality theorem does not immediately imply this theorem, it played an important role in discovering its proof, which we will discuss in our technical overview (cf. Section 1.2).

Amortized Complexity and Catalytic Algorithms. We finally discuss one more new result, which is morally related to both catalytic space and amortized complexity. Suppose that μ is a formal complexity measure, and suppose that we had proved the inequality $\mu(f) \leq \mu(g)$. Then, clearly, it also holds that $\mu(f) + \mu(h) \leq \mu(g) + \mu(h)$ for any boolean function h. Or, said another way: when presented with the inequality $\mu(f) \leq \mu(g)$, how do we know that we didn't first prove $\mu(h) + \mu(f) \leq \mu(h) + \mu(g)$ and then cancel $\mu(h)$ from each side? This phenomenon turns out to be intimately related to a notion of *catalysts* in algorithms. That is, we can consider, say, a comparator circuit C which, besides the usual literals, receives a boolean function $h : \{0,1\}^n \to \{0,1\}$ as an extra input, and outputs a "fresh" copy of h at the end of its computation (thus, h acts as a "catalyst", in that it enables the computation but is not consumed by it).

From this observation, we are naturally led to define the *catalytic size* of a circuit family $\mathcal{F}$, denoted $\mathcal{F}_{\text{cat}}$. This is the size of the smallest $\mathcal{F}$-circuit computing a function f for which there exists a multiset of boolean functions A that the circuit can take as input for free, as long as it outputs another copy of A at the end of the computation. As we will see later, it can be shown that

$$\underset{\sim}{\mathcal{F}}(f) \leq \mathcal{F}_{\text{cat}}(f) \leq \mathcal{F}(f)$$

for any circuit model $\mathcal{F}$, and in fact we will be able to give a *characterization* of $\mathcal{F}_{\text{cat}}$ as the *integral optimum* of the linear program that we construct for proving our main duality theorem. See Section 4 (and, in particular, the proof of Theorem 4.2) for details.

1.2. Technical Overview

Amortized and Strassen Duality. Our main duality theorem was inspired by the theory of *Strassen Duality*, which we first recall. Consider the problem of determining the minimum value $\omega \in \mathbb{R}$ such that any two $n \times n$ matrices can be multiplied together using $O(n^\omega)$ arithmetic operations (i.e. the *matrix multiplication exponent*). This is one of the central open problems in algorithms and computational complexity as many algorithms use matrix multiplication as a subroutine. It is known that $\omega \geq 2$, since by an independence argument we must read $\Omega(n^2)$ input values, while a series of algorithms has improved the upper bound from $\omega \leq 3$ all the way to $\omega \leq 2.372\ldots$ [LG14, AW21]. At a first glance, this problem does not have an "amortized" flavour; but, as explored in [Str86], following [Gar85], it turns out that ω is exactly determined by the *asymptotic tensor rank* of a certain tensor T: that is

$$2^\omega = \underset{\sim}{\mathrm{R}}(T) := \lim_{m \to \infty} \mathrm{R}(T^{\otimes m})^{1/m}$$

where $T^{\otimes m}$ is the m-fold tensor product of T and R is the tensor rank.

In order to study the asymptotic tensor rank (with the goal of understanding the matrix multiplication exponent ω), Strassen introduced a beautiful and very general duality theory for pre-orders on semirings [Str86, Str87, Str88] which is now termed *Strassen Duality* and an active area of study [CVZ18, Zui18, Fri20, Vra20]. In the special case of tensors, the essence of the theory is as follows. Given two 3-tensors $A : \mathbb{F}^{n_1} \times \mathbb{F}^{n_2} \times \mathbb{F}^{n_3} \to \mathbb{F}$ and $B : \mathbb{F}^{m_1} \times \mathbb{F}^{m_2} \times \mathbb{F}^{m_3} \to \mathbb{F}$ we say that A *restricts* to B, written $A \geq_T B$, if there are linear maps $L_i : \mathbb{F}^{m_i} \to \mathbb{F}^{n_i}$ such that $B = A \circ (L_1, L_2, L_3)$. (This is the 3-tensor equivalent of multiplying a matrix on the left and right by a matrix.) The ordering $\leq_T$ is known as a *pre-order*, as it satisfies the axioms of a partial order except for antisymmetry.

One of the main outcomes of Strassen's duality theory is that determining the asymptotic rank of tensors is essentially equivalent to the problem of understanding the family of *all* functions $\mu : \{\text{tensors}\} \to \mathbb{R}_{\geq 0}$ such that

- μ is *multiplicative* under tensor product $\otimes$, and *additive* under direct sum $\oplus$,
- μ is *monotone* under $\leq_T$, that is if $A \leq_T B$ then
$$\mu(A) \leq \mu(B),$$
- μ is *normalized*, so that at the diagonal tensor $\langle n \rangle$ we have
$$\mu(\langle n \rangle) \leq n.[3]$$

These maps μ are called *spectral points* (and are the clear analogues of formal complexity measures), and the set of all such maps X is called the *asymptotic spectrum* of tensors. One of the main results of Strassen's duality theory (specialised to tensors) states that:

Theorem 1.5 (Strassen's Duality Theorem). *For any tensor T,*

$$\underset{\sim}{\mathrm{R}}(T) = \max_{\mu \in X} \mu(T).$$

In other words: understanding the matrix multiplication exponent is *equivalent* to understanding $\max_{\mu \in X} \mu(T)$ for the special *matrix multiplication tensor* T.

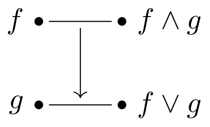
Circuits as Pre-Orders and Strassen Duality for Semigroups. Let us now describe in more detail how our duality theorem is related to Strassen Duality. Our main technical theorem (Theorem 4.5, which implies Theorem 1.1) can morally be interpreted as a special case of Strassen's theory where the underlying algebra is a *semigroup* equipped with a

[3]Strassen formally puts the stronger normalization $\mu(\langle n \rangle) = n$, which, for the purposes discussed here, is equivalent to requiring $\mu(\langle n \rangle) \leq n$.

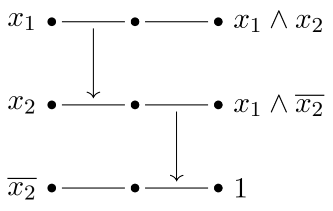
pre-order instead of a *semiring*. We now give a brief introduction to describing circuit models as pre-orders over a semigroup; a more detailed description is in Section 2.

This is best understood by example, so, let us consider the model of comparator circuits. In general, to fix some notation, we will use $\{\{\}\}$ to denote multisets, and we will consider circuits with multiple outputs as computing a multiset of boolean functions. Fix a positive integer n, and let F_n denote the set of all boolean functions on n input bits. If A, B are multisets of functions from F_n, then we write $A + B$ to denote the disjoint union of A and B as a multiset (in particular, we count multiplicities of functions). The collection of all multisets, equipped with disjoint union, forms a *semigroup*, since $+$ is associative and has an identity (the empty multiset $\{\{\}\}$), but elements do not have inverses.

A comparator circuit is a simple circuit model with a single gate, the *comparator gate*, that takes two inputs $(a, b) \in \{0, 1\}^2$ and outputs the two bits $(a \wedge b, a \vee b) \in \{0, 1\}^2$. The inputs of the comparator circuit are then labelled with boolean literals x_i or $\neg x_j$ for some input variables x_i, x_j. For example, if we draw the comparator gate as



then we can compute the collection of functions $\{\{x_1 \wedge x_2, x_1 \wedge \overline{x_2}, 1\}\}$ via the following comparator circuit of size three:



The *comparator circuit size* of a multiset of boolean functions A is the smallest number of literals in any comparator circuit computing all functions in A among its outputs (with multiplicities), and is denoted by $\mathsf{CC}(A)$. We note that, unlike formulas, comparator circuit size is *not* additive: the above circuit shows that it is possible to compute both $x_1 \wedge x_2$ and $x_1 \wedge \overline{x}_2$ using only three input literals (thus, $\mathsf{CC}(\{\{x_1 \wedge x_2, x_1 \wedge \overline{x}_2\}\}) \leq 3$), but it is also easy to see that $\mathsf{CC}(x_1 \wedge x_2) = \mathsf{CC}(x_1 \wedge \overline{x}_2) = 2$ since any comparator circuit must read both of the input literals. Thus $\mathsf{CC}(A + B) \neq \mathsf{CC}(A) + \mathsf{CC}(B)$ in general, and this means that comparator circuits can compute collections of boolean functions more efficiently than just creating a comparator circuit for each function individually and using them in parallel. This fact means that *amortized* comparator circuit size is meaningful, so, for any function f we let $\utilde{\mathsf{CC}}(f) := \lim_{m\to\infty} \mathsf{CC}(m \cdot f)/m$ be the *amortized comparator circuit size*, where $m \cdot f = \{\{f, \ldots, f\}\}$ is the multiset in which f appears m times. Clearly we have $\utilde{\mathsf{CC}}(f) \leq \mathsf{CC}(f)$.

Following Strassen, we can describe comparator circuit computation as an *ordering* over multisets of functions. Namely, let S denote the collection of all multisets over $F_n \cup \{\mathbf{1}\}$, where $\mathbf{1}$ is a special symbol representing a “unit cost”.[4] We define an ordering $\preceq_{\mathrm{CC}}$ on S as follows. First, if $f, g \in F_n$ then we have

$$\{\{f \wedge g, f \vee g\}\} \preceq_{\mathrm{CC}} \{\{f, g\}\},$$

and second, if ℓ is a boolean literal then we have

$$\{\{\ell\}\} \preceq_{\mathrm{CC}} \{\{\mathbf{1}\}\}.$$

We then extend the ordering to all multisets in the natural way (i.e. if S is any multiset, then $S + \{\{f \wedge g, f \vee g\}\} \preceq_{\mathrm{CC}} S + \{\{f, g\}\}$, and similarly for the rule for boolean literals), and we allow “forgetting” rule, which states that $A \subseteq B \implies A \preceq_{\mathrm{CC}} B$. From the definition it is now clear that $A \preceq_{\mathrm{CC}} B$ if and only there is a comparator circuit that, starting from the functions in B as inputs, produces all functions in A at the outputs; in particular, there is a comparator circuit of size s computing f if and only if $\{\{f\}\} \preceq_{\mathrm{CC}} \{\{\mathbf{1}, \mathbf{1}, \ldots, \mathbf{1}\}\}$, where the $\mathbf{1}$ is repeated s times.

Now we can easily describe the submodular complexity measures in an abstract way. They are simply the functions $\mu : S \to \mathbb{R}_{\geq 0}$ that are

- *Additive* with respect to multisets $A \in S$: if $A = B + C$ then $\mu(A) = \mu(B) + \mu(C)$ (in other words, μ is defined by its values at individual functions in F_n and $\mathbf{1}$).
- *Monotone* with respect to $\preceq_{\mathrm{CC}}$: if $A \preceq_{\mathrm{CC}} B$ then $\mu(A) \leq \mu(B)$.
- *Normalized* at $\mathbf{1}$: $\mu(\mathbf{1}) \leq 1$.

Now the analogy with Strassen Duality is apparent!

With this in hand, we can now loosely describe how to prove our duality theorem (cf. Theorem 1.1) in the special case of comparator circuits. The idea is fairly simple: we can interpret the quantity $\max_{\mu \in M_{\mathrm{CC}}} \mu(f)$ as a linear program and take its dual. Solutions to the dual program can then be interpreted as constructing comparator circuits computing *many* copies of f (and it is here that the pre-order perspective turns out to be indispensible). It follows by linear programming duality that we can characterize the maximum value of a submodular complexity measure in terms of amortized comparator circuit complexity. In fact, we are able to do something much stronger: we can fully characterize the quantity $\max_\mu \mu(A) - \mu(B)$, for any two multisets of boolean functions A, B.

Thanks to the above abstraction, we can easily generalize this approach to other circuit models simply by writing down the appropriate pre-order. In Section 3 we describe the general properties that such a pre-order must satisfy in order for our duality theorem to hold,

[4] The element $\mathbf{1} \in F$ (written in bold face) is a formal symbol and should not be confused with the constant boolean function $1 \in F_n$ (written in the normal font).

and our main technical duality theorem (Theorem 4.5) then applies to all such pre-order equipped semigroups simultaneously. It is remarkable that operating over semigroups seems to completely capture computation by many standard circuit models, and the restriction to semigroups from semirings means that the proof of duality can be made much simpler. Indeed, Strassen's proof of the duality theorem for semirings is quite complicated — building on Stone Duality and ultimately using the Axiom of Choice — while our proof is a relatively simple argument based on linear programming. An additional benefit of our linear programming proof is that our duality is *computable*, and it is not clear if this holds for Strassen's general duality theorem for semirings. We refer to Section 4 for details.

Exploiting Symmetry in Catalytic Space. We now describe how we prove our new upper bounds on catalytic branching programs (Theorem 1.4). As we have mentioned above, Razborov [Raz92] proved that any *submodular* complexity measure μ satisfies $\mu(f) = O(n)$ for any n-variable boolean function f. Razborov used a randomized construction which exploited[5] the following key symmetry property: if f_0, f_1 are both uniformly random boolean function on $n-1$ variables and f is a uniformly random boolean function on n variables, then

$$(\overline{x}_n \wedge f_0) \vee (x_n \wedge f_1) \sim (x_n \wedge f_0) \vee (\overline{x}_n \wedge f_1) \sim f$$

where by $X \sim Y$ we mean that the two random variables X and Y have the same distribution.

We first prove that if we *explicitly enforce* these symmetry properties, then we can strongly improve the upper bounds on μ, and indeed we can already do this for branching program complexity measures. Towards this, in the next theorem, by a *symmetric* formal complexity measure we mean a measure satisfying $\mu(f^{\oplus i}) \leq \mu(f)$ for every boolean function f and every $i \in [n]$, where $f^{\oplus i}$ is the boolean function obtained by negating the ith input to f.

Theorem 1.6. *For any* symmetric *branching program or submodular complexity measure μ and any boolean function f,*

$$\mu(f) \leq 2D_{avg}(f)$$

where $D_{avg}(f)$ is the minimum, over any decision tree computing f, of the expected number of queries made by the tree on uniform distribution of inputs.

Unfortunately, we cannot (yet) prove the above theorem for arbitrary branching program measures. However, by *symmetrizing* over the *orbit* $\mathsf{Orb}(f)$ of f under the action of negating any subset of inputs, it turns out that we *can* use the argument in the above theorem to compute all of $\mathsf{Orb}(f)$ efficiently on average by a small catalytic branching program (cf. Lemma 5.12). Then, by an extension of Potechin's construction (using the decision tree argument above), we show that it suffices to take $m = |\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))|$ in order to compute f by an m-catalytic branching program of total size $O(mn)$ (cf. Theorem 5.13), where $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ is the linear span of the functions in $\mathsf{Orb}(f)$ when treated as vectors

[5]Razborov actually exploited a very similar, though equivalent, property.

over $\mathbb{F}_2$. The above theorem then follows since if $\deg_2(f) = d$ then $\deg_2(g) \leq d$ for all $g \in \mathsf{Orb}(f)$ (as we are just negating input variables), and so we can bound the size of $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ by a dimension argument. See Section 5 for details.

1.3. Related Work

Closely related to our work is the study of *direct sum problems.* While appearing in many different forms, they are all usually variants of the following question:

> Is the best way of performing some computational task T many times in parallel simply to compute T independently each time, or, can we achieve an *economy of scale*, and perform all copies of T more efficiently on average?

Alternatively phrased, these types of problems study the *amortized complexity* of computation. Informally, if we are measuring the computational cost C of performing a task T, then in the direct sum problem we are interested in the behaviour of

$$\underset{\sim}{C}(T) := \lim_{m\to\infty} \frac{C(mT)}{m}$$

where mT represents the task of performing T in parallel m times[6]. If the cost C is *subadditive* — that is, $C(A+B) \leq C(A) + C(B)$ — then the above limit exists and is equal to its infimum by Fekete's Lemma[7], and we call the cost $\underset{\sim}{C}(T)$ the *amortized complexity* of T. This framework occurs independently in many different settings, such as:

- In *information theory*, classic results such as *Shannon's Source Coding theorem* and the *Slepian–Wolf theorem* characterize the *amortized communication* required when sending $m \to \infty$ independent copies of a random variable M over a communication channel (possibly with some side information) in terms of quantities such as the *entropy* and *mutual information.*

- In *communication complexity*, much work has been spent investigating the direct sum problem in various communication models. For example, in the model of *deterministic* communication complexity, Feder, Kushilevitz, Naor, and Nisan [FKNN95] showed that if a single copy of a function f requires C bits of communication, then the amortized cost of computing f in parallel $m \to \infty$ times is $\Omega(\sqrt{C})$ per copy. On the other hand, in *randomized* communication complexity — generalizing the example from information theory given above — amortized communication cost was shown by Braverman and Rao to be *exactly* equal to the so-called *information complexity* [BR14].

[6] Depending on the situation, it may be more convenient to think of m instances of T as either a "sum" or a "product" — we have stated amortized complexity for "sums", but in the latter case, we would rather be studying the value of $C(T^m)^{1/m}$ as $m \to \infty$.

[7] This useful basic lemma was first obtained in [Fek23], see for the proof [FR18].

- In *probabilistically checkable proofs* the direct sum problem for *2-prover games* is studied — these are games where a Verifier sends inputs x, y to two separate Provers, who must each send back messages to the Verifier without communicating with each other. Raz showed (in his now famous *parallel repetition theorem* [Raz98]), that the acceptance probability of the Verifier can be driven down exponentially by playing many games in *parallel* (this corresponds directly to a *tensor product* of games).

The aforementioned result of Potechin [Pot17] on catalytic branching programs, which we improve on in Theorem 1.4, has inspired several interesting works in the area of conditional disclosure of secrets (CDS) [AARV21, AA20]. These works include constructions in the framework of CDS that are analogous (but incomparable) to our construction for amortized branching programs in Theorem 1.4.

As we have mentioned above, the notion of catalytic space was introduced by Buhrman et al. [BCK+14], where they showed the surprising result that a logspace Turing Machine equipped with a catalytic tape can compute all of TC^1. These results are closely related to earlier results of Barrington [Bar89] and Ben-Or and Cleve [BC92] — for instance, it is easy to see that any width-w permutation branching program that cycle-computes a boolean function f in the sense of [Bar89] is also a catalytic branching program computing $w/2$ copies of f. Catalytic space algorithms have also recently seen application in improved algorithms for the *Tree-Evaluation Problem* in a recent paper of Cook and Mertz [CM20].

