Electronic Colloquium on Computational Complexity, Report No. 35 (2021)

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

March 12, 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, 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 show that the answer is yes: 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.

ISSN 1433-8092

Contents

1. Introduction

An ubiquitous theme in theoretical computer science and related fields 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 compute 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

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

where mT represents the task of performing T in parallel m times[1]. 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 an application of Fekete's Lemma[2], and we call the cost $\utilde{C}(T)$ the *amortized complexity* of computing T.

In many settings, amortized complexity is studied *explicitly* as a phenomenon of interest. For example:

- 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].

[1] 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 $m \to \infty$.

[2] 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).

However, sometimes amortized complexity can turn out to be the right parameter to study even if it is not apparent from the definition of the problem. A key example is 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 we must read all of the $O(n^2)$ 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, AW20]. At a first glance, this problem does not have a direct-sum 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

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

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 preorders on semirings [Str86, Str87, Str88], now termed *Strassen duality* and an active area of study [CVZ18, Zui18, Fri20, Vra20]. In the 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.) 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
$$\mu(\langle n \rangle) \leq n.^{3}$$

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

These maps μ are called *spectral points*, 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 1.1 (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*.

Readers who are familiar with circuit complexity may feel that the above description of spectral points is somewhat familiar to them — this is a for very good reason! Recall that a *boolean formula* is a 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 $f : \{0,1\}^n \to \{0,1\}$. Since all gates have fan-out 1, the topology of a boolean formula is a binary tree, and we therefore measure the size of the formula by the number of leaves in the tree. For any boolean function f we let $\mathsf{F}(f)$ be the size of the smallest boolean formula computing f. We note that it is one of the long-standing open problems in circuit complexity to prove superpolynomial (indeed, even super-*cubic* [Hås98, Tal14]) lower bounds on $\mathsf{F}(f)$ for *any* explicit boolean function f.

A classic approach for proving lower bounds on the size of formulas uses the so-called *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 properties:

- μ is *monotone* 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,

$$\mu(\ell) \leq 1.$$

Observe that 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 $\mathcal{C}$ denote the family of all formal complexity measures, then an easy induction on formulas shows that for any boolean function f and any formal complexity measure μ

$$\mathsf{F}(f) = \max_{\mu \in \mathcal{C}} \mu(f).$$

Now, given the obvious analogy between Strassen duality and formal complexity measures, it is very natural to ask: *What is going on here?* Or, less informally:

Question. Is there a deeper connection between Strassen Duality and boolean circuit complexity?

The main message of this paper is an answer to this question: **Yes**!

1.1. Our Results

Duality Theorems for Amortized Circuit Complexity. In this paper we study the *amortized circuit complexity* of boolean functions.

To describe our first main theorem, we will proceed with a special case. Let $\mathcal{F}$ be any family of boolean circuits composed of gates chosen from some finite gate set G. Given a *multiset* of functions $A = \{\{f_1, f_2, \ldots, f_m\}\}$, we define the $\mathcal{F}$*-circuit size* of A to be the minimum number of copies of input literals[4] required by any circuit in order to output all functions in A, and denote it by $\mathcal{F}(A)$. We then define the *amortized circuit complexity* of computing $f : \{0,1\}^n \to \{0,1\}$ to be

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

where $m \cdot f$ represents the task of computing m copies of f on the *same* input $x \in \{0,1\}^n$. Just as with formulas, for the family $\mathcal{F}$ we can define a corresponding family of formal complexity measures: these are the functions

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

such that

- μ is *monotone* with respect to the gates of $\mathcal{F}$. That is, for any gate $g \in G$ applied to functions $f_1, f_2, \ldots, f_m$ and producing functions $g_1, g_2, \ldots, g_n$,
$$\sum_{i=1}^{n} \mu(g_i) \leq \sum_{i=1}^{m} \mu(f_i).$$
- μ is *normalized*: for any input literal ℓ, $\mu(\ell) \leq 1$.

Let $X_{\mathcal{F}}$ denote the set of all $\mathcal{F}$-formal complexity measures. Our first main result is a *duality theorem* for amortized circuit complexity *à la* Strassen.

Theorem 1.2. *Let $\mathcal{F}$ be any family of boolean circuits composed of gates from some finite gate set G. For any boolean function $f : \{0,1\}^n \to \{0,1\}$,*

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

[4]This is the standard measure of size for many circuit models that *cannot* copy intermediate computations, such as boolean formulas, branching programs, and comparator circuits. Our general theorem statement can also handle other standard definitions of size; this example is just given for expository purposes.

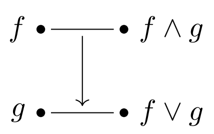


Figure 1: A comparator gate.

The above result for formulas follows since amortized formula size is just formula size. In fact, our main theorem (Theorem 3.1) is actually significantly more general than Theorem 1.2, as it can handle various generalizations of circuits, allowing more general measures of "size", gates with "filters" that only allow particular boolean functions as inputs, and other modifications (cf. Section 3).

As we have alluded to in the previous section, our main theorem was directly inspired by Strassen's theory of asymptotic spectra for semirings. Indeed, our main technical theorem (Theorem 3.5) can morally be interpreted as a special case of Strassen's theory where the underlying algebra happens in a *semigroup* instead of a *semiring*, which turn out to completely capture computation by standard circuit models. We give a new proof of Strassen's duality for semigroups using linear programming duality that is significantly simpler than the more general duality theorem for semirings, which we hope will be of independent interest (see Section 3 for more details).

To capture one nice corollary of our duality theorem, consider the model of *comparator circuits*. These are circuits composed of two types of gates: *comparator gates*, which take in as input two bits a, b and output the bits in sorted order $(a \wedge b, a \vee b)$ (see Figure 1). The corresponding family of formal complexity measures satisfy the following inequalities, for all boolean functions f, g and any input literal ℓ,

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

These formal complexity measures are known in the literature as *submodular complexity measures*, since the first inequality is a form of the submodular law. Submodular complexity measures were introduced and studied by Razborov [Raz92] (independently of comparator circuits), and he showed that a complexity measure he had earlier defined — called the *rank measure* [Raz90] — was submodular. In the same paper, by a simple and clever argument, Razborov also proved that *every* submodular complexity measure μ satisfies $\mu(f) = O(n)$, showing that these measures cannot be used to provide strong lower bounds.

