ADVANCING ALGEBRAIC AND LOGICAL APPROACHES TO CIRCUIT LOWER BOUNDS

by

Matthew William Anderson

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2012

Date of final oral examination: 05/10/2012

The dissertation is approved by the following members of the Final Oral Committee:

Dieter van Melkebeek, Associate Professor, Computer Sciences
C. Eric Bach, Professor, Computer Sciences
Jin-Yi Cai, Professor, Computer Sciences
Shuchi Chawla, Assistant Professor, Computer Sciences
Donald Passman, Professor, Mathematics
Lance Fortnow, Professor, Computer Science, Northwestern University

For Mom and Dad.

Acknowledgments

Out of all the parts of this dissertation this section – the Acknowledgments – was the easiest to write as I am, without doubt, grateful to the hundreds if not thousands of people that made this dissertation possible. Please bear with me as I take a moment to thank a few of those people (and excuse any inadvertent omissions).

First and foremost, I'd like to thank my advisor, Dieter van Melkebeek. Dieter, thank you for your tireless effort first as a teacher, where your exuberance for the material led you to provide many *bonus* hours of instruction. Then, later as an encouraging advisor where you patiently helped mold me into the researcher I am today. Thank you for your selfless dedication and the uncountably many hours you spent reading through and helping to improve the – oft poorly written – drafts of my work, including this one. You've been a staunch advocate and a principled role model. If someday I have the privilege of advising students of my own I hope that I am half the teacher, advisor, advocate, and friend that you have been (I hope, however, to have better handwriting).

To my co-authors: Dieter, Nicole Schweikardt, Luc Segoufin, and Ilya Volkovich. Without your help the research in this dissertation would not have been possible, and would certainly have been less enjoyable. I look forward to working with you all in the future.

To my defense committee (a superset of my prelim committee): Eric Bach, Jin-Yi Cai, Shuchi Chawla, Lance Fortnow, Dieter, and Donald Passman. Thank you for the time you've taken out of your busy schedules to serve on my committee, and to many of you for the excellent courses you taught that laid the foundations of my understanding of our field. I would also like to thank you for your patient and careful reading of this document.

To my friends and colleagues among the grad students at Madison, you made it fun to be here. Jeff, thanks for being someone to look up to, listening to all my crazy ideas and for contributing some of your own, and for all those practical jokes. Siddharth, your enthusiasm

for research will always be an ideal I aspire to, though maybe I just need to drink coffee, and I'll miss your constant visits to my office that have been a welcome distraction (including as I was writing this very line). David, thanks for all the discussions and support on the programming contest; it was nice to be able to bounce ideas off you. Tyson, I'll miss our periodic debates about irrelevant things. Thanks for always courteously letting me win, and for getting us all home safely through that massive blizzard on Midwest Theory Day 2010. William, I couldn't think of anything really clever to say, but thanks all the same. Dalibor, thanks for taking care of the refrigerator. Scott, thanks for putting up with me as an office-mate and graciously being on the wrong end of some of our practical jokes. I'd like to thank everyone, including those I didn't have a chance to mention by name, for their friendship, camaraderie, and support. I'm going to miss the theory students: our summer lunches sitting out on the terrace, our irregularly scheduled reading group, our prolonged, but constructive, practice talks, our adventures to theory day, our grossly disproportionate "Meet the Theory Students" events, and our now-defunct point system.

To my parents, Dennis and Lucy, and my sisters, Gaby and Sarah, thank you for your unwavering support, prayers, and love: I'm a "scientide" now! Thanks to my extended family in Missouri who have made our semi-annual vacations enjoyable and revitalizing.

I'd like to thank the numerous professors and instructors who supervised my teaching for teaching me how to teach: Eric, Paul Barford, Deb Deppler, Kunal Ghosh, Beck Hasti, Tom Reps, and Jim Skrentny; and the many students I interacted with during my studies at Wisconsin for their delightful inquisitiveness.