In combinatorics, the asymptotic spectrum of *graphs* was introduced in [Zui19] to characterize the Shannon capacity of graphs, which is an amortized combinatorial notion introduced by Shannon [Sha56] to understand the zero-error communication capacity of a noisy communication channel. Various extensions to the realm of quantum information were considered in [LZ21]. Jensen and Vrana employed Strassen's duality theorem to study the asymptotic properties of Local Operations and Classical Communication (LOCC) in quantum information theory [JV20].

Fritz introduced for the first time a semigroup version of the Strassen duality in [Fri17] (independent of the work of Strassen), and the reader will find our results to have similar flavour, albeit that we work in a more "finite" setting in several respects. His proof is based on the Hahn–Banach theorem, and so is also not computable. Extensions of the Strassen duality theory in several directions have been introduced in [Fri20] and [Vra20] recently.

1.4. Paper Organization

The rest of this paper is organized as follows. In Section 2, we give a detailed introduction to modelling boolean circuits as abstract semigroups equipped with pre-orders. Our introduction uses three running examples: *boolean formulas*, *branching programs*, and *comparator circuits*. (This section is expository, and it can be skipped to get to the main contributions of the paper.) In Section 3 we describe the abstract framework for our duality using semigroups with finitely generated pre-orders. In Section 4 we prove our main duality theorem, as well as our theorem characterizing "catalytic size". In Section 5 we give our new upper bounds on amortized complexity and catalytic space by exploiting symmetry.

2. Circuit Models as Pre-Orders

In this section we give an extended introduction to the paradigm of describing circuit models as *pre-orders*. This section is mainly included for expository purposes, to connect the "concrete" circuit models with the "abstract" approach using pre-orders that we will use in the proof of our duality theorem. Readers who feel comfortable with this approach can jump to Section 3.

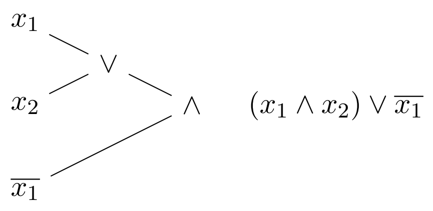
Definition 2.1. A *pre-order* on a set S is a relation $P \subseteq S^2$ that is *reflexive*, in that $(a, a) \in P$ for every $a \in S$, and *transitive* in that $(a, b) \in P$ and $(b, c) \in P$ implies $(a, c) \in P$ for all $a, b, c \in S$. If P is a pre-order then we will use the notation $a \leq_P b$ whenever $(a, b) \in P$.

Pre-orders are not unfamiliar to theoretical computer science—they appear prominently in various settings in the form of *reductions*. Here we will also think of pre-orders as reductions, but with a circuit-building flavour. In this introduction we will be guided by three concrete and familiar circuit models: boolean formulas, branching programs and comparator circuits. After having discussed those in some detail, we discuss an approach that generalizes the concrete instances.

2.1. Boolean Formulas

Throughout, let $x_1, x_2, \ldots$ denote variables that take boolean values, and let $\overline{x_1}, \overline{x_2}, \ldots$ denote their negations. As usual, by a *literal* we mean any variable or any negated variable. Let $F = F_n$ denote the set of all boolean functions $\{0, 1\}^n \to \{0, 1\}$. Since we will generally be interested in computing collections of boolean functions, we let $\mathbb{N}^F$ denote the set of all collections of boolean functions, that is, all multisets of elements of F, encoded as a vector of multiplicities. For multisets we use the notation $\{\{f_1, \ldots, f_n\}\}$.

A *boolean formula* is a tree whose leaf nodes are labelled by literals and whose internal nodes are labelled by $\wedge$ (the logical AND) or $\vee$ (the logical OR). In other words, a boolean formula is a circuit that computes a boolean function from input literals using two types of gates—the AND gate, and the OR gate—which both have fan-out one. The *size* of a boolean formula is the number of leaf nodes, and the *formula size* of a boolean function f is the size of the smallest formula computing it, denoted $\mathsf{F}(f)$. For example, the following boolean formula of size three computes the boolean function $(x_1 \wedge x_2) \vee \overline{x_1}$:



A natural and standard concept is the notion of a *formal complexity measure*. This is any function $\mu : F \to \mathbb{R}_{\geq 0}$ such that for any literal ℓ and any $f, g \in F$:

$$\begin{aligned} \mu(\ell) &\leq 1 \\ \mu(f \vee g) &\leq \mu(f) + \mu(g) \\ \mu(f \wedge g) &\leq \mu(f) + \mu(g) \end{aligned}$$

By induction over the tree-structure of the boolean formula we obtain the important property that for any boolean function f and any formal complexity measure μ, the formula size of f is at least $\mu(f)$,

$$\mu(f) \leq \mathsf{F}(f).$$

We also note that the formula size F *satisfies* these three properties, and so it itself is also a formal complexity measure.

From the definition of boolean formulas it is immediate that a single boolean formula can only compute a single boolean function. To compute a *collection* of boolean functions $\{\{f_1, \ldots, f_m\}\}$ we thus need a collection of boolean formulas, that is, a forest of trees with leaf nodes labelled by literals and internal nodes labelled by $\vee$ or $\wedge$. Therefore, we must define the formula size of a multiset as the sum of the formula size of each element, $\mathsf{F}(\{\{f_1, \ldots, f_m\}\}) = \sum_{i=1}^{m} \mathsf{F}(f_i)$. In particular, if we define the *amortized formula size* of a boolean function f by $\underset{\sim}{\mathsf{F}}(f) := \lim_{m \to \infty} \mathsf{F}(m \cdot f)/m$, where $m \cdot f := \{\{f, \ldots, f\}\}$ is the multiset in which f appears m times, then we find that

$$\mu(f) \leq \underset{\sim}{\mathsf{F}}(f) = \mathsf{F}(f).$$

The above discussion of computing a collection of boolean functions by formulas should feel like a strange exercise—clearly, formulas cannot compute collections of boolean functions in an *interesting* way. However, this simple treatment of boolean formulas prepares us for the discussion of the richer circuit models that are to follow, in which amortization *does* allow for clever algorithms. Before going to these other cicuit models, we introduce a *pre-order* point of view for boolean formula computation, and relate it to the formal complexity measures.

Definition 2.2. Let $\mathbf{1}$ be a formal symbol, which we will call the *unit* and which will function as our measure of "cost". Let $S = \mathbb{N}^{F \cup \{\mathbf{1}\}}$ be the set of all collections consisting of elements of F or the element $\mathbf{1}$. Define the ordering $\preceq_{\mathrm{F}}$ on S as follows. First, for any boolean functions f, g define

$$\{\{f \wedge g\}\} \leq \{\{f, g\}\}, \quad \{\{f \vee g\}\} \leq \{\{f, g\}\}.$$

Second, for any literal ℓ, let $\{\{\ell\}\} \leq \{\{\mathbf{1}\}\}$. Finally, let $\preceq_{\mathrm{F}} \supseteq \leq$ be the smallest pre-order containing $\leq$ that satisfies the following properties:

- *Forgetting.* If $A \subseteq B$ is an inclusion of multisets, then $A \preceq_{\mathrm{F}} B$.
- *Parallel Computation.* If $A \preceq_{\mathrm{F}} B$ then $A + C \preceq_{\mathrm{F}} B + C$.
- *Subroutine Composition.* If $A \preceq_{\mathrm{F}} B$ and $B \preceq_{\mathrm{F}} C$ then $A \preceq_{\mathrm{F}} C$.

The intended interpretation of $\preceq_{\mathrm{F}}$ is that it "builds" boolean formulas. The way to read the inequality $\{\{f \wedge g\}\} \preceq_{\mathrm{F}} \{\{f, g\}\}$, for example, is that $f \wedge g$ can be *obtained from* f and g using a boolean formula gate. The inequality $\{\{\ell\}\} \preceq_{\mathrm{F}} \{\{\mathbf{1}\}\}$, on the other hand, should be read as indicating that we have literals available at cost one. Finally, the three generating rules respectively correspond to (1) allowing (forests of) formulas to *forget* outputs in the computation, (2) to allow multiple disjoint formulas to do *parallel computation*, and to (3) allow formulas to be used in new formulas as *subroutines.*

The point of defining the pre-order $\preceq_{\mathrm{F}}$ is that for any $f \in F$ and literals ℓ_i we have $\{\{f\}\} \preceq_{\mathrm{F}} \{\{\ell_1, \ldots, \ell_m\}\}$ if and only if f can be computed by a formula whose leaf nodes are labelled by $\ell_1, \ldots, \ell_m$. More generally, $\{\{f_1, \ldots, f_m\}\} \preceq_{\mathrm{F}} \{\{g_1, \ldots, g_k\}\}$ if and only if the boolean functions $f_1, \ldots, f_m$ can be computed *from* the boolean functions $g_1, \ldots, g_k$ by a boolean formula (or rather by a "generalized" boolean formula in which the leaf nodes are labeled by $g_1, \ldots, g_k$ instead of literals). Thus, it should be clear that for a multiset of functions A, $\mathsf{F}(A)$ is exactly the minimum number n such that $A \preceq_{\mathrm{F}} \mathbf{n} := \{\{\mathbf{1}, \ldots, \mathbf{1}\}\}$, where $\mathbf{1}$ appears n times.

How does the pre-order $\preceq_{\mathrm{F}}$ relate to the formal complexity measures μ that we defined earlier? We can extend the formal complexity measures μ to functions $\mathbb{N}^F \to \mathbb{R}_{\geq 0}$ by saying, for any multiset $A \in S$, that $\mu(A) = \sum_{f \in A} \mu(f)$. These functions are then precisely the functions $S \to \mathbb{R}_{\geq 0}$ that are *monotone* under $\preceq_{\mathrm{F}}$ and *additive* under taking the addition of multisets, where we define addition on multisets as the disjoint union

$$\{\{a_1, \ldots, a_m\}\} + \{\{b_1, \ldots, b_k\}\} := \{\{a_1, \ldots, a_k, b_1, \ldots, b_k\}\},$$

which matches the notation $\mathbb{N}^F$ that we use for multisets of elements of F.

Finally, we point out four good properties that the pre-order $\preceq_{\mathrm{F}}$ has. These properties will turn out to be a sufficient condition in the main technical duality theorem of this paper, and will reoccur in the other circuit models.

- First, the pre-order $\preceq_{\mathrm{F}}$ is *finitely generated*, in the (to be made precise) sense that only finitely many rules are needed to generate the pre-order. Intuitively this corresponds to the circuit model having a finite gate set.

- Second, the pre-order is *bounded*, in the sense that every boolean function has a finite rank. This corresponds to the circuit model being *complete*, in that it can compute any boolean function.
- Third, the pre-order is *non-negative*, in the sense that if $A \subseteq B$ is an inclusion of multisets, then $A \preceq_{\mathrm{F}} B$. This property is called non-negativity as it can be equivalently defined as $A \preceq_{\mathrm{F}} A + B$ for any A, B, and it intuitively corresponds to a formula being able to compute all functions in A by first computing all of A and B and then "forgetting" about B.
- Finally, $\preceq_{\mathrm{F}}$ is a *semigroup pre-order* in the sense that if $A \preceq_{\mathrm{F}} B$, then $A+C \preceq_{\mathrm{F}} B+C$ for any multiset of boolean functions C. As we said before, this corresponds to "parallel composition": if we have a "generalized" formula that computes A from leaves labelled with elements of B, and another formula computing C, then we can put these formulas "side-by-side" and compute $A + C$ from $B + C$.

2.2. Branching Programs

We now explore *deterministic branching programs* as pre-orders, which we will now refer to just as *branching programs*. As in the previous section we introduce an ordering on the set $S = \mathbb{N}^{F \cup \{\mathbf{1}\}}$ that will correspond to branching program computation.

A deterministic branching program is defined as follows. We have a directed acylic graph with some dedicated *start nodes* $s_1, \ldots, s_n$ and some dedicated *output nodes* $a_1, \ldots, a_m$. The start nodes have in-degree zero and the output nodes have out-degree zero. All other nodes have unbounded in-degree, and out-degree two, and the two outgoing arcs are labelled by x_i and $\overline{x_i}$ respectively. Given an input x, every output node a_j computes a boolean function by taking the OR-sum over all directed paths from *any* start node s_i to a_j of the AND-product of the arc labels on the path. For example the following branching program of size four computes the boolean functions $f_1 = x_1 \vee \overline{x_2}$ and $f_2 = \overline{x_1} \vee x_2$:

$$\begin{tikzcd}
s_1 \bullet \arrow[r, "x_1"] \arrow[dr, "\overline{x_1}"'] & \bullet a_1 & f_1 \\
s_2 \bullet \arrow[r, "x_2"'] \arrow[ur, "\overline{x_2}"] & \bullet a_2 & f_2
\end{tikzcd}$$

and the following branching program of size eight computes the boolean functions $\{f_1, f_2\}$, where $f_1 = ((\overline{x_1} \vee x_2) \wedge \overline{x_1}) \vee ((x_1 \vee \overline{x_2}) \wedge \overline{x_3})$ and $f_2 = ((\overline{x_1} \vee x_2) \wedge \overline{x_1}) \vee ((\overline{x_1} \vee x_2) \wedge \overline{x_1})$:

$$\begin{tikzcd}
s_1 \bullet \arrow[r, "x_1"] \arrow[dr, "\overline{x_1}"'] & \bullet \arrow[r, "\overline{x_3}"] \arrow[dr, "x_3"'] & \bullet a_1 & f_1 \\
s_2 \bullet \arrow[r, "x_2"'] \arrow[ur, "\overline{x_2}"] & \bullet \arrow[r, "\overline{x_1}"'] \arrow[ur, "x_1"] & \bullet a_2 & f_2
\end{tikzcd}$$

Note that we do not require the start nodes s_i and output nodes a_j to be paired up in any way. In other words, we do not care from which start node a computation path starts at when it reaches an output node. Also, we do not require the output nodes to come in accept–reject pairs a_j, r_j, which is often required in the literature. This means that the collection of boolean functions $\{\{f_1, \ldots, f_n\}\}$ computed by our kind of branching program cannot necessarily be written as a disjoint union of collections $\{\{g, \overline{g}\}\}$ of a boolean function g and its negation $\overline{g}$. In this sense, our model is more general and subsumes the other definitions occurring in the literature. For any multiset of boolean functions $\{\{f_1, \ldots, f_n\}\}$, we define $\mathsf{BP}(\{\{f_1, \ldots, f_n\}\})$ to mean the size (= number of edges) in the smallest deterministic branching program computing the functions $f_1, \ldots, f_n$ among its outputs.

Neither boolean formulas nor deterministic branching programs can directly copy intermediate results in the sense that neither model can simulate a *copy gate* $x \mapsto (x, x)$ [Sub90] (as opposed to non-deterministic branching programs, which *can* simulate a copy gate, making their amortized version trivial). However, there is a crucial difference between boolean formulas and branching programs: in a boolean formula every gate outputs a *single* boolean function, whereas in a branching program each query outputs *two* boolean functions. This means that a branching program can compute a collection of boolean functions in a potentially interesting way using *overlapping subcomputations*, whereas a boolean formula can compute a collection of boolean functions only by computing each boolean function completely separately.

Formally speaking, we can phrase this as follows. Formula size F is *additive* over multisets, in the sense that $\mathsf{F}(A + B) = \mathsf{F}(A) + \mathsf{F}(B)$ for multisets A, B. However, this is not true of branching program complexity! Indeed, this can already be seen when computing the multiset $\{\{x \wedge y, x \wedge \overline{y}, \overline{x}\}\}$. The following branching program (with 4 edges) computes all of these functions simultaneously:

$$\begin{tikzcd}
s\, \bullet \arrow[r, "x"] \arrow[dr, "\overline{x}"'] & \bullet \arrow[r, "y"] \arrow[dr, "\overline{y}"'] & \bullet\, x \wedge y \\
& \bullet\, \overline{x} & \bullet\, x \wedge \overline{y}
\end{tikzcd}$$

Thus $\mathsf{BP}(\{\{x \wedge y, x \wedge \overline{y}, \overline{x}\}\}) \leq 4$. However, it is easy to see that $\mathsf{BP}(x \wedge y) = \mathsf{BP}(x \wedge \overline{y}) \geq 4$, since we must make at least two queries (to x and y) in any branching program for either of these functions. This means that BP is not additive, and so a branching program can sometimes more efficiently compute a family of functions together than it can compute them separately. It is therefore meaningful to introduce the *amortized branching program size* of a boolean function f as $\underset{\sim}{\mathsf{BP}}(f) = \lim_{m \to \infty} \mathsf{BP}(m \cdot f)/m$ where $m \cdot f = \{\{f, \ldots, f\}\}$ is the multiset in which f appears m times. As we will see later, $\underset{\sim}{\mathsf{BP}}(f)$ can be strictly smaller than $\mathsf{BP}(f)$.

The corresponding formal complexity measure for branching programs is called a *branching complexity measure*. This is any function $\mu : F \to \mathbb{R}_{\geq 0}$ such that for any literal ℓ and any $f, g \in F$:

$$\begin{aligned} \mu(f \vee g) &\leq \mu(f) + \mu(g) \\ \mu(f \wedge x_i) + \mu(f \wedge \overline{x_i}) &\leq \mu(f) + 2 \\ \mu(x_i) + \mu(\overline{x}_i) &\leq 2 \end{aligned}$$

By induction over the branching program we can prove that branching measures lower bound the number of edges in any branching program computing f, and it even lower bounds the amortized complexity $\underset{\sim}{\mathsf{BP}}(f)$.

Lemma 2.3. *Let A be a multiset of boolean functions, let B be a branching program computing A, and let μ be any branching complexity measure. If B has s edges then*

$$\sum_{f \in A} \mu(f) \leq s.$$

In particular, if $A = m \cdot f := \{\{f, f, \dots, f\}\}$ (repeated m times), then $m \cdot \mu(f) \leq \mathsf{BP}(m \cdot f)$, and so $\mu(f) \leq \underset{\sim}{\mathsf{BP}}(f)$.

Proof. We first observe that we can define any branching program B inductively by "growing" it from a collection of sink nodes S as follows. Initially, $S = \emptyset$, and at any given point in time in the process we will have a branching program B_i with a collection of sink nodes S. In a single step, we can choose to either:

- Choose two sink $u, v \in S$ and merge them together into a single sink node (that is, directing all incoming edges to u and v to the new merged node). This corresponds to applying an $\vee$ gate to the functions computed at the two sinks.
- Create a new source node by querying a variable: that is, we create a source node labelled with x_i that is connected to two new sink nodes, each corresponding to the outputs x_i, $\overline{x}_i$.
- For any variable x_i, choose a sink node $u \in S$ and "query" x_i, creating two new sink nodes connected to u with edges labelled by x_i and $\overline{x}_i$. This corresponds to replacing the sink node for a function f with two new sink nodes for $f \wedge x_i$ and $f \wedge \overline{x}_i$.