Theorem 1.3 (Theorem 2 in [Raz92])**.** *For any submodular complexity measure μ and any boolean function f depending on n variables, $\mu(f) = O(n)$.*

Therefore, as an immediate corollary of our duality theorem, we obtain *linear* upper bounds on the amortized comparator circuit complexity of any function f.

Corollary 1.4. *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.

We note that this also follows from the results of Potechin [Pot17] (although, it is not directly observed in Potechin's paper), who explicitly constructed a branching program witnessing that amortized branching program complexity is $O(n)$; however, Potechin's argument is notably more complex than Razborov's. We also note that this result is quite remarkable since comparator circuits *cannot* copy intermediate computations, and so there is no way to create many copies of f from one copy [Sub90].

Amortized Complexity and Catalytic Algorithms. Next, let μ be a submodular (or "comparator circuit") complexity measure, and suppose that we had proved the inequality $\mu(f) \leq \mu(g)$ held for μ. 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 the notion of *catalytic algorithms*. That is, we can consider a comparator circuit C which takes an extra input h, as some arbitrary boolean function, and outputs another "fresh" copy of h at the end of its computation (thus, h acts as a "catalyst", using terminology borrowed from chemical reactions).

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 is easy to show 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 Theorem 3.1 (this is a nice side-effect of our simplified proof of Strassen duality for semigroups). We again refer to Section 3 for details.

Improved Upper Bounds for Catalytic Space by Symmetry. The notion of catalytic circuit size that we have introduced above is similar to (but not the same as) the notion of *catalytic space* that has been recently studied in a sequence of papers [BCK$^+$14, GKM15, Pot17].

Let us recall the notion of catalytic space, introduced by Buhrman et al. [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 — this is in contrast to our notion of a catalytic circuit above, which is allowed to *depend* on its catalyst. Buhrman et al. [BCK+14] showed the surprising result 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]. 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 computing f once of size s, we can create an m-catalytic branching program for f with size $O(ms)$, simply by repeating the branching program independently m times.

We first note that our above duality theorem *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 more general than the definition of catalytic branching programs in that it does not require the above “pairing property” between source nodes and sink nodes. Rather, on an input x, a computation path starting at s_i can end up at any sink node it wants.

With regards to 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. We give an answer both to this question and also to Girard, Koucký, and McKenzie’s question above: it is possible to reduce both the number of copies and the total size whenever f has low degree as an $\mathbb{F}_2$-polynomial.

Theorem 1.5. *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 directly imply this theorem, it played an important role in discovering the proof, as we now sketch. 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 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 lemma 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 4.12). Then, by an extension of Potechin's construction (utilizing 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 4.13), where $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ is the linear span of the functions in $\mathsf{Orb}(f)$ when treated as vectors 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 4 for details.

1.2. Related Work

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 the combinatorial notion introduded 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. Extensions of the Strassen duality theory in several directions have been introduced in [Fri20] and [Vra20] recently.

1.3. 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 preorders. Our introduction uses three running examples: *boolean formulas*, *branching programs*, and *comparator circuits*. In Section 3 we prove our main duality theorem, as well as our theorem characterizing "catalytic size". In Section 4 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 *preorders*.

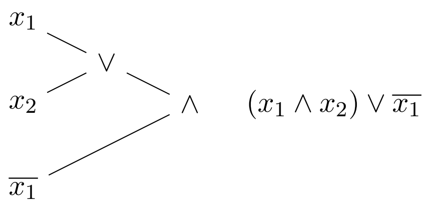
Definition 2.1. A *preorder* on a set S is a relation $P \subseteq S^2$ that is *reflexive*, in that $(a,a) \in P$ for every $a \in S$, and *transitive* in that $(a,b) \in P$ and $(b,c) \in P$ implies $(a,c) \in P$ for all $a,b,c \in S$. If P is a pre-order then we will use the notation $a \leq_P b$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

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

By induction over the tree-structure of the 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 $\mathsf{F}(f)$ *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

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

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 *preorder* 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 be 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

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

Second, for any literal ℓ, let $\{\ell\} \le \{\mathbf{1}\}$. Finally, let $\preceq_{\mathrm{F}} \supseteq \le$ be the smallest pre-order containing $\le$ 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, for example, the inequality $\{f\wedge g\} \preceq_{\mathrm{F}} \{f,g\}$ 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* formulas 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 preorder $\preceq_{\mathrm{F}}$ 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).

How does the preorder $\preceq_{\mathrm{F}}$ relate to the formal complexity measures μ that we defined ealier? We first naturally extend the formal complexity measures μ to functions $\mathbb{N}^F \to \mathbb{R}_{\ge 0}$ by saying, for any multiset $A\in S$, that $\mu(A) = \sum_{f\in A}\mu(f)$. These functions are 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

$$\{\{a_1, \dots, a_m\}\} + \{\{b_1, \dots, b_k\}\} \coloneqq \{\{a_1, \dots, a_k, b_1, \dots, b_k\}\},$$

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

How does the preorder $\preceq_{\mathrm{F}}$ relate to the formula size? A central concept here and later is the notion of *rank*. We define the *rank* $\mathrm{R}(A)$ of a collection of Boolean functions $A \in S$, with respect to the pre-order $\preceq_{\mathrm{F}}$, as the smallest number $n \in \mathbb{N}$ such that $A \preceq_{\mathrm{F}} \mathbf{n} \coloneqq \{\{\mathbf{1}, \dots, \mathbf{1}\}\}$, where in the multiset on the right-hand side the element $\mathbf{1}$ appears n times. From how we have set up the pre-order (in particular, from the rule $\{\ell\} \leq \{\mathbf{1}\}$ and the fact that any function can be computed by *some* boolean formula) we see first of all that the rank is finite, and second of all that the rank R precisely equals formula size F (which we defined as the smallest number of leaf nodes in any boolean formula computation). Naturally, we define the *amortized rank* of a boolean function f as $\underset{\sim}{\mathrm{R}}(f) \coloneqq \lim_{m\to\infty} \mathrm{R}(m \cdot f)/m$, so that $\underset{\sim}{\mathrm{R}}$ equals $\underset{\sim}{\mathsf{F}}$, and we have the important property that

$$\mu(f) \leq \underset{\sim}{\mathrm{R}}(f) = \mathrm{R}(f).$$

Again, in richer circuit models to come the equality between $\underset{\sim}{\mathrm{R}}(f)$ and $\mathrm{R}(f)$ may become a strict inequality.