This dissertation would not have been possible without the many excellent teachers that I had in the West Windsor-Plainsboro School District, at Carnegie Mellon University, and at the University of Wisconsin-Madison. To Bob Griffiths and Avrim Blum, thank you for advising my undergraduate senior thesis, my first foray into research. To Mrs. Gray, your AP English course is still the hardest class I've ever taken, thank you for teaching me how to

write. To Ms. Coppock, thank you for pointing out that I don't know everything; I still don't, but knowing that has been useful in life (and in grad school). To Mrs. Schweiger, thank you for making math exciting and starting me off on the path towards this dissertation.

I appreciate the generous financial support provided in part by the University of Wisconsin-Madison, Cisco Systems through the Distinguished Graduate Fellowship, and the National Science Foundation (grants 0523680, 0728809, and 1017597). I would also like to thank the ACM Special Interest Group on Algorithm and Computation Theory (SIGACT) and the Vilas Foundation for their grants to support conference travel.

Finally, I would like to thank L. Frank Baum for his whimsical novel *The Wonderful Wizard of Oz* [Bau00].

Thanks everyone!

Contents

Abstract

Complexity theory attempts to classify problems into classes according to their resource requirements such as time and space, and to understand the relative power of these resources. This dissertation is motivated, in particular, by the study of the limitations of Boolean and arithmetic circuits as models of computation. The goal of this area is to show that there are explicit problems that cannot be solved by small circuits. To this end we make progress along several avenues for proving such circuit lower bounds.

Polynomial Identity Testing is the fundamental problem of testing whether a given multivariate polynomial is identically zero. There is a natural efficient randomized algorithm. Showing that there is an efficient *deterministic* identity test, in a sufficiently general setting, implies long elusive circuit lower bounds. In the first part of this dissertation, we develop new deterministic identity tests for bounded-read multilinear arithmetic formulas, an interesting class of polynomials that encompasses and significantly generalizes several prior works.

Locality is a property of logical formulas that expresses that the formula cannot distinguish between two inputs that appear the same up to some distance parameter. Once established for a given set of logical formulas, the property can often be used to quickly argue separations from that set. In this dissertation we tightly characterize the locality of a logical system corresponding to AC^0, the class of languages solvable by families of constant-depth polynomial-size Boolean circuits. In doing so we give a meta-theorem for proving that certain graph properties cannot be computed in AC^0.

Kernelization is the process of transforming an equivalent instance of one problem into an instance of another problem where the size of the latter instance depends only on a single parameter. We look at a set system that is motivated by the study of kernelization, and give a tight bound on the size of that set system.

1 Introduction

The only way to discover the limits of the possible is to go beyond them into the impossible.

— Arthur C. Clarke

We begin our story in the middle. We join Dorothy, an orphan from Kansas; the Scarecrow, a strawman without a brain; and the Tinman, a woodcutter without a heart, as they travel the yellow-brick road towards a fateful meeting with the Wonderful Wizard of Lower Boundz.

A Quest for Identity

The road leads into a dark and foreboding forest. Not far within, Dorothy, the Scarecrow, and the Tinman begin to hear a muffled sobbing through the trees. As they push on, the noise becomes louder and louder until it is almost a roar. Turning a bend in the road they see a huge lion curled up in a ball next to a tree and crying relentlessly. "Are you OK?" Dorothy asks timidly of the cowering lion. In a start, the lion spins and darts behind a nearby tree "WHO... Who are you? Leave me alone!" "Please don't be afraid", Dorothy responds more confidently. "Can we help you?" "No, I'm afraid of everyone, and everything; no one can help," he whimpers.