We now prove the claim by induction over B, generated by the above process. In the base case the branching program has queried a single variable x_i, and so it contains two edges and we clearly have $\mu(x_i) + \mu(\overline{x}_i) \leq 2$. Suppose we have a branching program B computing all functions in the multiset A, and suppose we apply the first rule to A, choosing functions $f, g \in A$ and replacing them with $f \vee g$. This does not change the number of edges, but A is replaced with $(A \cup \{f \vee g\}) \setminus \{f, g\}$. By induction we have

$$s \geq \sum_{h \in A} \mu(h) \geq \mu(f \vee g) + \sum_{h \in A \setminus \{f,g\}} \mu(h),$$

where we have applied the first rule of a branching measure. The other two cases are handled similarly.

To see the final claim, we observe that

$$m \cdot \mu(f) = \sum_{f \in m \cdot f} \mu(f) \leq \mathsf{BP}(m \cdot f)$$

and dividing by m and taking $m \to \infty$ completes the proof. □

To summarize the previous argument, we now have observed that

$$\mu(f) \leq \underset{\sim}{\mathsf{BP}}(f) \leq \mathsf{BP}(f).$$

Moreover, the idea behind the previous proof informs the "pre-order" point of view for branching programs.

Definition 2.4. Define the ordering $\preceq_{\mathrm{BP}}$ on S as follows. First, for any boolean functions $f, g \in F$ and any input variable x_i define

$$\begin{aligned} \{\{f \vee g\}\} &\leq \{\{f, g\}\} \\ \{\{f \wedge x_i, f \wedge \overline{x_i}\}\} &\leq \{\{f, \mathbf{1}, \mathbf{1}\}\} \\ \{\{x_i, \overline{x}_i\}\} &\leq \{\{\mathbf{1}, \mathbf{1}\}\}. \end{aligned}$$

Then let $\preceq_{\mathrm{BP}}$ be the smallest ordering containing $\leq$ satisfying the following properties, for $A, B, C \in S$:

- If $A \subseteq B$ then $A \preceq_{\mathrm{BP}} B$.
- If $A \preceq_{\mathrm{BP}} B$ then $A + C \preceq_{\mathrm{BP}} B + C$.
- If $A \preceq_{\mathrm{BP}} B$ and $B \preceq_{\mathrm{BP}} C$ then $A \preceq_{\mathrm{BP}} C$.

The pre-order $\preceq_{\mathrm{BP}}$ "builds" branching programs (as in the proof of Lemma 2.3), and we have that $\{\{f\}\} \preceq_{\mathrm{BP}} \mathbf{n} \coloneqq \{\{\mathbf{1}, \ldots, \mathbf{1}\}\}$ if and ony if there is a branching program that computes f and that uses $n/2$ query gates. More generally, one can argue that $\{\{f_1, \ldots, f_m\}\} \preceq_{\mathrm{BP}} \{\{g_1, \ldots, g_k, \mathbf{1}, \ldots, \mathbf{1}\}\}$ if and only if there is a branching program computing all of the boolean functions $f_1, \ldots, f_m$ *from* the boolean functions $g_1, \ldots, g_k$ using as many query gates as there are units $\mathbf{1}$ in the multiset on the right-hand side.

How does $\preceq_{\mathrm{BP}}$ relate the the branching program measures μ? Linearly extending any branching program measure μ to a function $S \to \mathbb{R}_{\geq 0}$, the branching program measures $\mu : S \to \mathbb{R}_{\geq 0}$ are precisely the functions $S \to \mathbb{R}_{\geq 0}$ that are *monotone* under $\preceq_{\mathrm{BP}}$ and *additive* under taking the union of multisets.

One may think that perhaps $\underset{\sim}{\mathsf{BP}}$ is *itself* a branching program measure — after all, this holds for amortized formula size $\underset{\sim}{\mathsf{F}}$. This, however, is also not true. We have shown above that if $A = \{\{x \wedge y, x \wedge \overline{y}, \overline{x}\}\}$, then

$$\underset{\sim}{\mathsf{BP}}(A) \leq \mathsf{BP}(A) \leq 4$$

by direct construction. On the other hand, it is possible to construct branching program measures μ, μ', μ'' such that $\mu(x \wedge y) = 2$, $\mu'(x \wedge \overline{y}) = 2$, and $\mu''(\overline{x}) = 2$ (cf. Appendix A) (these must necessarily be *different* measures for each function). It therefore follows that

$$\underset{\sim}{\mathsf{BP}}(x \wedge y), \underset{\sim}{\mathsf{BP}}(x \wedge \overline{y}), \underset{\sim}{\mathsf{BP}}(\overline{x}) \geq 2,$$

and so

$$\underset{\sim}{\mathsf{BP}}(\{\{x \wedge y, x \wedge \overline{y}, \overline{x}\}\}) \neq \underset{\sim}{\mathsf{BP}}(x \wedge y) + \underset{\sim}{\mathsf{BP}}(x \wedge \overline{y}) + \underset{\sim}{\mathsf{BP}}(\overline{x}).$$

Thus $\underset{\sim}{\mathsf{BP}}$ is not additive and so it cannot be a branching program measure.

We point out one final interesting property. If $\mu(\{\{f, h\}\}) \leq \mu(\{\{g, h\}\})$, then it also follows (from additivity) that $\mu(f) \leq \mu(g)$. This is because

$$\mu(f) + \mu(h) = \mu(\{\{f, h\}\}) \leq \mu(\{\{g, h\}\}) = \mu(g) + \mu(h)$$

and then cancelling $\mu(h)$ on both sides. This statement is related to the phenomenon of *catalysts* in computation: it is possible that $\{\{f, h\}\} \preceq_{\mathrm{BP}} \{\{g, h\}\}$ but *not* $\{\{f\}\} \preceq_{\mathrm{BP}} \{\{g\}\}$, in which case we say that the boolean function h (or generally some collection of boolean functions) is a *catalyst* that enables the branching program inequality $\{\{f, h\}\} \preceq_{\mathrm{BP}} \{\{g, h\}\}$.

Concretely, we can imagine this as follows. Suppose that we have a branching program B, but, B is augmented with two "special" source nodes s_g, s_h that correspond to the functions g and h. Given any input x, these source nodes will be activated on x if and only if $g(x)$ or $h(x)$ evaluate to 1, respectively. Then, starting from these source nodes, the branching program B has two sink nodes corresponding to the functions f and h. The name "catalyst" is fitting here for h since it is used as an *input* to the program B, but also appears as an *output* of the program.

From this observation we are naturally lead to define the *catalytic branching program complexity* $\mathsf{BP}_{\mathrm{cat}}(f)$ as the smallest number $n \in \mathbb{N}$ such that there exists a collection of boolean functions $A \in S$ for which $\{\{f\}\} + A \preceq_{\mathrm{BP}} \mathbf{n} + A$. From general considerations that we go into later (cf. Section 3) we have

$$\max_{\mu} \mu(f) \leq \underset{\sim}{\mathsf{BP}}(f) \leq \mathsf{BP}_{\mathrm{cat}}(f) \leq \mathsf{BP}(f).$$

The main goal in Section 4 will be to understand which of these inequalities may be strict and when. We will also be able to characterize $\mathsf{BP}_{\mathrm{cat}}(f)$ in our duality theorem as the *integral* optimum for the linear program that we construct; this suggests that it is a natural parameter to consider.

Finally, observe that the branching program pre-order $\preceq_{\mathrm{BP}}$ has all the properties that we saw for the boolean formula pre-order $\preceq_{\mathrm{F}}$: it is bounded (any function can be computed by a branching program), nonnegative (we can forget outputs), finitely generated (there is a finite gate set) and a semigroup pre-order (we may compose branching programs).

2.3. Comparator Circuits

As we have already discussed comparator circuits in the introduction, we will be more brief here. We measure the size of a comparator circuit as the number of literals used at the start of the circuit; it is known that the number of gates is polynomially related to this measure without loss of generality [GR20]. The *comparator circuit size* of a collection of boolean functions $A \in S$ is the smallest size of a comparator circuit computing A and we denote this by $\mathsf{CC}(A)$. For any boolean function f we let $\underset{\sim}{\mathsf{CC}}(f) := \lim_{m\to\infty} \mathsf{CC}(m \cdot f)/m$ be the *amortized comparator circuit size*, where $m \cdot f = \{\{f, \ldots, f\}\}$ is the multiset in which f appears m times.

As with branching programs, we can encode comparator circuit computation as a pre-order.

Definition 2.5. Define the ordering $\preceq_{\mathrm{CC}}$ on $S = \mathbb{N}^{F \cup \{\mathbf{1}\}}$ as follows. First, for any boolean functions f, g and any input literal ℓ, define

$$\{\{f \vee g, f \wedge g\}\} \leq \{\{f, g\}\},$$
$$\{\{\ell\}\} \leq \{\{\mathbf{1}\}\}.$$

Then let $\preceq_{\mathrm{CC}}$ be the smallest ordering that contains $\leq$ and satisfies the following properties, for $A, B, C \in S$:

- If $A \subseteq B$, then $A \preceq_{\mathrm{CC}} B$.
- If $A \preceq_{\mathrm{CC}} B$, then $A + C \preceq_{\mathrm{CC}} B + C$.
- If $A \preceq_{\mathrm{CC}} B$ and $B \preceq_{\mathrm{CC}} C$, then $A \preceq_{\mathrm{CC}} C$.

Everything that we have discussed for branching programs works here, under the appropriate adjustments. As with formulas and branching programs, there is a corresponding complexity measure for comparator circuits called a *submodular complexity measure.*

Definition 2.6. A *submodular complexity measure* is any function $\mu : F \to \mathbb{R}_{\geq 0}$ such that for any literal ℓ and any $f, g \in F$:

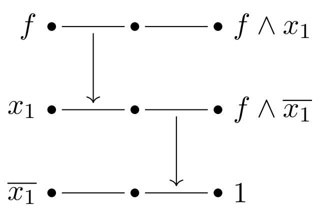
- $\mu(\ell) \leq 1$,
- $\mu(f \vee g) + \mu(f \wedge g) \leq \mu(f) + \mu(g)$.

Submodular complexity measures were introduced by Razborov [Raz92], independently of comparator circuits, in order to study a complexity measure he introduced called the *rank measure* [Raz90]. By induction over the comparator circuit (analogous to Lemma 2.3) we see that for any boolean function f and any submodular measure μ the amortized comparator circuit size of f is at least $\mu(f)$ (see, e.g. [RPRC16]). Moreover, just as with branching programs, we can define the *catalytic comparator circuit complexity* $\mathsf{CC}_{\mathrm{cat}}(f)$ as the smallest number $n \in \mathbb{N}$ such that there exists a collection of boolean functions $A \in S$ for which $\{\{f\}\} + A \preceq_{\mathrm{CC}} \mathbf{n} + A$. In the next section we will see that $\underset{\sim}{\mathsf{CC}}(f) \leq \mathsf{CC}_{\mathrm{cat}}(f) \leq \mathsf{CC}(f)$, and thus for submodular measures μ we have

$$\mu(f) \leq \underset{\sim}{\mathsf{CC}}(f) \leq \mathsf{CC}_{\mathrm{cat}}(f) \leq \mathsf{CC}(f).$$

The comparator circuit pre-order $\preceq_{\mathrm{CC}}$ has all the properties that we saw for the boolean formula pre-order $\preceq_{\mathrm{F}}$ and the branching program pre-order $\preceq_{\mathrm{BP}}$: it is bounded (any function can be computed by a comparator circuit), nonnegative (we can forget computational results), finitely generated (there is a finite gate set) and a semigroup pre-order (we may compose comparator circuits).

Comparator Circuits and Deterministic Branching Programs. Having discussed in some detail the branching program and comparator circuit models of computation, we should clarify their fundamental and simple relationship. Namely, comparator circuits can directly simulate branching programs. To see this, note that the comparator circuit gadget



precisely simulates a branching program query gate that queries the input variable x_1, and at exactly the same cost of 2 that we assigned to any query gate. The branching program OR gate is also trivially simulated in a comparator circuit. Thus comparator circuits are therefore at least as powerful as branching programs and, as a consequence of this very tight simulation, we have $\mathsf{CC} \leq \mathsf{BP}$ and $\underset{\sim}{\mathsf{CC}} \leq \underset{\sim}{\mathsf{BP}}$. In the same spirit, it is easily seen directly that any submodular complexity measure is a branching program complexity measure.

Monotone Comparator Circuits. We briefly indicate how everything in this section can naturally be adapted to capture computation by monotone comparator circuits (motived by our Conjecture 1.3). A boolean function $f : \{0,1\}^n \to \{0,1\}$ is called *monotone* if f is non-decreasing under flipping any input bit from 0 to 1. Let F_n^{mon} denote the set of all *monotone* boolean functions $\{0,1\}^n \to \{0,1\}$ and let $S^{\mathrm{mon}} = \mathbb{N}^{F_n^{\mathrm{mon}} \cup \{\mathbf{1}\}}$ be the corresponding semigroup. A *monotone comparator circuit* is a comparator circuit in which the literals that are used can only be variables x_i, and not their negations $\overline{x_i}$. This naturally gives rise to the *monotone comparator circuit size* mCC and its amortized version $\mathsf{m}\underset{\sim}{\mathsf{C}}\mathsf{C}$. It is clear how the comparator circuit pre-order $\preceq_{\mathrm{CC}}$ (Definition 2.5) can be adapted to give an ordering $\preceq_{\mathrm{CC}}^{\mathrm{mon}}$ on S^{mon} that captures monotone computation (by restricting to F_n^{mon} and relaxing the conditions $\{\{\ell\}\} \leq \{\{\mathbf{1}\}\}$ to $\{\{x_i\}\} \leq \{\{\mathbf{1}\}\}$ for all variables x_i).

Razborov [Raz92] naturally defined a *monotone submodular complexity measure* as any function $\mu : F_n^{\mathrm{mon}} \to \mathbb{R}_{\geq 0}$ such that for any variable x_i and any $f, g \in F_n^{\mathrm{mon}}$:

- $\mu(x_i) \leq 1$
- $\mu(f \vee g) + \mu(f \wedge g) \leq \mu(f) + \mu(g)$.

Again, as in the non-monotone setting, we obtain the general picture that for any such measure μ we have

$$\mu(f) \leq \mathsf{m}\underset{\sim}{\mathsf{C}}\mathsf{C}(f) \leq \mathsf{mCC}_{\mathrm{cat}}(f) \leq \mathsf{mCC}(f).$$

Finally, the monotone comparator circuit pre-order $\preceq_{\mathrm{CC}}^{\mathrm{mon}}$ has all the good (bounded, non-negative, finitely generated, semigroup) properties that $\preceq_{\mathrm{CC}}$ has.

3. Abstract Framework

Having seen three concrete circuit models described as pre-orders, and having seen the recurring concept of formal complexity measures, we will now take a more general approach. The goal of this part is to set up a general framework and define concepts that cover a general notion of a circuit model, and in particular boolean formulas, branching programs and comparator circuits.

3.1. Semigroups of Multisets over a Finite Set

We begin by describing the general kind of objects that we study. Let F be any finite set. Let $\mathbf{1} \in F$ be a special element, which we call the *unit*. This element will not have any special properties for now. (Later on, however, it will play a special role as a measure of "cost" in our definitions of notions of "rank".) Let $S = \mathbb{N}^F$ be the set of non-negative integer vectors indexed by F. For any two vectors $s = (s_f)_{f\in F} \in S$ and $t = (t_f)_{f\in F} \in S$ we define the sum $s + t = (s_f + t_f)_{f\in F}$ as the usual vector sum. With this operation the set S is a semigroup ($+$ is associative and has an identity but no inverses). We may identify the standard basis elements of S with the elements of F, and, under this identification we think of elements of S as formal linear combinations of elements of F with non-negative integer coefficients. Alternatively, we may think of the elements of S as *multisets* of elements of F. Namely, the element $(s_f)_{f\in F} \in S$ corresponds to the multiset in which the element f has multiplicity s_f. For any $n \in \mathbb{N}$ it will be convenient to let $\mathbf{n} := \{\{\mathbf{1}, \dots, \mathbf{1}\}\}$ where $\mathbf{1}$ appears n times.

Briefly connecting to concrete circuit models (boolean formulas, branching programs, comparator circuits), we point out that in those situations we will be considering the semigroup $S = \mathbb{N}^F$ where $F = F_n \cup \{\mathbf{1}\}$, and F_n is the set of all boolean functions $\{0,1\}^n \to \{0,1\}$, and the aforementioned element $\mathbf{1}$ is a formal symbol[8] that we will use to measure complexity, as we will explain.

3.2. Pre-Orders

First of all, we recall that a *pre-order* on a set S is a relation $P \subseteq S^2$ such that for every $a \in S$ it holds that $(a,a) \in P$ and for every $a,b,c \in S$ it holds that $(a,b) \in P$ and $(b,c) \in P$ implies $(a,c) \in P$. For any pre-order P and for every $a,b \in S$ we will write $a \leq_P b$ if and only if $(a,b) \in P$.

On the set $S = \mathbb{N}^F$ there is a natural pre-order, which we call the *pointwise pre-order*. This pre-order is defined by saying that for every $a,b \in S$ we have $a \leq b$ if and only if for every $f \in F$ it holds that $a_f \leq b_f$. Since the pointwise pre-order will play an important role in our proofs, and, since this is arguably the standard pre-order on $\mathbb{N}^F$, we will denote it by $\leq$.

We are interested in pre-orders on S of a special kind. These pre-orders behave well with respect to the semigroup structure of S and have some additional natural properties. In order to discuss them we need to introduce some basic terminology regarding pre-orders.

First we define a notion of *boundedness* for pre-orders which will allow us to define a *rank* function on S later. The notion of boundedness is where the special unit element $\mathbf{1}$ comes in for the first time.

Definition 3.1. We say that a pre-order P on S is *bounded* if for every $s \in S$ there is an $n \in \mathbb{N}$ such that $s \leq_P \mathbf{n}$, where, we recall, $\mathbf{n}$ denotes the multiset $\{\{\mathbf{1}, \dots, \mathbf{1}\}\}$ in which $\mathbf{1}$ appears n times.