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

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

- Second, the preorder is *bounded*, in the sense that every boolean function has a finite rank. This corresponds to the circuit model being *complete*, in that t can compute any boolean function.

- Third, the preorder 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 preorder* 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 *branching programs* as preorders. 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.

We offer two equivalent descriptions of branching program. A *branching program* is a circuit that computes a collection of boolean functions using two types of gates: the query gate, which takes as input a boolean function f and outputs $f \wedge x_i$ and $f \wedge \overline{x_i}$ for some variable x_i, and the OR gate, which takes as input two boolean functions f and g and outputs $f \vee g$. We measure the size of a branching program as the number of edges in the naturally associated directed graph. The branching program size of a collection of boolean functions is the smallest size of any branching program computing these functions. We denote the branching program size of a collection of boolean functions $A \in S$ by $\mathsf{BP}(A)$.

An equivalent description of branching programs is as follows. Consider 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 $f_2 = \overline{x_1} \vee x_2$:

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

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 $f_2 = ((\overline{x_1} \vee x_2) \wedge \overline{x_1}) \vee ((\overline{x_1} \vee x_2) \wedge \overline{x_1})$:

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

Note that, we do not require the start nodes s_i and output nodes a_j to be paired up in any way, that is, we do not care from what start node a path starts that reaches an output node. Also, we do not require the output nodes to come in accept–reject pairs a_j, r_j as is sometimes done in the literature. That is, 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 models in the literature.

Neither boolean formulas nor branching programs can directly copy intermediate results, in the sense that neither model can simulate a *copy gate* $x \mapsto (x, x)$ [Sub90]. Note, however, that 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 the query gates output *two* boolean functions. Thus 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. It is thus meaningful to introduce the *amortized branching program size* of a boolean function f as $\utilde{\mathsf{BP}}(f) = \lim_{m\to\infty} \mathsf{BP}(m \cdot f)/m$ where $m \cdot f = \{\{f, \dots, f\}\}$ is the multiset in which f appears m times. (Interestingly, $\utilde{\mathsf{BP}}(f)$ can be strictly smaller than $\mathsf{BP}(f)$, as we will discuss later!)

For branching programs a central concept is that of a *branching program measure.* This is any function $\mu : F \to \mathbb{R}_{\geq 0}$ such that for any literal ℓ and any $f, g \in F$:

- $\mu(\ell) \leq 1$,
- $\mu(f \vee g) \leq \mu(f) + \mu(g)$,
- $\mu(f \wedge x_i) + \mu(f \wedge \overline{x_i}) \leq \mu(f) + 2$.

By induction over the branching program we see that for any boolean function f and any branching program measure μ the branching program complexity of f is at least $\mu(f)$. Not only that is true, but it is even true that $\mu(f) \leq \utilde{\mathsf{BP}}(f)$,

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

We now give the preorder point of view for branching programs.

Definition 2.3. 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

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

Second, for any literal ℓ define $\{\ell\} \leq \{\mathbf{1}\}$. Finally, we let $\preceq_{\mathrm{BP}} \supset \leq$ 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 preorder $\preceq_{\mathrm{BP}}$ "builds" branching programs in the sense 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, $\{\{f_1, \ldots, f_m\}\} \preceq_{\mathrm{BP}} \{\{g_1, \ldots, g_k, \mathbf{1}, \ldots, \mathbf{1}\}\}$ 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.

How does $\preceq_{\mathrm{BP}}$ relate to the branching program size BP and the amortized branching program size $\utilde{\mathsf{BP}}$? We define the rank $\mathrm{R}(A)$ of a collection of boolean functions $A \in S$ with respect to $\preceq_{\mathrm{BP}}$ as the minimum $n \in \mathbb{N}$ such that $A \preceq_{\mathrm{BP}} n$. From the definition of $\preceq_{\mathrm{BP}}$ we see that R is equal to branching program size BP. We define the *amortized rank* by $\utilde{\mathrm{R}}(f) = \lim_{m\to\infty} \mathrm{R}(m \cdot f)/m$ where $m \cdot f = \{\{f, \ldots, f\}\}$ is the multiset in which f appears m times. The amortized rank thus equals the *amortized branching program complexity* $\utilde{\mathsf{BP}}$. We thus have the important property again that

$$\mu(f) \leq \utilde{\mathrm{R}}(f) \leq \mathrm{R}(f).$$

Besides the branching program measures lower bounding the amortized branching program size, we point out another interesting property. If $\mu(f) \leq \mu(g)$ for boolean functions f and g, then for any boolean function h it also holds that $\mu(\{\{f, h\}\}) = \mu(f) + \mu(h) \leq \mu(g) + \mu(h) = \mu(\{\{g, h\}\})$. This statement is related to the phenomenon of a *catalyst*: 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 inequaly $\{\{f, h\}\} \preceq_{\mathrm{BP}} \{\{g, h\}\}$. The name catalyst is fitting because h is used as an *input* of the program, but also appears as an *output* of the program. From this observation we are naturally lead to define the *catalytic branching program complexity* $\mathrm{R}_{\mathrm{cat}}(f) = \mathsf{BP}_{\mathrm{cat}}(f)$ 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 we have

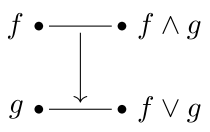
$$\max_{\mu} \mu(f) \leq \utilde{\mathrm{R}}(f) \leq \mathrm{R}_{\mathrm{cat}}(f) \leq \mathrm{R}(f).$$

The main goal in Section 3 will be to understand which of these inequalities may be strict and when.