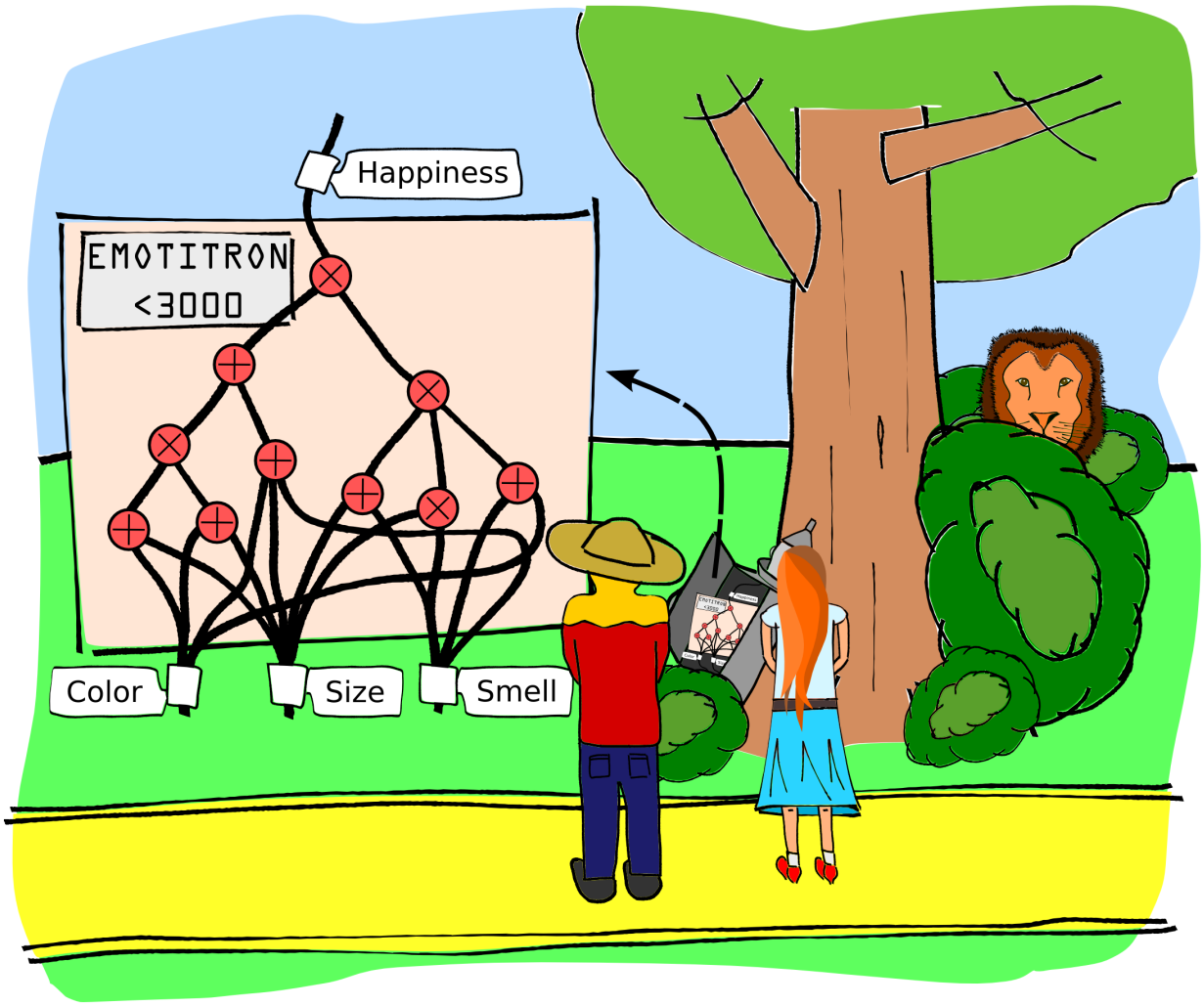
"Let's keep moving," the Tinman presses impatiently. "How can you be so insensitive?" Dorothy questions him. The Tinman, taken aback, blinks twice, "I... am... broken." He continues mechanically, "I feel nothing for this creature who is clearly in distress." For some time, Dorothy and the Scarecrow attempt to comfort the now stricken Tinman and the Lion with no success.

Drawing away from the others Dorothy and the Scarecrow confer. “We must help them.” “The Tinman feels nothing and the Lion fears everything.” “We need to find something the Tinman cares for and something the Lion isn’t afraid of.”

The two return to the group. “We may as well begin with you Tinman.” Dorothy asserts, and then asks him “What do you feel about the color of the sunrise?” “It is a consequence of atmospheric refraction,” he states matter-of-factly. “And what is your favorite kind of pie?” she follows up. “My favorite pies are apple, cherry, key lime, ...,” the list continues for some time before the Scarecrow stops him, “But what is your favorite?” The Tinman thinks for a moment, “I have no favorite.” This continues well into the night.