Secondly, we will be interested in pre-orders that have the following *composition* property (which the pointwise pre-order satisfies).

Definition 3.2. We call a pre-order P a *semigroup pre-order* if for every $a,b,c,d \in S$ it holds that if $a \leq_P b$ and $c \leq_P d$, then $a + c \leq_P b + d$.

Thirdly, we discuss a natural *non-negativity* property for pre-orders.

Definition 3.3. We say that P is *non-negative* if for every $a,b \in S$ it holds that $a \leq_P a + b$.

Informally, thinking of the elements of S as resources and of P as a collection of feasible transformations between resources, the non-negativity property says that we may always transform $a + b$ to a, that is, we may always throw away part of our resources.

[8]In particular, as mentioned earlier, the element $\mathbf{1} \in F$ (written in bold face) has nothing to do with the constant boolean function $1 \in F_n$.

Note that the pointwise pre-order on S that we defined earlier is non-negative. In fact, one verifies that for every pre-order P on S it holds that P is non-negative if and only if P extends the pointwise pre-order. For semigroup pre-orders, the non-negativity property has the following simple characterization.

Lemma 3.4. *Let P be a semigroup pre-order. Then P extends the pointwise pre-order if and only if for every $s \in S$ it holds that $0 \leq_P s$.*

Proof. Suppose that P extends the pointwise pre-order. Clearly, for every $s \in S$ we have $0 \leq s$ in the pointwise pre-order and so $0 \leq_P s$. On the other hand, suppose that $0 \leq_P s$ for every $s \in S$. Suppose that $a \leq b$. Then there is an element $c \in S$ such that $b = a + c$. We have $0 \leq_P c$ and $a \leq_P a$, and thus $a = a + 0 \leq_P a + c = b$, since P is a semigroup pre-order. □

Finally, we introduce a natural notion of *finiteness* for pre-orders.

Definition 3.5. We say that a pre-order P is *finitely generated* if there exists a finite collection of elements $(a_1, b_1), \ldots, (a_n, b_n) \in P$, called *generators*, such that for every $(a, b) \in P$ there are non-negative integers $y_i \in \mathbb{N}$ such that $a = \sum_{i=1}^n y_i a_i$ and $b = \sum_{i=1}^n y_i b_i$.

In other words, P is finitely generated if any valid inequality $a \leq_P b$ can be obtained as a non-negative integer combination of a finite collection of generating inequalities $a_i \leq_P b_i$. In particular, the pre-orders that we are interested in (the branching program pre-order, formula pre-order and comparactor circuit pre-order) are finitely generated. For example, the comparator circuit pre-order is generated by the comparator gate inequalities $\{\{f \vee g, f \wedge g\}\} \leq_P \{\{f, g\}\}$ and the non-negativity inequalities $\{\{\}\} \leq_P \{\{f\}\}$ for $f, g \in F_n$, and the literal inequalities $\{\{\ell\}\} \leq_P \{\{\mathbf{1}\}\}$ for all literals ℓ.

When a pre-order P has all the above properties, we call it a *good* pre-order:

Definition 3.6 (Good pre-order)**.** Given $S = \mathbb{N}^F$, with a distinguished unit $\mathbf{1} \in F$, we say that a pre-order P on S is *good* if P is a non-negative, finitely generated, bounded, semigroup pre-order.

In the rest of the paper we will (sometimes tacitly) assume that there is a distinguished element $\mathbf{1} \in F$.

3.3. Additive Monotones

Motivated by the formal complexity measures, branching program measures and comparator circuit measures we define the general concept of additive monotones. Given a good pre-order P on the set $S = \mathbb{N}^F$ relative to the distinguished unit element $\mathbf{1} \in F$ (Definition 3.6) we define the following concepts. We say a function $\mu : S \to \mathbb{R}_{\geq 0}$[9] is *$P$-monotone* if for

[9]It will turn out that for our purposes it suffices to restrict the range to non-negative rationals, but for now we stick to the reals.

any $a, b \in S$ if $a \leq_P b$, then $\mu(a) \leq \mu(b)$. We call μ *additive* if for any $a, b \in S$ we have $\mu(a+b) = \mu(a) + \mu(b)$. We say that μ is *normalized* if $\mu(\mathbf{1}) \leq 1$. We will say that μ is an *additive monotone* to mean that it is additive, monotone and normalized.

For $f \in F$ let $e_f \in \mathbb{N}^F$ be the vector that is zero everywhere except in coordinate f. In other words, e_f encodes the multiset $\{\{f\}\}$ and by a slight abusive of notation can be identified with f itself. Clearly, if μ is additive, then μ is completely determined by the values $(\mu(e_f))_{f \in F}$ that μ takes on F.

An important property of additive monotones μ, which follows immediately from the definition, is that

$$\mu(a) \leq \mu(b) \iff \mu(a+c) \leq \mu(b+c)$$

for any $a, b, c \in S$. In other words, additive monotones cannot distinguish between the inequality $a \leq_P b$ and $a + c \leq_P b + c$.

3.4. Catalysts

The observation that additive monotones cannot distinguish $a \leq_P b$ from $a + c \leq_P b + c$ gives rise to the notion of a catalyst, which will play an important role in our duality theorem and its proof. Consider an inequality of the form $a + t \geq_P b + t$ for some elements $a, b, t \in S$. Think of this inequality as a transformation from the "resources" a and t to the "resources" b and t. The resource t enables the transformation, but is in the end not consumed. Thus, we call such an inequality *catalytic* and we call the element t a *catalyst*.

It is natural to wonder whether this abstract notion of a catalyst is related to the model of catalytic space computation of Buhrman et al. [BCK+14], and while we will prove a new result about catalytic space computation later, it will come about differently. That is, we do as of yet not know how to model catalytic space computation using the abstract notion of a catalyst that we discuss here.

When P is a good pre-order on S there is a simple characterization of catalytic inequalities in P in terms of the pointwise pre-order. This will be a core lemma in the proof of our duality theorem.

Lemma 3.7. *Let P be a good pre-order with generators (v_i, w_i). For every $a, b \in S$, the following two statements are equivalent.*

1. *There exists an element $t \in S$ such that $a + t \geq_P b + t$.*
2. *There exist non-negative integers $y_i \in \mathbb{N}$ such that $a + \sum_i y_i(v_i - w_i) \geq b$ in the pointwise pre-order.*

Proof. Suppose that $a + t \geq_P b + t$. Since P is generated by (v_i, w_i), there exist non-negative integers $y_i \in \mathbb{N}$ such that $a + t = \sum_i y_i w_i$ and $b + t = \sum_i y_i v_i$. Then we have $\sum_i y_i v_i - b = t = \sum_i y_i w_i - a$. It follows that $a + \sum_i y_i(v_i - w_i) = b$, and in particular, $a + \sum_i y_i(v_i - w_i) \geq b$.

On the other hand, suppose that $a + \sum_i y_i(v_i - w_i) \geq b$. Then $a + \sum_i y_i v_i \geq b + \sum_i y_i w_i$. From combining the generating inequalities $w_i \geq_P v_i$ we also have $a + \sum_i y_i w_i \geq_P a + \sum_i y_i v_i$. Since P extends the pointwise pre-order, it then follows that $a + \sum_i y_i w_i \geq_P b + \sum_i y_i w_i$. ☐

We finish by isolating a fundamental boosting property of catalytic inequalities which formalizes that a catalyst can be "used over and over again". This will allow us to draw a connection between catalysts and amortized rank in the next section and we will use the boosting property together with Lemma 3.7 later in the proof of the main duality theorem.

Lemma 3.8 (Boosting property). *Let P be a semigroup pre-order. If $a + t \geq_P b + t$, then $ma + t \geq_P mb + t$ for every $m \in \mathbb{N}$.*

Proof. The proof is naturally by induction on m. The base case $m = 1$ is true by assumption. The induction hypothesis is that $(m-1)a + t \geq_P (m-1)b + t$. For the induction step we start with $ma + t = (m-1)a + a + t$. Next, the base case implies that $(m-1)a + a + t \geq_P (m-1)a + b + t$. Finally, the induction hypothesis implies that $(m-1)a + b + t \geq_P (m-1)b + b + t$. We combine all of this to conclude that $ma + t \geq_P mb + t$. ☐

3.5. Ranks

As we have seen in our earlier concrete discussion of circuit models as pre-orders, the notion of complexity (formula size, branching program size, comparatoric circuit size) always takes a similar form, namely of minimizing the amount of units $\mathbf{1}$ required to upper bound the target function in the pre-order. Here we define this concept generally as "rank", and look closely into two variations: the amortized rank and the catalytic rank.

Let P be a non-negative semigroup pre-order on S that is bounded. We will use P and the unitelements $\mathbf{1} \in F$ to define three natural rank functions on S: the rank, the catalytic rank, and the amortized rank. Our main objective is to understand these rank functions.

Definition 3.9. For every $s \in S$ we define the *rank* of s, denoted by $\mathrm{R}(s)$, as the smallest number $n \in \mathbb{N}$ such that $s \leq_P \mathbf{n}$, where, we recall, $\mathbf{n}$ is the multiset $\{\{\mathbf{1}, \ldots, \mathbf{1}\}\}$ that consists of n copies of $\mathbf{1}$. For every $s \in S$ we define the *catalytic rank* of s, denoted by $\mathrm{R}_{\mathrm{cat}}(s)$, as the smallest number $n \in \mathbb{N}$ such that there exists an element $t \in S$, the *catalyst*, such that $s + t \leq_P \mathbf{n} + t$. For every $s \in S$ we define the *amortized rank* of s, denoted by $\underset{\sim}{\mathrm{R}}(s)$, as the infimum $\underset{\sim}{\mathrm{R}}(s) \coloneqq \inf_n \mathrm{R}(ns)/n$.

In other words:

- The rank function $\mathrm{R}(s)$ measures the *cost* of the element s in terms of the unit $\mathbf{1}$
- The catalytic rank is a variation on rank that allows for an extra element to act as a catalyst
- The amortized rank measures the amortized cost of $ns = s + \cdots + s$ (n times) when $n \in \mathbb{N}$ goes to infinity.

We call any function $\phi : S \to \mathbb{R}$ *sub-additive* if for every $s, t \in S$ it holds that $\phi(s + t) \leq \phi(s) + \phi(t)$. Rank, catalytic rank, and amortized rank are sub-additive since P is a semigroup pre-order. Moreover, since rank is sub-additive and bounded, it follows from Fekete's Lemma that $\underset{\sim}{\mathrm{R}}(s) = \lim_{n\to\infty} \mathrm{R}(ns)/n$, which motivates the name *amortized* rank.

The important relation between the three ranks is:

Lemma 3.10. $\underset{\sim}{\mathrm{R}}(s) \leq \mathrm{R}_{\mathrm{cat}}(s) \leq \mathrm{R}(s)$.

Proof. To prove the first inequality, suppose that $\mathrm{R}(s) = n$. Then $s \leq_P \mathbf{n}$. This gives the inequality $s + t \leq_P \mathbf{n} + t$ for the trivial catalyst $t = 0$ (or in fact any choice of $t \in S$), and so $\mathrm{R}_{\mathrm{cat}}(s) \leq n$.

For the second inequality, suppose that $\mathrm{R}_{\mathrm{cat}}(s) = n$. Then $s + t \leq_P \mathbf{n} + t$ for some catalyst $t \in S$. From Lemma 3.8 it follows that $ms + t \leq_P mn\mathbf{1} + t$ for every $m \in \mathbb{N}$. Then it follows that $ms \leq_P ms + t$, since P is non-negative. Crucially, in this inequality we can use the same catalyst t regardless of the number of copies m. Thus we have the rank upper bound $\mathrm{R}(ms) \leq mn + \mathrm{R}(t)$ in which also $\mathrm{R}(t)$ has no dependence on m. This means that $\underset{\sim}{\mathrm{R}}(s) = \lim_{m\to\infty} \mathrm{R}(ms)/m \leq n + \lim_{m\to\infty} \mathrm{R}(t)/m \leq n$. □

Depending on the choice of S and P, the three rank functions R, $\mathrm{R}_{\mathrm{cat}}$ and $\underset{\sim}{\mathrm{R}}$ may or may not coincide with each other.

Example 3.11. In this example, R and $\mathrm{R}_{\mathrm{cat}}$ do not coincide. Let $S = \mathbb{N}^3$ and let $\mathbf{1} = (1, 0, 0)$. Let P be the good pre-order generated by

$$(1, 0, 1) \geq_P (0, 1, 1), \quad (2, 0, 0) \geq_P (0, 1, 0), \quad (2, 0, 0) \geq_P (0, 0, 1).$$

Then $\mathrm{R}((0, 1, 0)) = 2$, while $\mathrm{R}_{\mathrm{cat}}((0, 1, 0)) = 1$ since $(1, 0, 0) + (0, 0, 1) \geq_P (0, 1, 0) + (0, 0, 1)$.

Example 3.12. In this example, $\underset{\sim}{\mathrm{R}}$ and $\mathrm{R}_{\mathrm{cat}}$ do not coincide. Let $S = \mathbb{N}^2$ and let $\mathbf{1} = (1, 0)$. Let P be the good pre-order generated by $(2, 0) \geq_P (0, 3)$. Then we have $\mathrm{R}((0, 1)) = \mathrm{R}_{\mathrm{cat}}((0, 1)) = 2$ while $\underset{\sim}{\mathrm{R}}((0, 1)) = 2/3$.

Recall that $\mu : S \to \mathbb{R}_{\geq 0}$ is called an additive monotone if $\mu(a + b) = \mu(a) + \mu(b)$, $a \leq_P b \implies \mu(a) \leq \mu(b)$ for all $a, b \in S$, and $\mu(\mathbf{1}) \leq 1$. Additive monotones have the following important property:

Lemma 3.13. *If μ is an additive monotone, then for any $s \in S$ it holds that $\mu(s) \leq \underset{\sim}{\mathrm{R}}(s)$.*

Proof. By monotonicity and normalization, it follows that $\mu(s) \leq \mathrm{R}(s)$. Then for any $m \in \mathbb{N}$, it follows from additivity that $\mu(s) = \mu(ms)/m \leq \mathrm{R}(ms)/m$. Taking limits we find $\mu(s) \leq \underset{\sim}{\mathrm{R}}(s)$. □

3.6. General Circuit Pre-Order

We started this section by discussing concrete circuit models as pre-orders (boolean formulas, branching programs, comparator circuits) and we then proceeded by describing a general abstract framework of concepts like a semigroup of multisets, a good pre-order and rank functions. Here we will connect those two parts by discussing a general circuit pre-order that clearly subsumes the concrete circuit models that have seen. The main point is to see that this general circuit pre-order is a *good pre-order* as defined in Definition 3.6.

Definition 3.14 (General circuit pre-order)**.** As before, $\mathbf{1}$ denotes a formal symbol. A general circuit pre-order is any pre-order $\preceq$ on $S = \mathbb{N}^{F \cup \{\mathbf{1}\}}$ obtained as follows. Let

$$\begin{gathered}
\{\{f_{1,1}, \ldots, f_{1,m_1}\}\} \leq \{\{g_{1,1}, \ldots, g_{1,k_1}, \underbrace{\mathbf{1}, \ldots, \mathbf{1}}_{n_1}\}\} \\
\{\{f_{2,1}, \ldots, f_{2,m_2}\}\} \leq \{\{g_{2,1}, \ldots, g_{2,k_2}, \underbrace{\mathbf{1}, \ldots, \mathbf{1}}_{n_2}\}\} \\
\vdots \\
\{\{f_{r,1}, \ldots, f_{r,m_r}\}\} \leq \{\{g_{r,1}, \ldots, g_{r,k_r}, \underbrace{\mathbf{1}, \ldots, \mathbf{1}}_{n_r}\}\}
\end{gathered}$$

be a finite collection of inequalities, for boolean functions $f_{i,j}, g_{i,j} \in F_n$ and numbers $n_i \in \mathbb{N}$. Second, we let $\preceq$ be the smallest ordering that contains $\leq$ and satisfies the following properties, for $A, B, C \in S$:

- If $A \subseteq B$, then $A \preceq B$.
- If $A \preceq B$, then $A + C \preceq B + C$.
- If $A \preceq B$ and $B \preceq C$, then $A \preceq C$.

Finally, we assume that for every boolean function f there is a number $n \in \mathbb{N}$ such that $\{\{f\}\} \preceq \mathbf{n}$ where $\mathbf{n} = \{\{\mathbf{1}, \ldots, \mathbf{1}\}\}$ is the multiset in which $\mathbf{1}$ appears n times.

The general circuit pre-order $\preceq$ encodes, in a very general sense, a circuit model with a finite "gate set" or set of allowed operations on boolean functions. The allowed operations that may be performed in the model each come with their own *cost* $n_i \in \mathbb{N}$.

Lemma 3.15. *The general circuit pre-order $\preceq$ of Definition 3.14 is a good pre-order.*

Proof. We need $\preceq$ to be bounded, finitely generated, non-negative and a semigroup pre-order. Indeed, by definition, the pre-order is bounded, finitely generated by the explicitly given generators corresponding to the gates of the circuit and the non-negativity inequalities $0 \preceq \{\{f\}\}$ for $f \in F_n$, non-negative (because of the explicitly imposed requirement that $A \subseteq B$ implies $A \preceq B$) and a semigroup pre-order (because of the explicitly imposed $A \preceq B$ implies $A + C \preceq B + C$). ◻

One verifies directly that the boolean formula pre-order $\preceq_{\mathrm{F}}$ (Definition 2.2), the branching program pre-order $\preceq_{\mathrm{BP}}$ (Definition 2.4) and the comparator circuit pre-order $\preceq_{\mathrm{CC}}$ (Definition 2.5) are special cases of the general circuit pre-order and are thus *good* pre-orders by Lemma 3.15.

With respect to the general circuit pre-order $\preceq$, the rank $\mathrm{R}(A)$ of $A \in S$ is by definition (Definition 3.9) the smallest number $n \in \mathbb{N}$ such that $A \preceq \mathbf{n} \coloneqq \{\{\mathbf{1}, \ldots, \mathbf{1}\}\}$. The rank equals the complexity of computing a collection of boolean functions with respect to the agreed upon gate set and the prescribed costs n_i per gate.

The amortized rank $\underset{\sim}{\mathrm{R}}(f)$ of $f \in F$ is by definition (Definition 3.9) given by $\underset{\sim}{\mathrm{R}}(f) = \inf_{m \to \infty} \mathrm{R}(m \cdot f)/m$ (and the infimum may equivalently be replaced by a limit since the rank is sub-additive). The amortized rank thus equals the amortized complexity of a boolean function.