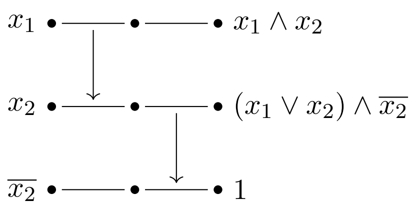
Finally, observe that the branching program preorder $\preceq_{\mathrm{BP}}$ has all the properties that we saw for the boolean formula preorder $\preceq_{\mathrm{F}}$: it is bounded (any function can be computed by a branching program), nonnegative (we can forget outputs), finitely generated (there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

2.3. Comparator Circuits

A *comparator circuit* is a circuit that computes a collection of boolean functions from a collection of literals using negation gates and a *comparator gate* which, given two boolean functions f and g as input, outputs the boolean functions $f \vee g$ and $f \wedge g$. We will 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 $\mathsf{CC}(A)$. In a figure, if we draw the comparator gate as



then we can compose comparator gates to, for instance, compute the collection of functions $\{x_1 \wedge x_2, (x_1 \vee x_2) \wedge \overline{x_2}, 1\}$ via the following comparator circuit of size three:



For any boolean function f we let $\underset{\sim}{\mathsf{CC}}(f) \coloneqq \lim_{m\to\infty} \mathsf{CC}(m \cdot f)/m$ be the *amortized comparator circuit size*, where $m \cdot f = \{\{f, \ldots, f\}\}$ is the multiset in which f appears m times.

Definition 2.4. Define the ordering $\preceq_{\mathrm{CC}}$ on $S = \mathbb{N}^{F \cup \{\mathbf{1}\}}$ as follows. First, for any boolean functions f, g define

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

Second, for any literal ℓ define $\{\ell\} \leq \{\mathbf{1}\}$. Finally, we let $\preceq_{\mathrm{CC}} \supset \leq$ be the smallest ordering containing $\leq$ satisfying 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, *mutatis mutandis*. Rank R and amortized rank $\underset{\sim}{\mathrm{R}}$ equal comparator circuit size CC and amortized comparator circuit size $\underset{\sim}{\mathsf{CC}}$. As with formulas and branching programs, there is a corresponding complexity measure for comparator circuits called a *submodular complexity measure*.

Definition 2.5. 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 we see that for any boolean function f and any submodular measure μ the comparator circuit size of f is at least $\mu(f)$ (see, e.g. [RPRC16]). But not only that is true, any submodular complexity measure μ even lower bounds the asymptotic comparator circuit size $\underset{\sim}{\mathsf{CC}}$,

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

The comparator circuit preorder $\preceq_{\mathrm{CC}}$ has all the properties that we saw for the boolean formula preorder $\preceq_{\mathrm{F}}$ and the branching program preorder $\preceq_{\mathrm{BP}}$: 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 preorder (we may compose comparator circuits).

2.4. Abstract Framework

Having seen three concrete circuit models described as preorders, and having seen the recurring concept of formal complexity measures and rank functions, 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.

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 $S = \mathbb{N}^F$ 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. We may identify the standard basis elements of S with the elements of F. Under this identification we may think of the elements of S as formal linear combinations of the 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.

Preorders

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 preorder 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 preorder, which we call the *pointwise preorder*. This preorder 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 $a_f \leq b_f$. The pointwise preorder will play an important role in our proofs. Since this is arguably the standard preorder on $\mathbb{N}^F$ we will denote the pointwise preorder simply by $\leq$.

We are interested in preorders on S of a special kind. These preorders 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 preorders.

We begin by defining a notion of *boundedness* for preorders. This notion will allow us to define a *rank* function on S later.

Definition 2.6. For any subset $U \subseteq S$ we say that a preorder P on S is *U-bounded* if for every $s \in S$ there are elements $u_1, \ldots, u_n \in U$ such that $s \leq_P u_1 + \cdots + u_n$.

Secondly, we will be interested in preorders that have the following *composition* property.

Definition 2.7. We call a preorder P a *semigroup preorder* if for every $a, b, c, d \in S$ it holds that if $a \leq_P b$ and $c \leq_P d$, then $a + c \leq_P b + d$.

For example, the pointwise preorder is a semigroup preorder.

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

Definition 2.8. We say that P is *non-negative* if for every $a, b \in S$ it holds that $a \leq_P a + b$.

Let us pause for a moment in order to discuss what the non-negativity property means. 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.

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

Lemma 2.9. *Let P be a semigroup preorder. Then P extends the pointwise preorder if and only if for every $s \in S$ it holds that $0 \leq_P s$.*

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

Finally, we introduce a natural notion of *finiteness* for preorders.

Definition 2.10. We say that a preorder 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 $a \le_P b$ can be obtained as a non-negative integer combination of the finite collection of generating inequalities $a_i \le_P b_i$.

When a preorder P has all the above properties, we call it a *good* preorder:

Definition 2.11 (Good preorder)**.** Given $S = \mathbb{N}^F$ and a subset $U \subseteq S$, we say that a preorder P on S is *good* if P is a non-negative, finitely generated, U-bounded, semigroup preorder.

Example 2.12. Let $S = \mathbb{N}^k$. Let $U = \{u\}$ with $u = (1, \ldots, 1)$. Let P be the pointwise preorder on S. Then P is good.

More generally, again for $S = \mathbb{N}^k$, let U be any collection of vectors such that for every $1 \le i \le k$ there is an element $u \in U$ such that $u_i > 0$. Again let P be the pointwise preorder on S. Then P is good.

We finish by connecting back to our discussion of the concrete circuit models (boolean formulas, branching programs, comparator circuits) and pointing out that in those situations we are considering the semigroup $S = \mathbb{N}^{F \cup \{\mathbf{1}\}}$ where F is the set of all boolean functions $\{0,1\}^n \to \{0,1\}$, the element $\mathbf{1}$ is a formal symbol, and $U = \{\mathbf{1}\}$ consists only of the element $\mathbf{1}$.

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 preorder P on the set $S = \mathbb{N}^F$, we say a function $\mu : S \to \mathbb{Q}_{\ge 0}$ is *P-monotone* if for any $a, b \in S$ if $a \le_P b$, then $\mu(a) \le \mu(b)$. 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 *U-normalized* for a subset $U \subseteq S$ if for every $u \in U$ it holds that $\mu(u) \le 1$. When U is clear from the context we will say that μ is an *additive monotone* to mean that it is additive, monotone and U-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

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

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

Catalysts

The observation that additive monotones cannot distinguish $a \le_P b$ from $a+c \le_P b+c$ gives rise to the notion of a catalyst, which will play an important role in our duality theorem. Consider an inequality of the form $a + t \ge_P b + t$ 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*.

The knowledgeable reader may now be wondering 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 non-negative finitely generated semigroup preorder on S, there is a simple characterization of catalytic inequalities in P in terms of the pointwise preorder.

Lemma 2.13. *Let P be a good preorder 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 $a + t \ge_P b + t$.*
2. *There exist non-negative integers $y_i \in \mathbb{N}$ such that $a + \sum_i y_i(v_i - w_i) \ge b$ in the pointwise preorder.*

Proof. Suppose that $a + t \ge_P b + t$. 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, $a + \sum_i y_i(v_i - w_i) \ge b$.

On the other hand, suppose that $a + \sum_i y_i(v_i - w_i) \ge b$. Then $a + \sum_i y_i v_i \ge b + \sum_i y_i w_i$. From combining the generating inequalities $w_i \ge_P v_i$ we also have $a + \sum_i y_i w_i \ge_P a + \sum_i y_i v_i$. Since P extends the pointwise preorder, it then follows that $a + \sum_i y_i w_i \ge_P b + \sum_i y_i w_i$. $\square$

We finish by isolating a simple trick for catalytic inequalities that we will use together with Lemma 2.13 later.

Lemma 2.14. *Let P be a semigroup preorder. If $a + t \geq_P b + t$, then $ma + t \geq_P mb + t$ for every $m \in \mathbb{N}$.*

Proof. We give a proof 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 $ma + t \geq_P mb + t$. $\square$

Ranks

As we have seen in our earlier concrete discussion of circuit models as preorders, the notion of rank should be thought of as a complexity meausure. Here we define it generally, and look precisely into two variations: the amortized rank and the catalytic rank.

Let P be a non-negative semigroup preorder on S that is U-bounded for some subset $U \subseteq S$. We will use P and the elements of U 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 2.15. 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 there exist elements $u_1, \ldots, u_n \in U$ such that $s \leq_P u_1 + \cdots + u_n$. 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*, and elements $u_1, \ldots, u_n \in U$ such that $s + t \leq_P u_1 + \cdots + u_n + t$. 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 $\mathrm{R}(s)$ tells us the *cost* of the element s in terms of the elements of U. The catalytic rank is a variation on rank that allows for an extra element to act as a catalyst. Finally, 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. This follows directly from the assumption that P is a semigroup preorder. 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 2.16. $\underset{\sim}{\mathrm{R}}(s) \leq \mathrm{R}_{\mathrm{cat}}(s) \leq \mathrm{R}(s)$.

Proof. Suppose that $\mathrm{R}(s) = n$. Then $s \leq_P u_1 + \cdots + u_n$ for some $u_i \in U$. This gives the inequality $s + t \leq_P u_1 + \cdots + u_n + t$ with the trivial catalyst $t = 0$, and so $\mathrm{R}_{\mathrm{cat}}(s) \leq n$.

Suppose that $\mathrm{R}_{\mathrm{cat}}(s) = n$. Then $s + t \leq_P u_1 + \cdots + u_n + t$ for some $u_i \in U$ and $t \in S$. From Lemma 2.14 it follows that $ms + t \leq_P m(u_1 + \cdots + u_n) + t$ for every $m \in \mathbb{N}$. We have $ms \leq_P ms + t$, since P is non-negative. Thus $\mathrm{R}(ms) \leq mn + \mathrm{R}(t)$, and so $\underset{\sim}{\mathrm{R}}(s) \leq n$. $\square$

Depending on the choice of S, U and P, the three rank functions R, $\mathrm{R}_{\mathrm{cat}}$ and $\underset{\sim}{\mathrm{R}}$ may or may not coincide with each other. We say that the rank function R is *additive* if $\mathrm{R}(s+t) = \mathrm{R}(s) + \mathrm{R}(t)$ for every $s, t \in S$. We say that R is *additive on multiples* if $\mathrm{R}(ns) = n\,\mathrm{R}(s)$ for every $n \in \mathbb{N}$ and $s \in S$. One verifies directly that R is additive on multiples if and only if $\underset{\sim}{\mathrm{R}}$ and R coincide.

Example 2.17. For any choice of $U \subseteq S = \mathbb{N}^k$ such that the pointwise preorder is U-bounded, the rank R is additive on multiples, and thus the three rank functions $\underset{\sim}{\mathrm{R}}$, $\mathrm{R}_{\mathrm{cat}}$ and R coincide. We give two examples to see what the rank functions look like.

Let $U = \{u_1, \ldots, u_k\}$ where $u_1 = (1, 0, \ldots, 0)$, $u_2 = (0, 1, \ldots, 0)$, etcetera, are the vectors of weight one. Then the pointwise preorder is U-bounded. For every $s \in S$, the rank is given by $\mathrm{R}(s) = \sum_i s_i$. This rank function is additive.

Let $U = \{u\}$ with $u = (1, \ldots, 1)$. Then the pointwise preorder is U-bounded. For every $s \in S$, the rank is given by $\mathrm{R}(s) = \max_i s_i$. Athough this rank is additive on multiples, it is not additive in the general sense. For example, for $k = 2$, we have $\mathrm{R}((1,1)) = 1$ while also $\mathrm{R}((0,1)) = 1$, $\mathrm{R}((1,0)) = 1$ and $(0,1) + (1,0) = (1,1)$.