In between the rank and the amortized rank, we have the catalytic rank $\mathrm{R}_{\mathrm{cat}}(f)$, which is defined (Definition 3.9) as the smallest number $n \in \mathbb{N}$ such that there exists an element $A \in S$, the catalyst, for which $\{\{f\}\} + A \preceq \mathbf{n} + A$. The catalytic rank equals the complexity of computing f with respect to the gate set at given costs, with the additional freedom of using any collection of functions as a catalyst—we can use these boolean functions at zero cost in our circuit, as long as we make sure to recompute and return them at the end of the computation.

Finally, a formal complexity measure $\mu : S \to \mathbb{R}_{\geq 0}$ with respect to $\preceq$ is any additive monotone $S \to \mathbb{R}_{\geq 0}$. These properties and previously discussed general considerations (Lemma 3.10 and Lemma 3.13) give the "trivial" inequalities:

$$\max_{\mu} \mu(f) \leq \underset{\sim}{\mathrm{R}}(f) \leq \mathrm{R}_{\mathrm{cat}}(f) \leq \mathrm{R}(f).$$

The goal in the next section will be to understand these inequalities more precisely; which ones can be strict and how can we recognize that? We will approach this question by means of a duality theorem.

As a last remark, we emphasize that other concrete circuit models will fit in this framework as well. The notion of asymptotic rank will be more interesting for some than for others. Perhaps the most natural model to consider, besides the aforementioned, is the *boolean circuit* model, which computes boolean functions from literals using OR-gates and AND-gates with fan-out two, say. However, in this model the asymptotic rank becomes trivial. Namely, this model clearly allows us to *copy* results at very low cost. Thus the rank (i.e. circuit size) of $m \cdot f \coloneqq \{\{f, \ldots, f\}\}$ is essentially the same as the rank of $\{\{f\}\}$, and we find that $\underset{\sim}{\mathrm{R}}(f) = \lim_{m \to \infty} \mathrm{R}(m \cdot f)/m = 1$ for every boolean function f in the boolean circuit model.

4. Duality

Strassen [Str88], motivated by the problem of designing fast matrix multiplication algorithms, introduced the theory of asymptotic spectra to study the amortized (i.e., asymptotic) properties of basic objects in mathematics and computer science, and in particular bilinear maps (i.e., tensors). The duality that we introduce here can be thought of as the simplest meaningful instance of this theory. We give a self-contained proof using linear programming duality, and we connect our duality to the theory of formal complexity measures.

The setting is as we have prepared in the previous section: let $S = \mathbb{N}^F$ be a semigroup under coordinatewise addition where F is a finite set and let P be a good pre-order on S (Definition 3.6), relative to $\mathbf{1} \in F$. Relative to P and $\mathbf{1}$ are defined on S the rank R, the catalytic rank $\mathrm{R}_{\mathrm{cat}}$, and the amortized rank $\underset{\sim}{\mathrm{R}}$.

Our duality theorem will be phrased in terms of additive monotones $S \to \mathbb{R}_{\geq 0}$ that we discussed before. In fact, it will turn out to be equivalent (and simpler) to consider only the *rational-valued* additive monotones $S \to \mathbb{Q}_{\geq 0}$ of which we briefly recall the definition. Let $\mu : S \to \mathbb{Q}_{\geq 0}$. We say that μ is *additive* if for every $a, b \in S$ we have $\mu(a + b) = \mu(a) + \mu(b)$. If μ is additive, then μ is determined by its restriction to the standard basis elements of S, that is, we can think of μ as a function $F \to \mathbb{Q}_{\geq 0}$. For a pre-order P on S we say that μ is P-monotone if for every $a, b \in S$ if $a \leq_P b$, then $\mu(a) \leq \mu(b)$. We say that μ is *normalized* if $\mu(\mathbf{1}) \leq 1$. Let M be the set of all functions $\mu : S \to \mathbb{Q}_{\geq 0}$ that are additive, P-monotone and normalized. We call the elements of M simply the *additive monotones*.

Theorem 4.1. *For every $f \in F$ we have that*

$$\max_{\mu \in M} \mu(f) = \underset{\sim}{\mathrm{R}}(f).$$

In particular, for any boolean function f the maximization of $\mu(f)$ over all branching complexity meaures μ equals the amortized branching program size $\underset{\sim}{\mathsf{BP}}(f)$, and the maximization of $\mu(f)$ over all submodular measures μ (comparator cicuit complexity measures) equals the amortized comparator circuit size $\underset{\sim}{\mathsf{CC}}(f)$.

The general picture arising from Theorem 4.1 is that the additive monotones, asymptotic rank, catalytic rank and rank are related by $\max_{\mu \in M} \mu(f) = \underset{\sim}{\mathrm{R}}(f) \leq \mathrm{R}_{\mathrm{cat}}(f) \leq \mathrm{R}(f)$. Generally speaking, $\underset{\sim}{\mathrm{R}}$ may not (and in our applications will not) be an element of M. In other words, for different $f \in F$ the maximization of $\mu(f)$ over $\mu \in M$ may be attained by different μ.

Lets take a moment to repeat the concrete meaning of Theorem 4.1. Recall from our discussion of concrete circuit models that, for any branching program complexity measure μ it holds that μ lower bounds the amortized branching program size $\mu(f) \leq \underset{\sim}{\mathsf{BP}}(f)$. Theorem 4.1 applied to the branching program pre-order $\preceq_{\mathrm{BP}}$ implies that in fact for every boolean function f the maximum of $\mu(f)$ over all branching program measures μ is *equal* to the amortized branching program size, $\max_\mu \mu(f) = \underset{\sim}{\mathsf{BP}}(f)$. However, $\underset{\sim}{\mathsf{BP}}$ is itself not a branching program measure.

Similarly, for submodular measures μ it holds that μ lower bounds the amortized comparator circuit size $\mu(f) \leq \underset{\sim}{\mathsf{CC}}(f)$, and Theorem 4.1 says that $\max_\mu \mu(f) = \underset{\sim}{\mathsf{CC}}(f)$

where the maximization goes over all submodular measures μ. However, $\underset{\sim}{\mathsf{CC}}$ is itself not a submodular measure.

Theorem 4.1 naturally applies just as well to monotone comparator circuits (which is naturally defined by a preorer as discussed at the end of Section 2.3) to give that for every monotone boolean function f the maximum of $\mu(f)$ over all submodular complexity measures on monotone functions μ is equal to the amortized monotone comparator circuit size, $\max_\mu \mu(f) = \mathsf{m}\underset{\sim}{\mathsf{CC}}(f)$.

We return to the general setting. The amortized rank $\underset{\sim}{\mathrm{R}}$ being characterized as the pointwise maximum of the additive monotones μ by Theorem 4.1, what can be said about the *catalytic* rank $\mathrm{R}_{\mathrm{cat}}$? Recall that generally $\underset{\sim}{\mathrm{R}}(f) \leq \mathrm{R}_{\mathrm{cat}}(f)$ with the inequality potentially being strict. As part of our proof of Theorem 4.1, in which we phrase $\max_{\mu\in M}\mu(f)$ as a linear program, we will naturally obtain the following characterization of the catalytic rank.

Theorem 4.2. *For every $f \in F$ we have that $\mathrm{R}_{\mathrm{cat}}(f)$ equals the optimal integral solution of the dual of the linear program $\max_{\mu\in M}\mu(f)$.*

Obviously, the statement of Theorem 4.2 will make more sense to us once we have understood precisely how $\max_{\mu\in M}\mu(f)$ is a linear program, which we will do in a moment.

Finally, we will prove Theorem 4.1 as a special case of the following *amortized inequality* version of the duality theorem (and this is where the pre-order point of view on circuit models is indispensable):

Theorem 4.3. *For every $u, v \in S$, the following are equivalent:*

1. *$\mu(v) \geq \mu(u)$ for every $\mu \in M$*
2. *there is an element $t \in S$ and an integer $k \geq 1$ such that $t + kv \geq_P t + ku$*
3. *there is an integer $c \geq 0$ and an integer $k \geq 1$ such that for all $n \in \mathbb{N}$ we have that $nkv + c\mathbf{1} \geq_P nku$.*

The meaning of Theorem 4.3 is as follows, say for comparator circuits (and the analogous interpretation obviously holds for branching programs as well). If for two multisets of functions $A = \{\{f_1, \ldots, f_m\}\}$ and $B = \{\{g_1, \ldots, g_k\}\}$ and for all submodular measures μ it holds that $\mu(A) \geq \mu(B)$, then there is a collection of (catalyst) functions $\{\{t_1, \ldots, t_r\}\}$ such that from the collection of functions

$$\{\{t_1, \ldots, t_r, \underbrace{f_1, \ldots, f_1}_{k}, \ldots, \underbrace{f_m, \ldots, f_m}_{k}\}\}$$

we can compute the collection of functions

$$\{\{t_1, \ldots, t_r, \underbrace{g_1, \ldots, g_1}_{k}, \ldots, \underbrace{g_m, \ldots, g_m}_{k}\}\}$$

using a comparator circuit.

4.1. Linear Programming

The proof of Theorem 4.1 and Theorem 4.3 is an application of the well-known strong duality theorem for linear programming. For concreteness, the version we use is the following [Sch99, Section 7.4, Equation (19)]:

Theorem 4.4 (Strong duality for linear programming)**.** *Let $A \in \mathbb{Q}^{d_1 \times d_2}$ be a matrix and let $b \in \mathbb{Q}^{d_1}$ and $c \in \mathbb{Q}^{d_2}$ be vectors. Then*

$$\max\{c \cdot x \mid x \in \mathbb{Q}^{d_2}_{\geq 0},\, Ax \leq b\} = \min\{b \cdot y \mid y \in \mathbb{Q}^{d_1}_{\geq 0},\, A^T y \geq c\}$$

provided that both sets are nonempty.

In our application of Theorem 4.4 the sets $\{x \in \mathbb{Q}^{d_2}_{\geq 0} \mid Ax \leq b\}$ and $\{y \in \mathbb{Q}^{d_1}_{\geq 0} \mid A^T y \geq c\}$ will indeed both be nonempty, as a simple consequence of the definition of additive monotones and a good pre-order, respectively.

In other words, in the commonly used notation for linear programs, the following *primal-dual pair* of linear programs give the same optimal value.

$$\boxed{\begin{array}{rl} \max & c \cdot x \\ \text{subject to} & Ax \leq b \\ & x \geq 0 \end{array}} \qquad \boxed{\begin{array}{rl} \min & b \cdot y \\ \text{subject to} & A^T y \geq c \\ & y \geq 0 \end{array}}$$

Before going into the full proof of Theorem 4.1, we describe a more explicit instantiation, in the setting of branching programs, of the ingredients that appear in the proof and in particular of the linear program to which the strong duality will be applied.

On a high level, the primal program $\max c \cdot x$ will correspond to maximizing over branching program complexity measures μ evaluated at a fixed boolean function f, and the dual program $\min b \cdot y$ will correspond to minimizing over amortized branching programs that compute f.

More precisely, the inequalities in the primal are given by the monotonicity and normalization conditions on μ with respect to the branching program pre-order $\preceq_{\mathrm{BP}}$.

$$\boxed{\begin{array}{rrll} \max & \mu(f) & & \\ \text{subject to} & \mu(g \vee h) & \leq \mu(g) + \mu(g) & \forall g, h \in F_n, \\ & \mu(g \wedge x_i) + \mu(g \wedge \overline{x}_i) & \leq \mu(g) + 2 & \forall g \in F_n, i \in [n], \\ & \mu(x_i) + \mu(\overline{x}_i) & \leq 2 & \forall i \in [n], \\ & \mu(\mathbf{1}) & \leq 1 & \\ & \mu & \geq 0 & \end{array}}$$

Thus the primal feasible region will correspond precisely to the branching program complexity measures μ, and the optimal value is precisely $\max_{\mu \in M} \mu(f)$.

The dual program corresponds precisely to branching programs that compute f, amortized. Let R denote the set of all generating inequalities for the branching program pre-order $\preceq_{\mathrm{BP}}$, that is, the inequalities

$$\begin{aligned}\{\{f \vee g\}\} &\preceq_{\mathrm{BP}} \{\{f, g\}\} \\ \{\{f \wedge x_i, f \wedge \overline{x_i}\}\} &\preceq_{\mathrm{BP}} \{\{f, \mathbf{1}, \mathbf{1}\}\} \\ \{\{x_i, \overline{x_i}\}\} &\preceq_{\mathrm{BP}} \{\{\mathbf{1}, \mathbf{1}\}\}\end{aligned}$$

for all boolean functions $f, g \in F_n$, and variables x_i. For any generating inequality $r \in R$ and any boolean function g we will write $r \vdash g$ if r *produces* g and we will write $g \vdash r$ if r *consumes* g. For example, for $r =$ "$\{\{f \vee g\}\} \preceq_{\mathrm{BP}} \{\{f, g\}\}$" we would write the three statements $r \vdash f \vee g$, $f \vdash r$, and $g \vdash r$. To keep track of the varying costs of the generating inequalities, we set $c_r = 0$ for any OR-inequality and $c_r = 2$ for any branching inequality. With this notation, in the following dual program, the vector $y = (y(r))_{r \in R}$ will encode the recipe for a (amortized, as it turns out from careful analysis) branching program that computes f, and $\sum_{\mathbf{1} \vdash r} c_r y(r)$ measures its cost:

$$\boxed{\begin{array}{rr} \text{min} & \sum_{\mathbf{1} \vdash r} c_r y(r) \\ \text{subject to} & \sum_{r \vdash g} y(r) - \sum_{g \vdash r} y(r) \geq 0 \quad \forall g \in F_n, \\ & \sum_{r \vdash f} y(r) - \sum_{f \vdash r} y(r) \geq 1 \\ & y \geq 0 \end{array}}$$

In other words, the dual program optimizes over rational linear combinations of the generating inequalities of $\preceq_{\mathrm{BP}}$, minimizing the number of generating inequalities that consume $\mathbf{1}$ and making sure that, overall, consumption and production at least cancel out, except for when it comes to the boolean function f of which we want to have a net production of at least one. A more careful analysis will reveal how feasible vectors $y \in \mathbb{Q}_{\geq 0}^R$ are in fact not guaranteed to correspond to proper branching programs, but at best correspond to catalytic branching programs and generally to amortized branching programs.

4.2. Full Proof

We will prove in this section the duality theorem for the amortized rank (Theorem 4.1) and the duality theorem for amortized inequalities (Corollary 4.9).

The following theorem is the main technical duality theorem. Theorem 4.1 and Theorem 4.3 will be derived from it in a simple manner afterwards.

As before, let $S = \mathbb{N}^F$ for a finite set F that contains a distinguished element $\mathbf{1} \in F$, let P be a good pre-order on S relative to $\mathbf{1}$ (Definition 3.6), and let M be the set of additive monotones $S \to \mathbb{Q}_{\geq 0}$.

Theorem 4.5. *For every $u, v \in S$ we have that*

$$\max_{\mu \in M} \mu(u) - \mu(v)$$

equals the minimum non-negative rational number r such that there exists an element $t \in S$, an integer $k \geq 1$ such that

$$t + kv + kr\mathbf{1} \geq_P t + ku. \tag{2}$$

To prove Theorem 4.5 we prove two lemmas. We set up some notation that we use in both lemmas. We will apply the strong duality theorem of linear programming (Theorem 4.4) with the following choice of matrix A and vector b. Let $(v_1, w_1), \ldots, (v_d, w_d)$ be (any fixed choice of) generators of P. Let e be the cardinality of F. Let A be the $(d+1) \times e$ matrix with the first d rows given by the vectors $v_1 - w_1, \ldots, v_d - w_d$ corresponding to the generators and the last row given by the vector that encodes the unit $\{\{\mathbf{1}\}\} \in \mathbb{N}^F$. Let b be the $(d+1)$-vector with the first d coefficients equal to 0 and the last coefficient equal to 1.

For $u, v \in S$, let $s = u - v \in \mathbb{Z}^F$.

Lemma 4.6. *We have*

$$\{x \in \mathbb{Q}^F_{\geq 0} \mid Ax \leq b\} = \{(\mu(f))_{f \in F} \mid \mu \in M\}. \tag{3}$$

and therefore

$$\max\{s \cdot x \mid x \in \mathbb{Q}^F_{\geq 0},\ Ax \leq b\} = \max_{\mu \in M} \mu(u) - \mu(v) \tag{4}$$

Proof. Let $x \in \mathbb{Q}^F_{\geq 0}$ satisfy $Ax \leq b$. Define the function $\mu_x : S \to \mathbb{Q}_{\geq 0}$ by setting $\mu_x(f) = x_f$ for every $f \in F$, and then extending additively to all of S. Then μ_x is additive by construction. It follows from the inequality $Ax \leq b$ that μ_x is normalized and P-monotone. Thus $\mu_x \in M$. We see directly that $s \cdot x = \mu_x(s)$. It remains to show that every element $\mu \in M$ is equal to μ_x for some $x \in \mathbb{Q}^F_{\geq 0}$ such that $Ax \leq b$. This is easy to see, since we may simply define $x = (x_f)_{f \in F}$ by $x_f = \mu(f)$ for every $f \in F$. Then $Ax \leq b$ follows from the fact that $\mu \in M$. This proves the claim. □

Lemma 4.7. *Let $r_{\min}$ be the minimum $r \in \mathbb{Q}_{\geq 0}$ such that there exists a $t \in S$ and an integer $k \geq 1$ such that $t + kv + kr\mathbf{1} \geq_P t + ku$. Then $r_{\min} = \min\{b \cdot y \mid y \in \mathbb{Q}^{d+1}_{\geq 0},\ A^T y \geq s\}$.*

Proof. We first prove that

$$r_{\min} \leq \min\{b \cdot y \mid y \in \mathbb{Q}^{d+1}_{\geq 0},\ A^T y \geq s\}.$$

Let $y \in \mathbb{Q}^{d+1}_{\geq 0}$ satisfy $A^T y \geq s$. The inequality $A^T y \geq s$ implies, by construction of A, that we have the pointwise inequality

$$y_1(v_1 - w_1) + \cdots + y_d(v_d - w_d) + y_{d+1}\mathbf{1} \geq s = u - v.$$

There is a positive integer $k \in \mathbb{N}$ such that all elements ky_i are integral. We multiply both sides of the inequality by k to obtain

$$ky_1(v_1 - w_1) + \cdots + ky_d(v_d - w_d) + ky_{d+1}\mathbf{1} + kv \geq ku.$$

From this it follows using Lemma 3.7 that there exists an element $t \in S$ such that

$$k y_{d+1} \mathbf{1} + kv + t \geq_P ku + t.$$

By construction of b we have $b \cdot y = y_{d+1}$ and so $b \cdot y \geq r_{\min}$. This proves one direction of the claim.