Example 2.18. In this example, R and $\mathrm{R}_{\mathrm{cat}}$ do not coincide. Let $S = \mathbb{N}^3$ and let $U = \{u\}$ where $u = (1,0,0)$. Let P be the good preorder 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 2.19. In this example, $\underset{\sim}{\mathrm{R}}$ and $\mathrm{R}_{\mathrm{cat}}$ do not coincide. Let $S = \mathbb{N}^2$ and let $U = \{u\}$ where $u = (1,0)$. Let P be the good preorder 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 $\mu : S \to \mathbb{Q}_{\geq 0}$ is called an additive monotone if $\mu(a+b) = \mu(a) + \mu(b)$, $a \leq_b \implies \mu(a) \leq \mu(b)$ for all $a, b \in S$, and $\mu(u) \leq 1$ for every $u \in U$. Additive monotones have the following important property:

Lemma 2.20. *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 U-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 $\mu(s) \leq \underset{\sim}{\mathrm{R}}(s)$. ☐

General circuit preorder

We started this section by discussing concrete circuit models as preorders (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 preorder and rank functions. Here we will connect those two parts by discussing a general circuit preorder that clearly subsumes the concrete circuit models that have seen. The main point is to see that this general circuit preorder is a *good preorder* as defined in Definition 2.11.

Definition 2.21 (General circuit preorder)**.** As before, $\mathbf{1}$ denotes a formal symbol. A general circuit preorder is any preorder $\preceq$ on $S = \mathbb{N}^{F \cup \{\mathbf{1}\}}$ obtained as follows. Let

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

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, for any literal, we let $\{\ell\} \leq \{\mathbf{1}\}$. Third, we let $\preceq \supset \leq$ be the smallest ordering containing $\leq$ satisfying 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 $\mathbf{n} = \{\{\mathbf{1}, \dots, \mathbf{1}\}\}$ is the multiset in which $\mathbf{1}$ appears n times.

The general circuit preorder $\preceq$ 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* $n_i \in \mathbb{N}$, and with literals available at cost one.

Lemma 2.22. *The general circuit preorder $\preceq$ of Definition 2.21 is a good preorder.*

Proof. We need $\preceq$ to be U-bounded for some subset $U \subseteq F$, finitely generated, non-negative and a semigroup preorder. By definition, the preorder is U-bounded for $U = \{\mathbf{1}\}$, finitely generated by the explicitly given generators, non-negative (because of the explicitly imposed requirement that $A \subseteq B$ implies $A \preceq B$) and a semigroup preorder (because of the explicitly imposed $A \preceq B$ implies $A + C \preceq B + C$). □

One verifies directly that the boolean formula preorder $\preceq_{\mathrm{F}}$ (Definition 2.2), the branching program preorder $\preceq_{\mathrm{BP}}$ (Definition 2.3) and the comparator circuit preorder $\preceq_{\mathrm{CC}}$ (Definition 2.4) are special cases of the general circuit preorder and are thus *good* preorders by Lemma 2.22.

With respect to the general circuit preorder $\preceq$, the rank $\mathrm{R}(A)$ of $A \in S$ is by definition (Definition 2.15) the smallest number $n \in \mathbb{N}$ such that $A \preceq \mathbf{n} := \{\{\mathbf{1}, \dots, \mathbf{1}\}\}$. 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 2.15) given by $\underset{\sim}{\mathrm{R}}(f) = \lim_{m\to\infty} \mathrm{R}(m \cdot f)/m$. 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 2.15) 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 $\preceq$-monotone function $S \to \mathbb{R}_{\geq 0}$. These properties and previously discussed general considerations 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 $m \cdot f := \{\{f, \ldots, f\}\}$ is essentially the same as the rank of $\{f\}$, and we find that $\underset{\sim}{\mathrm{R}}(f) = \lim_{m\to\infty} \mathrm{R}(m \cdot f)/m = 0$ for every boolean function f in the boolean circuit model.

3. Duality

Strassen [Str88], motivated by the problem of designing fast matrix multiplication algorithms, introduced the theory of asymptotic spectra to study the amortized (or 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.

Our duality theorem will be phrased in terms of a collection of 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 preorder 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 *U-normalized* if for every $u \in U$ we have $\mu(u) \leq 1$. 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 3.1. *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 3.1 is that the additive monotones, asymptotic rank, catalytic rank and rank are related by $\max_{\mu \in M} \mu(f) = \underset{\sim}{\mathrm{R}}(f) \leq \mathrm{R}_{\mathrm{cat}}(f) \leq \mathrm{R}(f)$.

Lets take a moment to repeat the concrete meaning of Theorem 3.1. 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 3.1 applied to the branching program preorder $\preceq_{\mathrm{BP}}$ 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, $\max_\mu \mu(f) = \underset{\sim}{\mathsf{BP}}(f)$.

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 3.1 says that $\max_\mu \mu(f) = \underset{\sim}{\mathsf{CC}}(f)$ where the maximization goes over all submodular measures μ.

The amortized rank $\underset{\sim}{\mathrm{R}}$ being characterized as the pointwise maximum of the additive monotones μ by Theorem 3.1, 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. During our proof of Theorem 3.1, in which we phrase $\max_{\mu \in M} \mu(f)$ as a linear program, we obtain the following characterization of the catalytic rank.

Theorem 3.2. *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 3.2 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 3.1 as a special case of the following *preorder* version of the duality theorem (and this is where the preorder point of view on circuit models is indispensable):

Theorem 3.3. *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 $nkv + c \geq_P nku$.*

The meaning of Theorem 3.3 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.

3.1. Linear programming

The proof of Theorem 3.1 and Theorem 3.3 is an application of the well-known strong duality theorem for linear programming:

Theorem 3.4 (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*

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

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

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

Before going into the full proof of Theorem 3.1, 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 the highest 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 preorder $\preceq_{\mathrm{BP}}$.

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

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 preorder $\preceq_{\mathrm{BP}}$, that is, the inequalities

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

for all boolean functions $f, g \in F_n$, variables x_i, and literals ℓ. 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 $c_r = 1$ for any literal inequality and $c_r = 2$ for any branching inequality. 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:

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

In other words, the dual program optimizes over rational linear combinations of the generating inequalities of $\preceq_{\mathrm{BP}}$, minimizing the 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 $y \in \mathbb{Q}^R_{\geq 0}$ 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.

3.2. Full proof

We will prove in this section the duality theorem for the amortized rank (Theorem 3.1) and the duality theorem for amortized inequalities (Corollary 3.6).

The following theorem is the main technical duality theorem. Theorem 3.1 and Theorem 3.3 will be derived as a simple consequence afterwards.

Theorem 3.5. *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 $t \in S$ and an integer $k \geq 1$ such that

$$t + kv + kr \geq_P t + ku.$$

Proof. Let d be the number of generators of P. Let e be the cardinality of U. Let f be the cardinality of F. We will denote the generators of P by (v_1, w_1), ..., (v_d, w_d).

We will apply the strong duality theorem of linear programming (Theorem 3.4) with the following choice of matrix A and vector b. Let A be the $(d+e) \times f$ matrix with the first d rows given by the vectors $v_1 - w_1, \ldots, v_d - w_d$ and the next e rows given by the elements of U. Let b be the $d+e$ vector with the first d elements equal to 0 and the next e elements equal to 1.

Let $s = u - v \in \mathbb{Z}^F$. In order to prove the theorem, we will prove two claims, namely

$$\max\{s \cdot x \mid x \in \mathbb{Q}^F_{\geq 0},\, Ax \leq b\} = \max_{\mu \in M} \mu(u) - \mu(v) \tag{1}$$

and

$$\min\{b \cdot y \mid y \in \mathbb{Q}^{d+e}_{\geq 0},\, A^T y \geq s\} = r_{\min} \tag{2}$$

where we define $r_{\min}$ to 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

$$t + kv + kr \geq_P t + ku.$$

By Theorem 3.4 the left-hand side of (1) equals the left-hand side of (2). It follows that $\max_{\mu \in M} \mu(u) - \mu(v)$ equals $r_{\min}$, which proves the theorem.

To prove (1), it suffices to show that

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

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, and thus (1).

We will now prove (2). We first prove that

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

Let $y \in \mathbb{Q}^{d+e}_{\geq 0}$ satisfy $A^T y \geq s$. The inequaliy $A^T y \geq s$ says that

$$y_1(v_1 - w_1) + \cdots + y_d(v_d - w_d) + y_{d+1}u_1 + \cdots + y_{d+e}u_e \geq s = u - v.$$

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

$$ky_1(v_1 - w_1) + \cdots + ky_d(v_d - w_d) + ky_{d+1}u_1 + \cdots + ky_{d+e}u_e + kv \geq ku.$$

From this it follows using Lemma 2.13 that there exists an element $t \in S$ such that

$$ky_{d+1}u_1 + \cdots + ky_{d+e}u_e + kv + t \geq_P ku + t.$$

Thus $b \cdot y = y_{d+1} + \cdots + y_{d+e} \geq r_{\min}$. This proves one direction of (2).

We will now prove the other direction of (2), namely

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

Suppose that $kv + kr + t \geq_P ku + t$. It follows from Lemma 2.13 that there is an element $y \in \mathbb{Q}^{d+e}_{\geq 0}$ such that

$$y_1(v_1 - w_1) + \cdots + y_d(v_d - w_d) + y_{d+1}u_1 + \cdots + y_{d+e}u_e + v \geq u$$

where $b \cdot y = y_{d+1} + \cdots + y_{d+e} = r$. For this y it holds that $A^T y \geq s$. We conclude that (2) is true. □

Corollary 3.6. *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 $t + kv + kr \geq_P t + ku$*
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 $nkv + nkr + c \geq_P nku$.*

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 3.5 it follows that $t + kv + kr \geq_P t + ku$ for some $t \in S$ and some integer $k \geq 1$.

If there is an element $t \in S$ such that $t + kv + kr \geq_P t + ku$, then from Lemma 2.14 it follows that there is a constant $c \in \mathbb{N}$ such that for all $n \in \mathbb{N}$ we have $nkv + nkr + c \geq_P nku$.

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

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

Then, using the upper bound $r \geq \mu(r)$, 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 stating explicitely the special case of Corollary 3.6 where we set $r = 0$.

Theorem 3.3. *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 $nkv + c \geq_P nku$.*

Proof. The claim follows directly from Corollary 3.6 by setting $r = 0$. □

We also obtain the duality for the amortized rank as a corollary of Corollary 3.6.

Theorem 3.1. *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 3.6 with $v = 0$, to obtain that for all $n \in \mathbb{N}$ we have that $nr + o(n) \geq_P nu$. This implies that $r \geq \underset{\sim}{\mathrm{R}}(u)$. □

Remark 3.7. *Note that in our definition of M we required that for every $\mu \in M$ it holds that $0 \leq \mu(u) \leq 1$ for every $u \in U$. We do not require that $\mu(u) = 1$ for every $u \in U$. The latter would be to strict a requirement, since it would imply by Theorem 3.3 that for every $u_1, u_2 \in U$ there are an integer $k \geq 1$ and an integer $c \geq 0$ such that for all $n \in \mathbb{N}$ we have*

$$nku_1 + c \geq_P nku_2.$$

The following example gives a choice of S, P and U where this is false.

Example 3.8. Let $S = \mathbb{N}^2$ and let $U = \{u_1, u_2\}$ where $u_1 = (0, 1)$ and $u_2 = (1, 0)$. Let P be the pointwise preorder on S. Then P is a non-negative, finitely generated, U-bounded, semigroup preorder. For every $s = (s_1, s_2) \in S$, we have $\mathrm{R}(s) = s_1 + s_2$. Not only are u_1 and u_2 incomparable in P, also there cannot be integers $k \geq 1$ and $c \geq 0$ such that for all $n \in \mathbb{N}$ it holds that $nku_1 + c \geq_P nku_2$. In other words, u_1 and u_2 are incomparable even if we amortize P.

4. 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.

4.1. Symmetry and Formal Complexity Measures

Definition 4.1. 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 already for branching program complexity measures.