We will now prove the other direction of the claim, namely

$$r_{\min} \geq \min\{b \cdot y \mid y \in \mathbb{Q}_{\geq 0}^{d+1},\ A^T y \geq s\}.$$

Suppose that $kv + kr\mathbf{1} + t \geq_P ku + t$. It follows from Lemma 3.7 that there is an element $y \in \mathbb{Q}_{\geq 0}^{d+1}$ such that, in the pointwise pre-order, we have

$$y_1(v_1 - w_1) + \cdots + y_d(v_d - w_d) + y_{d+1}\mathbf{1} + v \geq u$$

where $b \cdot y = y_{d+1} = r$. For this y it holds that $A^T y \geq s$. □

The proof of Lemma 4.7 is easily adapted to prove the following lemma (in which the rationals are replaced by integers), which we will use later to characterize the catalytic rank.

Lemma 4.8. *Let $r_{\min}^{\text{cat}}$ be the minimum $r \in \mathbb{Z}_{\geq 0}$ such that there exists a $t \in S$ such that $t + v + r\mathbf{1} \geq_P t + u$. Then $r_{\min}^{\text{cat}} = \min\{b \cdot y \mid y \in \mathbb{Z}_{\geq 0}^{d+1},\ A^T y \geq s\}$.*

Proof of Theorem 4.5. By Lemma 4.6 we have

$$\max_{\mu \in M} \mu(u) - \mu(v) = \max\{s \cdot x \mid x \in \mathbb{Q}_{\geq 0}^{F},\ Ax \leq b\}$$

and by Lemma 4.7 we have

$$r_{\min} = \min\{b \cdot y \mid y \in \mathbb{Q}_{\geq 0}^{d+1},\ A^T y \geq s\}.$$

The set $\{x \in \mathbb{Q}_{\geq 0}^{F},\ Ax \leq b\}$ always contains the zero vector, and in particular is nonempty. Also the set $\{y \in \mathbb{Q}_{\geq 0}^{d+1},\ A^T y \geq s\}$ always contains at least one vector by the boundedness property of the good pre-order P, and in particular is nonempty. Therefore, by linear programming duality (Theorem 4.4), it follows that $\max_{\mu \in M} \mu(u) - \mu(v)$ equals $r_{\min}$, which proves the theorem. □

We obtain the following corollary.

Corollary 4.9. *For every $u, v \in S$ and $r \in \mathbb{Q}_{\geq 0}$, the following are equivalent:*

1. *$\mu(v) + r \geq \mu(u)$ for every $\mu \in M$*
2. *there is an element $t \in S$ and an integer $k \geq 1$ such that $t + kv + kr\mathbf{1} \geq_P t + ku$*

3. *there is an integer $c \geq 0$ and an integer $k \geq 1$ such that for all $n \in \mathbb{N}$ we have that $nkv + nkr\mathbf{1} + c\mathbf{1} \geq_P nku$.*

Proof. If $\mu(v) + r \geq \mu(u)$ for every $\mu \in M$, then $\max_{\mu \in M} \mu(u) - \mu(v) \leq r$. Then from Theorem 4.5 it follows that $t + kv + kr\mathbf{1} \geq_P t + ku$ for some $t \in S$ and some integer $k \geq 1$.

If there is an element $t \in S$ such that $t + kv + kr\mathbf{1} \geq_P t + ku$, then from Lemma 3.8 and the fact that t is bounded it follows that there is a constant $c \in \mathbb{N}$ such that for all $n \in \mathbb{N}$ we have $nkv + nkr\mathbf{1} + c\mathbf{1} \geq_P nku$.

If $nkv + nkr\mathbf{1} + c\mathbf{1} \geq_P nku$ for all $n \in \mathbb{N}$, then for every $\mu \in M$, using that μ is P-monotone and additive, we have that

$$nk\mu(v) + \mu(nkr) + \mu(c) \geq nk\mu(u).$$

Then, using the upper bound $nkr \geq \mu(nkr)$, and after dividing by nk on both sides, we get that

$$\mu(v) + r + \mu(c)/(nk) \geq \mu(u).$$

We let n go to infinity to get $\mu(v) + r \geq \mu(u)$. □

It is worth considering explicitely the special case of Corollary 4.9 where we set $r = 0$, which gives the following theorem (restated):

Theorem 4.3. *For every $u, v \in S$, the following are equivalent:*

1. *$\mu(v) \geq \mu(u)$ for every $\mu \in M$*
2. *there is an element $t \in S$ and an integer $k \geq 1$ such that $t + kv \geq_P t + ku$*
3. *there is an integer $c \geq 0$ and an integer $k \geq 1$ such that for all $n \in \mathbb{N}$ we have that $nkv + c\mathbf{1} \geq_P nku$.*

Proof. The claim follows directly from Corollary 4.9 by setting $r = 0$. □

We also obtain the duality for the amortized rank as a corollary of Corollary 4.9 (restated):

Theorem 4.1. *For every $f \in F$ we have that*

$$\max_{\mu \in M} \mu(f) = \underset{\sim}{\mathrm{R}}(f).$$

In particular, for any boolean function f the maximization of $\mu(f)$ over all branching complexity meaures μ equals the amortized branching program size $\underset{\sim}{\mathsf{BP}}(f)$, and the maximization of $\mu(f)$ over all submodular measures μ (comparator cicuit complexity measures) equals the amortized comparator circuit size $\underset{\sim}{\mathsf{CC}}(f)$.

Proof. The inequality $\max_{\mu \in M} \mu(u) \leq \underset{\sim}{\mathrm{R}}(u)$ is clear. We now prove the other inequality $\max_{\mu \in M} \mu(u) \geq \underset{\sim}{\mathrm{R}}(u)$. Let $r = \max_{\mu \in M} \mu(u)$. Then, in particular, $r \geq \mu(u)$ for all $\mu \in M$. We apply Corollary 4.9 with $v = 0$, to obtain that for all $n \in \mathbb{N}$ we have that $(nr + o(n))\mathbf{1} \geq_P nu$. This implies that $r \geq \underset{\sim}{\mathrm{R}}(u)$. □

Theorem 4.2. *For every $f \in F$ we have that $\mathrm{R}_{\mathrm{cat}}(f)$ equals the optimal integral solution of the dual of the linear program $\max_{\mu \in M} \mu(f)$.*

Proof. This follows directly from Lemma 4.8 applied to $u = f$ and $v = 0$. □

5. Upper Bounds on Amortized and Catalytic Complexity by Symmetry

In the previous sections we have studied in depth the dual role that branching program complexity measures and submodular measures have in relation to amortized complexity. In this section we prove upper bounds on such measures as well as explicit efficient constructions of amortized branching programs.

Symmetry turns out to be the powerful ingredient for our efficient constructions. First, we consider branching program measures that have a natural symmetry condition. We prove a strong upper bound on such measures in terms of the average decision tree depth. Then, we extend this result by proving a strong upper bound for ordinary branching progam measures on *orbits* of any boolean function under the natural symmetry.

Second, we use the symmetry ideas thus developed to improve the known bounds for catalytic space computation in the sense of [BCK+14]. The important hurdle to overcome in this construction is that in order to get catalytic space algorithms we need the start and output nodes in our branching programs to be paired up in a stronger fashion than we have been enforcing so far. This we are able to do by an extensive modification of the catalytic branching program presented by Potechin [Pot17], keeping careful track of symmetries.

5.1. Symmetry and Formal Complexity Measures

Definition 5.1. A branching program complexity measure is *symmetric* if $\mu(f) \leq \mu(f^{\oplus i})$ for every boolean function f on n variables and every $i \in [n]$, where $f^{\oplus i}$ is the function obtained from f by negating the ith input variable.

As we have mentioned in the introduction, Razborov [Raz92] proved that any *submodular* complexity measure μ satisfies $\mu(f) = O(n)$ for any n-variate boolean function f. Razborov used a randomized construction, and his argument uses the following key symmetry property: if f_0, f_1 are both uniformly random boolean function on $n-1$ variables and f is a uniformly random function on n variables, then

$$(\overline{x}_n \wedge f_0) \vee (x_n \wedge f_1) \sim (x_n \wedge f_0) \vee (\overline{x}_n \wedge f_1) \sim f$$

where $X \sim Y$ if the two variables have the same distribution. We first show that by *explicitly* introducing these symmetry properties, we can strongly improve the upper bounds, and prove upper bounds even for branching program complexity measures rather than submodular measures.

Lemma 5.2. *For any symmetric branching program complexity measure μ and any boolean function f,*

$$\mu(f) \leq 2(D_{avg}(f) + 2)$$

where $D_{avg}(f)$ is the minimum expected number of queries made by any decision tree computing f over the uniform distribution.

Proof. First suppose that f is a constant function. From the axioms of a branching program complexity measure one can show $\mu(f) \leq 4$: this is because if $f = 1$ then

$$\mu(1) \leq \mu(x \vee \overline{x}) \leq \mu(x) + \mu(\overline{x}) \leq 2$$

and if $f = 0$ then

$$\mu(0) \leq \mu(x \wedge \overline{x}) + \mu(x) + \mu(\overline{x}) \leq \mu(x) + \mu(\overline{x}) + 2 \leq 4.$$

Now, assume that f depends on at least one variable, and let T be the decision tree witnessing $D_{avg}(f)$. Without loss of generality, assume that the first variable queried by T is x_n, and note that T gives a representation of f as

$$f = (f_0 \wedge \overline{x}_n) \vee (f_1 \wedge x_n)$$

where f_0, f_1 are functions that do not depend on x_n. Finally, note that $D_{avg}(f) = D_{avg}(f^{\oplus n})$, since we can obtain a decision tree for $f^{\oplus n}$ from the decision tree for f by negating the first variable.

By definition of T we have

$$D_{avg}(f) = \frac{D_{avg}(f_0)}{2} + \frac{D_{avg}(f_1)}{2} + 1. \tag{5}$$

We claim that $\mu(f) \leq 2D_{avg}(f)$ by induction on n. If $n = 1$ then since f depends on its one input variable, it is either x_n or its negation, and thus $\mu(f) \leq 1 = D_{avg}(f)$. Since $\mu(f) = \mu(f^{\oplus n})$, by the branching program complexity measure axioms we have

$$\begin{aligned}
2\mu(f) &= \mu(f) + \mu(f^{\oplus n}) \\
&= \mu((f_0 \wedge \neg x_n) \vee (f_1 \wedge x_n)) + \mu((f_0 \wedge x_n) \vee (f_1 \wedge \neg x_n)) \\
&\leq \mu(f_0 \wedge \neg x_n) + \mu(f_1 \wedge x_n) + \mu(f_0 \wedge x_n) + \mu(f_1 \wedge \neg x_n) \\
&\leq \mu(f_0) + \mu(f_1) + 4.
\end{aligned}$$

Now, applying the inductive hypothesis and Equation 5,

$$\begin{aligned}\mu(f_0) + \mu(f_1) + 4 &\leq 2(D_{avg}(f_0) + 2) + 2(D_{avg}(f_1) + 2) + 4 \\ &= 4(D_{avg}(f) + 2).\end{aligned}$$

Dividing by 2 yields the lemma. □

We cannot extend this result to general branching program complexity measures. However, by symmetrizing, we can still use the above argument to compute the *orbit* of f efficiently on average.

Definition 5.3. For any boolean function $f : \{0,1\}^n \to \{0,1\}$ and any $T \subseteq [n]$ let

$$\mathsf{Orb}_T(f) := \{f^{\oplus S} : S \subseteq T\}$$

be the orbit of f under the action of negating any subset of bits in T. Let Π_T denote the symmetry group of F_n corresponding to this action, so that $\mathsf{Orb}_T(f) = \Pi_T \cdot f$. (Note that Π_T is isomorphic to a direct product of n groups, either S_2 in coordinates of T or the trivial group in coordinates outside of T). If $T = [n]$ then we will just write $\mathsf{Orb}(f)$.

Let $f : \{0,1\}^n \to \{0,1\}$ be any boolean function, let $i \in [n]$, and let $b \in \{0,1\}$. Let $f_{i\leftarrow b} : \{0,1\}^{n-1} \to \{0,1\}$ denote the function obtained from f by substituting in b for the i^{th} input of f. For any set of functions A let $M(A, i, b) := \{\{g_{i\leftarrow b} : g \in A\}\}$ denote the multiset obtained by substituting b for the ith bit for each function in A, and note that $|M(A, i, b)| = |A|$ since $M(A, i, b)$ is a multiset. In order to prove our upper bound using symmetrization, we will need the following group-theoretic lemma. Roughly speaking, it shows that if we take every function in $\mathsf{Orb}(f)$ and set the ith bit to $b \in \{0,1\}$, then the result will be many copies of $\mathsf{Orb}(f_{i\leftarrow 1})$ and $\mathsf{Orb}(f_{i\leftarrow 0})$. This will be quite useful since we will ultimately need to understand the action of querying a single bit in a decision tree on $\mathsf{Orb}(f)$.

Lemma 5.4. *Let $f : \{0,1\}^n \to \{0,1\}$ be a boolean function, let $i \in [n]$, and let $b \in \{0,1\}$. Let $u_b = |\mathsf{Orb}(f)|/2|\mathsf{Orb}(f_{i\leftarrow b})|$. Then either*

- $\mathsf{Orb}(f_{i\leftarrow 1}) = \mathsf{Orb}(f_{i\leftarrow 0})$ *and $M(\mathsf{Orb}(f), i, b)$ can be partitioned into $u_0 + u_1$ copies of* $\mathsf{Orb}(f_{i\leftarrow 1})$.
- $\mathsf{Orb}(f_{i\leftarrow 1}) \neq \mathsf{Orb}(f_{i\leftarrow 0})$, *and $M(\mathsf{Orb}(f), i, b)$ can be partitioned into u_0 copies of* $\mathsf{Orb}(f_{i\leftarrow 0})$ *and u_1 copies of* $\mathsf{Orb}(f_{i\leftarrow 1})$.

In either case, we can write $|M(\mathsf{Orb}(f), i, b)| = u_0|\mathsf{Orb}(f_{i\leftarrow 0})| + u_1|\mathsf{Orb}(f_{i\leftarrow 1})|$.

Proof. First, observe that if f is a constant function, then $\mathsf{Orb}(f) = \mathsf{Orb}(f_{i\leftarrow b})$ for $b \in \{0,1\}$ and so $u_0 = u_1 = 1/2$ and we are in the first bullet. From now on, we assume that f depends on at least one of its input variables.

Let $U = [n] \setminus \{i\}$. We can write $\mathsf{Orb}(f) = \mathsf{Orb}_U(f) \cup \mathsf{Orb}_U(f^{\oplus i})$ with $|\mathsf{Orb}_U(f)| = |\mathsf{Orb}_U(f^{\oplus i})|$ since Π_U is a subgroup of $\Pi_{[n]}$ and acting on f by a subgroup of $\Pi_{[n]}$ partitions $\mathsf{Orb}(f)$. If $g, h : \{0,1\}^n \to \{0,1\}$ then let $g \simeq_b h$ if $g_{i\leftarrow b} = h_{i\leftarrow b}$.

We begin by claiming that for any function g, $M(\mathsf{Orb}_U(g), i, b)$ can be partitioned into $u = |\mathsf{Orb}_U(g)|/|\mathsf{Orb}(g_{i\leftarrow b})|$ copies of the orbit $\mathsf{Orb}(g_{i\leftarrow b})$. To see this, observe that $\simeq_b$ is an equivalence relation, and so $\mathsf{Orb}_U(g)$ is partitioned by $\simeq_b$ into sets $\{S_1, S_2, \ldots, S_m\}$. Furthermore, $\simeq_b$ is invariant under the action of Π_U, and so we have that $|S_k| = |S_\ell|$ for each k, ℓ and Π_U acts on the collection of sets in the natural way. This means that $m = |\mathsf{Orb}(g_{i\leftarrow b})|$ and proves the claim.

First assume that $\mathsf{Orb}_U(f) \neq \mathsf{Orb}(f)$. Then

$$M(\mathsf{Orb}(f), i, b) = M(\mathsf{Orb}_U(f), i, b) \sqcup M(\mathsf{Orb}_U(f^{\oplus i}), i, b)$$

where $\sqcup$ denotes disjoint union. By applying the above partitioning claim to both $M(\mathsf{Orb}_U(f), i, b)$ and $M(\mathsf{Orb}_U(f^{\oplus i}), i, b)$ we get integers u_0, u_1 such that

$$u_0 = \frac{|\mathsf{Orb}_U(f^{\oplus i})|}{|\mathsf{Orb}(f^{\oplus i}_{i\leftarrow 1})|} = \frac{|\mathsf{Orb}(f)|}{2|\mathsf{Orb}(f_{i\leftarrow 0})|}, \quad u_1 = \frac{|\mathsf{Orb}_U(f)|}{|\mathsf{Orb}(f_{i\leftarrow 1})|} = \frac{|\mathsf{Orb}(f)|}{2|\mathsf{Orb}(f_{i\leftarrow 1})|},$$

and $M(\mathsf{Orb}(f), i, b)$ is u_0 copies of $\mathsf{Orb}(f_{i\leftarrow 0})$ and u_1 copies of $\mathsf{Orb}(f_{i\leftarrow 1})$, proving the lemma in this case.

Now, assume that $\mathsf{Orb}_U(f) = \mathsf{Orb}(f)$, which also implies $\mathsf{Orb}_U(f^{\oplus i}) = \mathsf{Orb}(f)$. This means that $\mathsf{Orb}(f_{i\leftarrow 0}) = \mathsf{Orb}(f_{i\leftarrow 1})$, as for any $g \in \mathsf{Orb}(f_{i\leftarrow 0})$ we have, for some S and T,

$$g = f^{\oplus S}_{i\leftarrow 0} = f^{\oplus S\cup\{i\}}_{i\leftarrow 1} = f^{\oplus T}_{i\leftarrow 1}$$

where the last step follows since $\mathsf{Orb}_U(f) = \mathsf{Orb}(f)$. Now, since $\mathsf{Orb}_U(f) = \mathsf{Orb}(f) = \mathsf{Orb}_U(f^{\oplus i})$, we have

$$M(\mathsf{Orb}(f), i, b) = M(\mathsf{Orb}_U(f), i, b) = M(\mathsf{Orb}_U(f^{\oplus i}), i, b).$$

Since $\mathsf{Orb}(f_{i\leftarrow 0}) = \mathsf{Orb}(f_{i\leftarrow 1})$, by applying the above partitioning argument we see that $M(\mathsf{Orb}(f), i, b)$ can be partitioned into some positive number u of copies of $\mathsf{Orb}(f_{i\leftarrow 1})$. In this case, to see the final claim of the lemma, note that

$$u = |\mathsf{Orb}(f)|/|\mathsf{Orb}(f_{i\leftarrow 1})| = 2u_0 = 2u_1$$

and also $\mathsf{Orb}(f_{i\leftarrow 1}) = \mathsf{Orb}(f_{i\leftarrow 0})$, so

$$|M(\mathsf{Orb}(f), i, b)| = u|\mathsf{Orb}(f_{i\leftarrow 1})| = u_0|\mathsf{Orb}(f_{i\leftarrow 0})| + u_1|\mathsf{Orb}(f_{i\leftarrow 1})|. \qquad \square$$

The next theorem bounds the complexity of the orbit of f.