Lemma 4.2. *For any symmetric branching program complexity measure μ and any boolean function f,*

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

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

Proof. Let f be any boolean function on n boolean variables, and let T be the decision tree witnessing $D_{avg}(f)$. Assume that f depends on at least one variable; otherwise the upper bound is trivial. 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 can see that

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

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

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

Now, applying the inductive hypothesis and the definition of D_{avg}, we have

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

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 4.3. 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 a direct product 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.

Lemma 4.4. *Let $f : \{0,1\}^n \to \{0,1\}$ be a boolean function, let $i \in [n]$, and let $b \in \{0,1\}$. 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 c ofopies 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* $\mathsf{Orb}(f_{i\leftarrow 1})$, *where* $u_b = |\mathsf{Orb}(f)|/2|\mathsf{Orb}(f_{i\leftarrow b})|$.

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. 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 subgroup of $\Pi_{[n]}$ refines the orbit of f under $\Pi_{[n]}$. 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 first record some properties of $\simeq_b$.

- The relation $\simeq_b$ is an equivalence relation.
- If $S \subseteq U$ then since $i \notin S$, $g \simeq_b h$ if and only if $g^{\oplus S} \simeq_b h^{\oplus S}$.

Since $\simeq_b$ is an equivalence relation, we know that $\mathsf{Orb}_U(g)$ for any function g is partitioned by $\simeq_b$ into sets $\{S_1, S_2, \ldots, S_m\}$. Furthermore, since $\simeq_b$ is invariant under the action of Π_U, we have that $|S_i| = |S_j|$ for each i, j and furthermore Π_U acts on the collection of sets in the natural way, so $m = |\mathsf{Orb}(g_{i\leftarrow b})|$. In other words, $M(\mathsf{Orb}_U(g), i, b)$ can be partitioned into $u = |S_1| = |\mathsf{Orb}_U(g)|/|\mathsf{Orb}(g_{i\leftarrow b})|$ copies of the orbit $\mathsf{Orb}(g_{i\leftarrow b})$.

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 argument 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 implies that $\mathsf{Orb}(f_{i\leftarrow 0}) = \mathsf{Orb}(f_{i\leftarrow 1})$, as for any $g \in \mathsf{Orb}(f)$ there are sets $S, T \subseteq U$ such that $g = f^{\oplus S} = f^{\oplus S\cup\{i\}}$, and so $g_{i\leftarrow 1} = f^{\oplus S}_{i\leftarrow 1} = f^{\oplus T}_{i\leftarrow 0}$. 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 4.5. *For any boolean function f on n bits and any branching program complexity measure μ*

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

Proof. The proof is by induction on n, the number of variables on which f depends. If $n = 1$ then $f = x_1$, $D_{avg}(f) = 1$, and $\mathsf{Orb}(x_1) = \{x_1, \overline{x}_1\}$, so

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

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 have

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

where the last equality follows since we have substituted in 1 for x_n every g in $\mathsf{Orb}(f)$ (note that $g_1^{\oplus n} = g_0^{\oplus n}$). Therefore, by applying Lemma 4.4 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

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

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 4.6. *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 4.7. *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,

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

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 4.5 to each orbit and get

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

On the other hand, since H is closed under taking $\oplus$ with 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. □

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 4.8. *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 that 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. □

4.2. 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 4.9. 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 $r_1, r_2, \ldots, r_m$, and satisfies the following property. 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 4.10 (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 $2^{\ell+O(s)}$.*

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 4.11. 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.

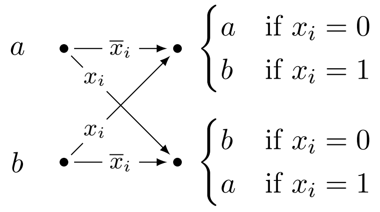

a
b
$\overline{x}_i$
x_i
x_i
$\overline{x}_i$
$\begin{cases} a & \text{if } x_i = 0 \\ b & \text{if } x_i = 1 \end{cases}$
$\begin{cases} b & \text{if } x_i = 0 \\ a & \text{if } x_i = 1 \end{cases}$

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

- *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 4.5 as a catalytic branching program. We note that since our duality theorem (Theorem 3.1) 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 4.12. *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 4.5, 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 $n = 1$ then $f = x_1$ or $\overline{x}_1$, $D_{avg}(f) = 1$, and $\mathsf{Orb}(x_1) = \{x_1, \overline{x}_1\}$. It is easy to see that $\mathsf{swap}(0, 1, i) = (\overline{x}_i, x_i)$, which is a branching program of size 4.

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 4.4.

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. 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 $(g, g^{\oplus n}) \in B$, and applying swap gates to each pair, we will obtain (by Lemma 4.4) 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 4.4, 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 $b \in \{0,1\}$.

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

This case is virtually identical to the previous case. In this case let $u_b = |\mathsf{Orb}(f)|/|\mathsf{Orb}(f_b)|$ 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 4.4. 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 4.7 to create a catalytic branching program computing f with much better complexity.

Theorem 4.13. *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 4.8, $\mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$ is closed under XORing with f and also closed under the action of $\Pi_{[n]}$. Also by Lemma 4.8, $1 \in \mathsf{span}_{\mathbb{F}_2}(\mathsf{Orb}(f))$, 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 4.12 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): one copy applied to $\{a_1, a_2, \ldots, a_C\}$ and one copy applied to $\{r_1, r_2, \ldots, r_C\}$. 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 4.12. 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. $\square$

5. 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 $n^{O(1)}$)?

Acknowledgements

The authors thank Aaron Potechin and Avi Wigderson for helpful conversations. 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

[AW20] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication, 2020.

[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.

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

[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 '14, page 857–866, 2014.

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

[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.

[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.

[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.

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

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

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

[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.

[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.

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

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

[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.

[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.

[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.

[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.

[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.

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

[Raz92] Alexander A. Razborov. On submodular complexity measures. In *Poceedings of the London Mathematical Society symposium on Boolean function complexity*, pages 76–83, 1992.

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

[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.

[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.

[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*, SFCS '86, pages 49–54, Washington, DC, USA, 1986. IEEE Computer Society.

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

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

[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.

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

[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.