Theorem 5.5. *For any boolean function f on n bits and any branching program complexity measure μ*

$$\mu(\mathsf{Orb}(f)) \le 2|\mathsf{Orb}(f)|(D_{avg}(f) + 2).$$

Proof. The proof is by induction on n, the number of variables on which f depends. Again, if $n = 0$ then $\mathsf{Orb}(f) = \{f\}$, and so one can show (as was shown in Lemma 5.2) that $\mu(f) \le 4$. If $n = 1$ then $f = x_1$, $D_{avg}(f) = 1$, and $\mathsf{Orb}(x_1) = \{x_1, \overline{x}_1\}$, so

$$\mu(x_1) + \mu(\overline{x}_1) = 2 \le 4 = 2|\mathsf{Orb}(f)|D_{avg}(f).$$

Now, for the induction step. Let T be a decision tree for f witnessing $D_{avg}(f)$, and suppose w.l.o.g. that x_n is the first variable queried by the decision tree. We can write $f = (f_0 \wedge \overline{x}_n) \vee (f_1 \wedge x_n)$ where $f_b = f_{n \leftarrow b}$, and note that since T witnesses $D_{avg}(f)$ we have

$$D_{avg}(f) = \frac{D_{avg}(f_0)}{2} + \frac{D_{avg}(f_1)}{2} + 1.$$

Since f depends on x_n, the action of negating the nth input partitions $\mathsf{Orb}(f)$ into pairs $(g, g^{\oplus n})$, so, let B be the set of all such pairs. Then, applying the axioms of a branching program measure, we again have

$$\begin{aligned}
\mu(\mathsf{Orb}(f)) &= \sum_{g \in \mathsf{Orb}(f)} \mu(g) \\
&= \sum_{(g, g^{\oplus n}) \in B} \mu(g) + \mu(g^{\oplus n}) \\
&= \sum_{(g, g^{\oplus n}) \in B} \mu((g_0 \wedge \overline{x}_n) \vee (g_1 \wedge x_n)) + \mu((g_0 \wedge x_n) \vee (g_1 \wedge \overline{x}_n)) \\
&\le \sum_{(g, g^{\oplus n}) \in B} \mu(g_0 \wedge \overline{x}_n) + \mu(g_1 \wedge x_n) + \mu(g_0 \wedge x_n) + \mu(g_1 \wedge \overline{x}_n) \\
&\le \sum_{(g, g^{\oplus n}) \in B} \mu(g_0) + \mu(g_1) + 4 \\
&= \sum_{g \in M(\mathsf{Orb}(f), n, 1)} \mu(g) + 2
\end{aligned}$$

where the last equality follows since we have substituted in 1 for x_n in every g in $\mathsf{Orb}(f)$ (note that $g_1^{\oplus n} = g_0$). By applying Lemma 5.4 we can partition $M(\mathsf{Orb}(f), n, 1)$ into copies of $\mathsf{Orb}(f_{n \leftarrow 1}), \mathsf{Orb}(f_{n \leftarrow 0})$. By using the final claim in the lemma, we can write this as

$$\begin{aligned}
\sum_{g \in M(\mathsf{Orb}(f), n, 1)} \mu(g) + 2 &= 2|\mathsf{Orb}(f)| + u_0 \mu(\mathsf{Orb}(f_0)) + u_1 \mu(\mathsf{Orb}(f_1)) \\
&= 2|\mathsf{Orb}(f)| + \frac{|\mathsf{Orb}(f)|}{2|\mathsf{Orb}(f_0)|} \mu(\mathsf{Orb}(f_0)) + \frac{|\mathsf{Orb}(f)|}{2|\mathsf{Orb}(f_1)|} \mu(\mathsf{Orb}(f_1)).
\end{aligned}$$

By the induction hypothesis applied to f_0, f_1, and using the definition of D_{avg}, this previous expression is at most

$$\begin{aligned}
&2|\mathsf{Orb}(f)| + \frac{|\mathsf{Orb}(f)|}{2|\mathsf{Orb}(f_0)|}2|\mathsf{Orb}(f_0)|(D_{avg}(f_0)+2) + \frac{|\mathsf{Orb}(f)|}{2|\mathsf{Orb}(f_1)|}2|\mathsf{Orb}(f_1)|(D_{avg}(f_1)+2)\\
&= 2|\mathsf{Orb}(f)| + |\mathsf{Orb}(f)|(D_{avg}(f_0)+2) + |\mathsf{Orb}(f)|(D_{avg}(f_1)+2)\\
&= 2|\mathsf{Orb}(f)|(D_{avg}(f)+2).
\end{aligned}$$

The proof is complete. □

We can use the above argument to improve known amortized circuit complexity upper bounds for both branching programs and comparator circuits. We will defer the argument for branching programs to the next section, as it is slightly more complicated (and, as we will see, it also provides improved bounds on *nonuniform catalytic space*). For comparator circuits the argument is significantly simpler, and is illustrated next. The main "trick" in our argument is the following well-known fact about XOR.

Lemma 5.6. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be any boolean function. Then for every function $g \in \{0,1\}^n$ there is a unique function $h : \{0,1\}^n \to \{0,1\}$ such that $g = h \oplus f$.*

Proof. Let $h = g \oplus f$. □

Theorem 5.7. *Let $f : \{0,1\}^n \to \{0,1\}$, and let $H \subseteq F_n$ be any set of boolean functions such that*

- *If $g \in H$ then $g \oplus f \in H$.*
- *$\Pi_{[n]} \cdot H = H$, that is, H is closed under negating any subset of inputs.*
- *H is closed under negation: if $f \in H$ then $\overline{f} \in H$.*

Since H is closed under negating any subset of inputs, let $\mathsf{Orb}(g_1), \mathsf{Orb}(g_2), \ldots, \mathsf{Orb}(g_m)$ be a partition of H into orbits. Then for any submodular complexity measure μ,

$$\mu(f) \leq \frac{8}{|H|}\sum_{i=1}^{m} |\mathsf{Orb}(g_i)|(D_{avg}(g_i)+2).$$

Proof. Since H is closed under negating any subset of inputs, there are functions $g_1, g_2, \ldots, g_m$ such that we can partition H into orbits $\mathsf{Orb}(g_1), \mathsf{Orb}(g_2), \ldots, \mathsf{Orb}(g_m)$. Moreover, as comparator circuit complexity measures are also branching program complexity measures, we can apply Theorem 5.5 to each orbit and get

$$\mu(H) \leq \sum_{i=1}^{m} 2|\mathsf{Orb}(g_i)|(D_{avg}(g_i)+2).$$

On the other hand, since H is closed under taking $\oplus f$, we can partition the functions in H up into pairs (g,h) such that $g = h \oplus f$. Furthermore, since H is closed under negation, we have the "complementary" pair $(\overline{g}, \overline{h})$ in H — to see this, note that if $g = h \oplus f$, then

$$\overline{g} = h \oplus f \oplus 1 = (h \oplus 1) \oplus f = \overline{h} \oplus f.$$

With this in mind, let B be the set of all such pairs, and let B^+ be the set of pairs obtained by keeping exactly one of (g,h) or $(\overline{g}, \overline{h})$ for each pair g, h. Observe that a submodular complexity measure is also a formula complexity measure — that is, it satisfies the inequalities $\mu(f \circ g) \leq \mu(f) + \mu(g)$ for $\circ \in \{\wedge, \vee\}$. Using this fact, for any x, y we have

$$\begin{aligned} \mu(x) + \mu(y) + \mu(\overline{x}) + \mu(\overline{y}) &\geq \mu(x \wedge \overline{y}) + \mu(\overline{x} \wedge y) \\ &\geq \mu((\overline{x} \wedge y) \vee (x \wedge \overline{y})) \\ &= \mu(x \oplus y). \end{aligned}$$

Therefore, since H is closed under negation, we have

$$\mu(H) = \sum_{(g,h) \in B^+} \mu(g) + \mu(h) + \mu(\overline{g}) + \mu(\overline{h}) \geq \sum_{(g,h) \in B^+} \mu(g \oplus h) = \frac{|H|\mu(f)}{4}.$$

Combining together the two inequalities on $\mu(H)$ yields the theorem. $\square$

The next lemma gives a nice family of sets H that the previous theorem can be applied to. We will use it crucially in the next section on improved bounds for catalytic space.

Lemma 5.8. *Let $f \in F_n$ be any boolean function such that f is not the constant 0. The set $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ satisfies the following three properties.*

- *If $g \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ then $g \oplus f \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$.*
- *$\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ is closed under negating any subset of inputs.*
- *If $g \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ then $\overline{g} \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$.*

Proof. It is clear that $f \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$, and also if $g \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ then $g \oplus f \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$. Now, suppose that $g \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ and let $S \subseteq [n]$. We can write $g = \sum_{i=1}^m f_i$ where $f_i \in \mathsf{Orb}(f)$ for each i. Note that for any functions h_1, h_2, $(h_1 \oplus h_2)^{\oplus S} = h_1^{\oplus S} \oplus h_2^{\oplus S}$. Therefore

$$g^{\oplus S} = \left(\sum_{i=1}^m f_i \right)^{\oplus S} = \sum_{i=1}^m f_i^{\oplus S} \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$$

since $f_i^{\oplus S} \in \mathsf{Orb}(f)$ if $f_i \in \mathsf{Orb}(f)$.

Finally, we observe that $1 \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$, which implies that the set is also closed under negation since $g \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ implies $\overline{g} = 1 \oplus g \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$. We proceed by induction on $m \leq n$, the number of variables on which f depends. If $m = 0$ then $f = 1$ (since $f \neq 0$) and we are done. So, by way of induction, assume that if g is any function depending on $m \geq 1$ variables then $1 \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(g))$. Let f be any function depending on $m + 1$ variables.

If x_i is any variable on which f depends, then we can write $f = x_i q + r$ as a polynomial over $\mathbb{F}_2$, where $q \neq 0, r$ are polynomials that do not depend on x_i. Consider the function $f^{\oplus i} = (1 + x_i)q + r$, which is also in $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ since $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ is closed under negating input variables. Adding these two polynomials together yields

$$f + f^{\oplus i} = x_i q + r + (1 + x_i)q + r = q.$$

Since $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ is closed under negating input variables, and since $(g + h)^{\oplus S} = g^{\oplus S} + h^{\oplus S}$, it follows that $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(q)) \subseteq \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$. By induction, $1 \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(q))$, completing the proof of the lemma. $\square$

5.2. Better Bounds for Catalytic Space

In this section we use the arguments from the previous section to improve the known bounds on *catalytic space*, a model first introduced in [BCK+14]. The next definition appears in [GKM15].

Definition 5.9. Let m be a positive integer and let f be a boolean function. An *m-catalytic branching program* for a function f is a branching program P with the following properties:

- The program P has m start nodes $s_1, s_2, \ldots, s_m$, m accept nodes $a_1, a_2, \ldots, a_m$, and m reject nodes $r_1, r_2, \ldots, r_m$.
- On any input x and for any $i \in [m]$, if $f(x) = 1$ then a computation path starting at the node s_i will end at the accept node a_i, and if $f(x) = 0$ then a computation path starting at s_i will end at the reject node r_i.

We note that this is more specialized than our definition of a branching program as the start nodes and end nodes are *paired*, whereas the more general notion of a branching program computing a multiset of functions allows computation paths beginning at any start node to stop at any sink node. This pairing property is closely related to catalytic space, thanks to the following proposition of Girard, Koucký, and McKenzie [GKM15].

Proposition 5.10 (Proposition 9 of [GKM15]). *Let f be a function that can be computed in space s using a catalytic tape of size $\ell \leq 2^s$. Then there is a 2^ℓ-catalytic branching program computing f of size $2^{\ell + O(s)}$.*

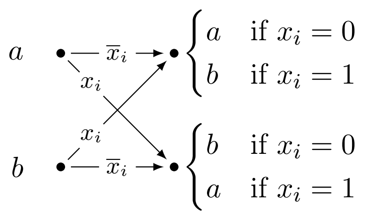

a
b
$\overline{x}_i$
x_i
x_i
$\overline{x}_i$
$\begin{cases} a & \text{if } x_i = 0 \\ b & \text{if } x_i = 1 \end{cases}$
$\begin{cases} b & \text{if } x_i = 0 \\ a & \text{if } x_i = 1 \end{cases}$

Figure 2: A swap gate, implemented by a branching program using 4 edges.

Potechin constructed a catalytic branching program computing f with $m = 2^{2^n-1}$ and total size $O(mn)$, and asked if the number of copies required can be reduced while maintaining the amortized bound of $O(n)$ [Pot17]. In this section we show the answer is *yes*, provided that the degree of f is small when representing it as a polynomial over $\mathbb{F}_2$.

Our catalytic branching program is an extensive modification of the catalytic branching program presented by Potechin, and is crucially modelled on the tools we developed in the previous section. We will construct our branching program out of copies of the following small component, which we call a *swap gate*.

Definition 5.11. A *swap gate* $\mathsf{swap}(a, b, i)$ is the function which takes as input two bits $a, b \in \{0, 1\}$, as well as an input variable x_i, and outputs (a, b) if $x_i = 0$ and (b, a) if $x_i = 1$.

Swap gates have three very nice properties that we will crucially use in our construction.

- *Reversible.* Swap gates are *reversible*: $\mathsf{swap}(\mathsf{swap}(a, b, i), i) = (a, b)$. In the branching program in Figure 2 this is represented by the fact that if we reverse all directions of the edges we get another branching program.
- *XOR-Invariance.* For any boolean function f, if $\mathsf{swap}(a, b, i) = (c, d)$ then $\mathsf{swap}(f \oplus a, f \oplus b, i) = (f \oplus c, f \oplus d)$. Indeed, invariance holds for *any* operator applied to the inputs.
- *Input Evaluation.* Let $g, h : \{0, 1\}^n \to \{0, 1\}$ be boolean functions, and suppose that $g^{\oplus i} = h$ for some $i \in [n]$. Then $\mathsf{swap}(g, h, i) = (g_{i\leftarrow 0}, g_{i\leftarrow 1})$. This is most easily seen by a case argument: if $x_i = 0$ then the first output is $g_{i\leftarrow 0}$ and the second output is $h_{i\leftarrow 0} = g_{i\leftarrow 1}$. On the other hand, if $x_i = 1$ then the first output is $h_{i\leftarrow 1} = g_{i\leftarrow 0}$ and the second output is $g_{i\leftarrow 1}$.

The next lemma is the main lemma used in our construction, and corresponds exactly to an instantiation of the argument in Theorem 5.5 as a catalytic branching program. We note that since our duality theorem (Theorem 4.1) does not apply to catalytic branching programs (only standard branching programs), we cannot immediately deduce anything about catalytic branching programs from it; so, instead we will need to proceed directly. In the next lemma we will discuss branching programs with some "standard" start nodes (labelled with 1 initially), and other start nodes "labelled" with 0. These nodes can be ignored, and are just a technical convenience when describing the proof.

Lemma 5.12. *For any boolean function $f : \{0,1\}^n \to \{0,1\}$ there is a branching program composed of swap gates that, starting from $|f^{-1}(1)|$ copies of 1 and $|f^{-1}(0)|$ copies of 0, computes every function in $\mathsf{Orb}(f)$. The size of the branching program is $2|\mathsf{Orb}(f)|D_{avg}(f)$.*

Proof. We essentially follow the proof of Theorem 5.5, using the inequalities in the proof to guide the construction of our branching program. As in that theorem, our proof is by induction on n, the number of variables on which f depends. If $n = 0$ then f is constant, and we can compute 0 or 1 by a single accept or reject node with 0 edges.

Now, by way of induction, suppose that T is a decision tree for f witnessing $D_{avg}(f)$, and suppose without loss of generality that x_n is the first variable queried by the tree. We can write $f = (f_0 \wedge \overline{x}_n) \vee (f_1 \wedge x_n)$ where $f_b = f_{n\leftarrow b}$. Again note that since T witnesses $D_{avg}(f)$,

$$D_{avg}(f) = \frac{D_{avg}(f_0)}{2} + \frac{D_{avg}(f_1)}{2} + 1.$$

By the induction hypothesis, for $b \in \{0,1\}$ there is a branching program P_b, composed only of swap gates, computing every function in $\mathsf{Orb}(f_b)$ with size $2|\mathsf{Orb}(f_b)|D_{avg}(f_b)$ from $|f_b^{-1}(1)|$ copies of 1 and $|f_b^{-1}(0)|$ copies of 0. Also, since f depends on x_n, negating x_n partitions the orbit $\mathsf{Orb}(f)$ into pairs $(g, g^{\oplus n})$, so, let B be the set of all such pairs. We have two cases, depending on the two outcomes of Lemma 5.4.

Case 1. $\mathsf{Orb}(f_0) = \mathsf{Orb}(f_1)$.

In this case, let $u = |\mathsf{Orb}(f)|/|\mathsf{Orb}(f_1)|$ be the positive integer such that $M(\mathsf{Orb}(f), n, b) = \{\{g_{n\leftarrow b} \mid g \in \mathsf{Orb}(f)\}\}$ can be partitioned into u copies of $\mathsf{Orb}(f_1)$. It is easier to see the proof in "reverse". That is, suppose that we had computed all of $\mathsf{Orb}(f)$ using a branching program B. For each pair $(g, g^{\oplus n}) \in B$, applying a swap gate $\mathsf{swap}(g, g^{\oplus n}, x_n)$ will output the pair $(g_0, g_1) = (g_b, g_b^{\oplus n})$ by the *Input Evaluation* property of the swap gate. Thus, by starting with all pairs $(g, g^{\oplus n}) \in B$, and applying swap gates to each pair, we will obtain (by Lemma 5.4) u copies of $\mathsf{Orb}(f_1)$. Now, by applying the reversibility property of swap gates, we can therefore compute all of $\mathsf{Orb}(f)$ from u copies of $\mathsf{Orb}(f_1)$ using an appropriate number of swap gates.

Formally, create u copies of the branching program P_1, which will now output u copies of the orbit $\mathsf{Orb}(f_1) = \mathsf{Orb}(f_0)$. By using the values $u_b = |\mathsf{Orb}(f)|/2|\mathsf{Orb}(f_b)|$ from Lemma 5.4, and using the final claim in that lemma, we can bound the total size of the branching program as

$$\begin{aligned}2|\mathsf{Orb}(f)| + u \cdot 2|\mathsf{Orb}(f_1)|D_{avg}(f_1) &= 2|\mathsf{Orb}(f)| + 2u_0|\mathsf{Orb}(f_1)|D_{avg}(f_1) + 2u_1|\mathsf{Orb}(f_1)|D_{avg}(f_1)\\ &= 2|\mathsf{Orb}(f)| + 2u_0|\mathsf{Orb}(f_0)|D_{avg}(f_0) + 2u_1|\mathsf{Orb}(f_1)|D_{avg}(f_1)\\ &= 2|\mathsf{Orb}(f)| + |\mathsf{Orb}(f)|D_{avg}(f_0) + |\mathsf{Orb}(f)|D_{avg}(f_1)\\ &= 2|\mathsf{Orb}(f)|\left(1 + \frac{D_{avg}(f_0)}{2} + \frac{D_{avg}(f_1)}{2}\right)\\ &= 2|\mathsf{Orb}(f)|D_{avg}(f)\end{aligned}$$

where we have also used the definition of average-case decision tree depth. Finally, we note that $|f^{-1}(b)| = u|f_1^{-1}(b)|$ for $b \in \{0,1\}$ by the above partitioning claim.

Case 2. $\mathsf{Orb}(f_0) \neq \mathsf{Orb}(f_1)$.

This case is virtually identical to the previous case. In this case let $u_b = |\mathsf{Orb}(f)|/2|\mathsf{Orb}(f_b)|$ be the positive integers such that $M(\mathsf{Orb}(f), n, b)$ can be partitioned into u_b copies of $\mathsf{Orb}(f_b)$ for $b \in \{0,1\}$. We once again proceed in "reverse". Suppose we had a branching program computing all of $\mathsf{Orb}(f)$. Again, for each pair $(g, g^{\oplus n}) \in B$, applying a swap gate $\mathsf{swap}(g, g^{\oplus n}, x_n)$ will output the pair (g_0, g_1), by the *Input Evaluation* property of the swap gate. Thus, by starting with *all* pairs $(g, g^{\oplus n})$ and applying swap gates, we will produce u_0 copies of $\mathsf{Orb}(f_0)$ and u_1 copies of $\mathsf{Orb}(f_1)$ by Lemma 5.4. Again, since swap gates are reversible, we can therefore compute all of $\mathsf{Orb}(f)$ from u_0 copies of $\mathsf{Orb}(f_0)$ and u_1 copies of $\mathsf{Orb}(f_1)$.

So, using the induction hypothesis, create u_0 copies of the branching program P_0 and u_1 copies of the branching program P_1. As described above, we can compute all of $\mathsf{Orb}(f)$ by using an additional $|\mathsf{Orb}(f)|/2$ swap gates. This means the total size of the branching program is

$$\begin{aligned}&2|\mathsf{Orb}(f)| + 2u_0|\mathsf{Orb}(f_0)|D_{avg}(f_0) + 2u_1|\mathsf{Orb}(f_1)|D_{avg}(f_1)\\ &= 2|\mathsf{Orb}(f)| + |\mathsf{Orb}(f)|D_{avg}(f_0) + |\mathsf{Orb}(f)|D_{avg}(f_1)\\ &= 2|\mathsf{Orb}(f)|D_{avg}(f).\end{aligned}$$

This completes the proof. □

We can use this lemma with the "XOR trick" in Theorem 5.7 to create a catalytic branching program computing f with much better complexity.

Theorem 5.13. *Let $f : \{0,1\}^n \to \{0,1\}$ be any boolean function and let $d = \deg_2(f)$ be the degree of f when represented as an $\mathbb{F}_2$ polynomial. Then f can be computed by an m-catalytic branching program with $m \leq 2^{\binom{n}{\leq d}-1}$ and with total size $O(mn)$.*

Proof. By Lemma 5.8, $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ is closed under XORing with f and also closed under the action of $\Pi_{[n]}$. Also by Lemma 5.8, $1 \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$, and so it follows that for any $x \in \{0,1\}^n$ we have

$$|\{g \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f)) : g(x) = 1\}| = |\{g \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f)) : g(x) = 0\}|$$

since for every $g \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$, $\overline{g} = 1 + g \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$. Let C be this number of 1s/0s, and note that $C = |\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))|/2$.

Let $\mathsf{Orb}(g_1), \mathsf{Orb}(g_2), \ldots, \mathsf{Orb}(g_m)$ be the decomposition of $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ under the action of $\Pi_{[n]}$. First, apply Lemma 5.12 to each orbit $\mathsf{Orb}(g_i)$ twice, creating two branching programs $P_{i,0}, P_{i,1}$ composed of swap gates that each compute all of $\mathsf{Orb}(g_i)$. Now, reverse all the swap gates in $P_{i,0}$ for each $i \in [m]$, creating a branching program that (intuitively) computes C copies of 1 and C copies of 0 from $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$

So, we now have two branching programs: one, by taking $P_{i,1}$ for $i \in [m]$, computing all of $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ from C copies of 1 and C copies of 0. The second, by taking $P_{i,0}$ for $i \in [m]$, gives a branching program computing C copies of 1 and C copies of 0 from $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$. Now, for each $g \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$, take the node computing $g \oplus f$ in the first branching program and *merge* it with the node taking g as input in the second branching program. Since the second program is composed entirely of swap gadgets, by the *XOR-invariance* property of the swap gadget the program will now output C copies of $1 \oplus f = \overline{f}$, and C copies of $0 \oplus f = f$. This yields a branching program computing C copies of f at nodes $a_1, a_2, \ldots, a_C$, and C copies of $\overline{f}$ at nodes $r_1, r_2, \ldots, r_C$; however, we do not have the *pairing* property between start nodes and the accept/reject nodes. To ensure the pairing property, we create two more copies of the branching program and run it in reverse (again, exploiting reversibility of the swap gate). We use the nodes $\{a_1, \ldots, a_C\}$ as the start nodes for one copy of the program, and $\{r_1, \ldots, r_C\}$ as the start nodes for the other copy of the program. Finally, the C start nodes of the program will be the C initial nodes labelled with 1.

Now we estimate the complexity of the program. The size of the program is at most $6\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))n$, since we take 3 copies of a branching program of size at most $2\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))n$ by Lemma 5.12. Letting $\deg_2(f) = d$ we see that $|\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))| \leq 2^{\binom{n}{\leq d}}$, since every function in $\mathsf{Orb}(f)$ has degree at most d and taking a linear span cannot increase the degree. This completes the proof. □

6. Open Problems

The work presented here suggests many open problems. Most pressingly: what other "direct-sum type" phenomena can we study using Strassen-type duality theorems? There are many natural direct-sum type problems in complexity theory that seem amenable to this technique (such as parallel repetition or information complexity, as discussed in the introduction). Is there a way to study query or proof complexity models in this framework?

A second group of questions concerns *monotone* circuit complexity. Unlike the case with non-monotone circuit complexity, we can use formal complexity measures to actually prove explicit exponential lower bounds for monotone circuits. For example, in contrast to the known $O(n)$ bounds for submodular complexity measures shown by Razborov [Raz92], one can use a *monotone* submodular complexity measure (the so-called *rank measure*

[Raz90]) to prove *strongly exponential* lower bounds on monotone circuit complexity for comparator circuits, boolean formulas, and switching networks [PR17, PR18]. All of our duality theorems also hold for monotone circuit complexity, and this shows that amortized monotone comparator circuit complexity can be strongly exponential! For this reason, we conjecture that *amortized monotone comparator circuit complexity* is simply equal to monotone comparator circuit complexity, and it is natural to wonder if this also holds for other monotone models.

A final natural question is whether or not it is possible to further improve non-uniform catalytic space complexity. Similar to the bounds proven by [Pot17], our catalytic space upper bounds achieve $O(n)$-size per copy of the function computed, while significantly decreasing the number of copies. Is it possible to further decrease the number of copies, perhaps while trading off into the amortized size (that is, increasing $O(n)$ to $n^{O(1)}$)?

A. An Explicit Complexity Measure

In this appendix we describe an explicit small branching program complexity measure to prove a simple property of the amortized branching program size. Recall that our duality theorem specialized to branching programs says that the amortized branching program size $\undertilde{\mathsf{BP}}$ is the pointwise maximum over all branching program complexity measures,

$$\undertilde{\mathsf{BP}}(f) = \max_{\mu \in M_{\mathrm{BP}}} \mu(f). \tag{6}$$

This raises the basic question whether $\undertilde{\mathsf{BP}}$ in an element of M_{BP}, that is, whether $\undertilde{\mathsf{BP}}$ is a branching program measure. Indeed, this would trivialize Equation 6. In Section 2.2 we argued that $\undertilde{\mathsf{BP}}$ is not itself a branching program measure, because it is not additive. Our argument relied on the existence of branching program measures $\mu, \mu', \mu'' : F_2 \to \mathbb{R}_{\geq 0}$ with the property that $\mu(x \wedge y) = 2$, $\mu'(x \wedge \overline{y})$, and $\mu''(\overline{x}) = 2$. We give a construction of such measures here. We note that via a similar argument and construction it can also be shown that the amortized comparator circuit size is not additive.

We present the measure here, which was found by computer search. From the construction of μ one can easily obtain μ' and μ'' by a natural permutation of μ's values. We describe μ in the following table, in which we identify boolean functions $f \in F_2$ with their set of one-inputs $f^{-1}(1) = \{x \in \{0,1\}^2 : f(x) = 1\}$:

f	$\mu(f)$	f	$\mu(f)$
$\emptyset$	2	$\{00, 01, 10, 11\}$	0
$\{00\}$	2	$\{01, 10, 11\}$	0
$\{01\}$	1	$\{00, 10, 11\}$	1
$\{10\}$	1	$\{00, 01, 11\}$	1
$\{11\}$	2	$\{00, 01, 10\}$	0
$\{00, 01\}$	1	$\{10, 11\}$	1
$\{00, 10\}$	1	$\{01, 11\}$	1
$\{00, 11\}$	2	$\{01, 10\}$	0

We finish by noting the remarkable symmetries that this measure μ has. Not only does it satisfy $\mu(f) = 2 - \mu(\overline{f})$, but $\mu(f)$ is also invariant under swapping the two inputs of f and under simultaneously negating the two inputs of f.

Acknowledgements

The authors thank Aaron Potechin, Avi Wigderson, James Cook, Ian Mertz, and Toniann Pitassi for helpful conversations, and thank Benny Appelbaum for pointers to the closely related papers [AARV21, AA20]. They also thank several anonymous reviewers for very careful readings of an earlier draft of the paper. R.R. was supported by NSERC, the Charles Simonyi Endowment, and indirectly supported by the National Science Foundation Grant No. CCF-1900460. J.Z. was supported by the National Science Foundation Grant No. CCF-1900460. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[AA20] Benny Applebaum and Barak Arkis. On the power of amortization in secret sharing: d-uniform secret sharing and CDS with constant information rate. *ACM Trans. Comput. Theory*, 12(4):24:1–24:21, 2020. doi:10.1145/3417756.

[AARV21] Benny Applebaum, Barak Arkis, Pavel Raykov, and Prashant Nalini Vasudevan. Conditional disclosure of secrets: Amplification, closure, amortization, lower-bounds, and separations. *SIAM J. Comput.*, 50(1):32–67, 2021. doi:10.1137/18M1217097.

[AW21] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 522–539. SIAM, 2021. arXiv:2010.05846, doi:10.1137/1.9781611976465.32.

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989. doi:10.1016/0022-0000(89)90037-8.

[BC92] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992. doi:10.1137/0221006.

[BCK$^+$14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: Catalytic space. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC 2014, page 857–866, 2014. doi:10.1145/2591796.2591874.

[BR14] Mark Braverman and Anup Rao. Information equals amortized communication. *IEEE Trans. Inf. Theory*, 60(10):6058–6069, 2014. doi:10.1109/TIT.2014.2347282.

[CM20] James Cook and Ian Mertz. Catalytic approaches to the tree evaluation problem. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 752–760. ACM, 2020. doi:10.1145/3357713.3384316.

[CVZ18] Matthias Christandl, Péter Vrana, and Jeroen Zuiddam. Universal points in the asymptotic spectrum of tensors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 289–296. ACM, 2018. doi:10.1145/3188745.3188766.

[Fek23] Michael Fekete. Über die verteilung der wurzeln bei gewissen algebraischen gleichungen mit ganzzahligen koeffizienten. *Mathematische Zeitschrift*, 17(1):228–249, 1923.

[FKNN95] Tomás Feder, Eyal Kushilevitz, Moni Naor, and Noam Nisan. Amortized communication complexity. *SIAM J. Comput.*, 24(4):736–750, 1995. doi:10.1137/S0097539792235864.

[FR18] Zoltan Furedi and Imre Z. Ruzsa. Nearly subadditive sequences, 2018. arXiv:1810.11723.

[Fri17] Tobias Fritz. Resource convertibility and ordered commutative monoids. *Math. Struct. Comput. Sci.*, 27(6):850–938, 2017. doi:10.1017/S0960129515000444.

[Fri20] Tobias Fritz. A generalization of Strassen's Positivstellensatz. *Communications in Algebra*, pages 1–18, 2020. doi:10.1080/00927872.2020.1803344.

[Gar85] Philip Alan Gartenberg. *Fast rectangular matrix multiplication.* PhD thesis, UCLA, 1985.

[GKM15] Vincent Girard, Michal Koucký, and Pierre McKenzie. Nonuniform catalytic space and the direct sum for space. In *Electron. Colloq. Comput. Complex. (ECCC)*, volume 22, page 138, 2015. URL: http://eccc.hpi-web.de/report/2015/138.

[GR20] Anna Gál and Robert Robere. Lower bounds for (non-monotone) comparator circuits. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020*, volume 151 of *LIPIcs*, pages 58:1–58:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.58.

[Hås98] Johan Håstad. The shrinkage exponent of De Morgan formulas is 2. *SIAM J. Comput.*, 27(1):48–64, 1998. doi:10.1137/S0097539794261556.

[JV20] Asger Kjærulff Jensen and Péter Vrana. The asymptotic spectrum of LOCC transformations. *IEEE Trans. Inf. Theory*, 66(1):155–166, 2020. doi:10.1109/TIT.2019.2927555.

[Khr72] V.M. Khrapchenko. A method of obtaining lower bounds for the complexity of π-schemes. *Math. Notes Acad. Sci. USSR*, 11:474–479, 1972.

[LG14] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC 2014)*, pages 296–303. 2014. doi:10.1145/2608628.2608664.

[LZ21] Yinan Li and Jeroen Zuiddam. Quantum asymptotic spectra of graphs and non-commutative graphs, and quantum Shannon capacities. *IEEE Transactions on Information Theory*, 67(1):416–432, 2021. doi:10.1109/TIT.2020.3032686.

[Pot17] Aaron Potechin. A note on amortized branching program complexity. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, volume 79, pages 4:1–4:12, 2017. doi:10.4230/LIPIcs.CCC.2017.4.

[PR17] Toniann Pitassi and Robert Robere. Strongly exponential lower bounds for monotone computation. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1246–1255. ACM, 2017. doi:10.1145/3055399.3055478.

[PR18] Toniann Pitassi and Robert Robere. Lifting nullstellensatz to monotone span programs over any field. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium* *on Theory of Computing, STOC 2018*, pages 1207–1219. ACM, 2018. doi:10.1145/3188745.3188914.

[Raz90] Alexander A. Razborov. Applications of matrix methods to the theory of lower bounds in computational complexity. *Combinatorica*, 10(1):81–93, 1990. doi:10.1007/BF02122698.

[Raz92] Alexander A. Razborov. On submodular complexity measures. In *Poceedings of the London Mathematical Society symposium on Boolean function complexity*, pages 76–83, 1992. URL: https://dl.acm.org/doi/10.5555/167687.167709.

[Raz98] Ran Raz. A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803, 1998. doi:10.1137/S0097539795280895.

[RPRC16] Robert Robere, Toniann Pitassi, Benjamin Rossman, and Stephen A. Cook. Exponential lower bounds for monotone span programs. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016*, pages 406–415. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.51.

[Sch99] Alexander Schrijver. *Theory of linear and integer programming.* Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[Sha56] Claude E. Shannon. The zero error capacity of a noisy channel. *Institute of Radio Engineers, Transactions on Information Theory*, IT-2(September):8–19, 1956. doi:10.1109/TIT.1956.1056798.

[Str86] Volker Strassen. The asymptotic spectrum of tensors and the exponent of matrix multiplication. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, FOCS 1986, pages 49–54, Washington, DC, USA, 1986. IEEE Computer Society. doi:10.1109/SFCS.1986.52.

[Str87] Volker Strassen. Relative bilinear complexity and matrix multiplication. *J. Reine Angew. Math.*, 375/376:406–443, 1987. doi:10.1515/crll.1987.375-376.406.

[Str88] Volker Strassen. The asymptotic spectrum of tensors. *J. Reine Angew. Math.*, 384:102–152, 1988. doi:10.1515/crll.1988.384.102.

[Sub90] Ashok Subramanian. *The computational complexity of the circuit value and network stability problems.* PhD thesis, Stanford University, 1990.

[Tal14] Avishay Tal. Shrinkage of De Morgan formulae by spectral techniques. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 551–560. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.65.

[Vra20] Péter Vrana. A generalization of Strassen's spectral theorem, 2020. arXiv:2003.14176.

[Weg87] Ingo Wegener. *The complexity of boolean functions*. Wiley-Teubner, 1987.

[Zui18] Jeroen Zuiddam. *Algebraic complexity, asymptotic spectra and entanglement polytopes*. PhD thesis, University of Amsterdam, 2018.

[Zui19] Jeroen Zuiddam. The asymptotic spectrum of graphs and the Shannon capacity. *Comb.*, 39(5):1173–1184, 2019. doi:10.1007/s00493-019-3992-5.