The next morning, Dorothy observes, “We need more information about how the Tinman forms emotions. Tinman, may we look at the circuitry inside your chest?” “Go ahead,” he replies, “I’m sure I won’t feel a thing.” With the Scarecrow’s help Dorothy opens the Tinman’s metallic torso. Inside, buried in a tangle of wires they find a circuit board reading EMOTITRON-<3000. “This must be it!... Your emotions are determined by an arithmetic formula!” Dorothy exclaims. “What’s an ‘a-rythm-o-matic form mule’?” the Scarecrow asks, having forgotten grade-school algebra. “Well,” Dorothy begins, “it takes in a set of constants and variables, which in this case are properties like ‘Color’, ‘Smell’, and ‘Size’, and combines them arithmetically, using $+$ and $\times$ operations.” “The magnitude of the resulting value expresses your feelings on the matter,” she concludes pointing at a single wire in the Tinman’s chest cavity labeled ‘Happiness’. “If we can demonstrate a point where your happiness formula is non-zero, it will prove you have a heart, Tinman.” The Tinman’s face remains neutral.

“Hurry up,” the Lion moans, “trying all possible settings will take too long.” “Suppose we only had one property, say ‘Color’?” Dorothy followed, “If I recall, a non-zero formula of size s can’t have more than s zeroes.” (Dorothy goes to an excellent public magnet school in Kansas.) “This means we would only have to try $s + 1$ distinct colors before we find a



color the Tinman cares about."

"It shouldn't be much harder with more than one property." She pauses, then argues inductively, "Suppose we have picked good values for all the properties except one. Then, the resulting formula only depends on one property and has smaller size. Assuming the partial assignment so far is good (that is, we haven't zeroed the formula), picking a random element from a set of size $100s$ should succeed in distinguishing the formula from zero." "This means that if we pick a value for each property randomly from a set of $100s$ potential values we will find something the Tinman cares about 99% of the time," she concludes. "RANDOMNESS!!!" the Lion shrieks and bolts out of earshot down the road.

Undeterred, the companions turn back to the Tinman to try out their observation. “Huge green neutral squishy pizza.” “Nothing.” “Large yellow menacing solid flower.” “Nothing,” he repeats. “Small white harmless fluffy bunny.” The Tinman blinks, then smiles “...Yes.” “It worked!” Dorothy and the Scarecrow exclaim together. The Tinman, recovered, asks “What’s the Lion’s problem?” “I guess he’s afraid of randomness,” posits the Scarecrow. The three set off to find the missing Lion.

As they search they ponder how to solve the Lion’s problem. “We can’t randomly guess something he isn’t afraid of, he’s too scared of randomness to let us do that.” “It’s even more of a problem that we can’t see how he determines what he’s afraid of. We can’t open up his head and physically examine his amygdala,” the Tinman reasons. “We can only test his response to certain questions.” The three walk quietly for a while. “Let’s assume that his fear is also determined by a formula,” offers the Tinman. “Since we don’t know which particular formula models the Lion’s fear we should come up with a set of inputs that work for all formulas.” Dorothy adds, “Certainly a small random set would work...” “We need to generate this set quickly, so it must be small, and we must do so without using randomness,” the Tinman counters. “But HOW?...”

Unbeknownst to them, the Wicked Witches of Algebrization, Natural Proofs, and Relativization gather around an opalescent crystal ball watching our heroes from afar. The youngest guffaws “Foolish little girl, no one knows whether such a small hitting set can be constructed efficiently.” The middle sister scoffs “You don’t even know how to do it for depth three formulas!” The eldest, Relativization, chides the other two, “Sisters, we must be careful. If they are successful in finding an efficiently computable hitting set we will be defeated. They will reach the Emerald City and be granted an audience with the Wizard of Lower Boundz.”

Meanwhile, the companions continue to search for the Lion. “This problem makes my head hurt,” the Scarecrow mutters to no one. Suddenly, the soft voice of Glenda, the Good

Witch of Prior Work, permeates the dark forest "Fear not. The Lion's fear formula is a constant-read multilinear formula." Her voice slowly fades. After a day or two – which by all accounts seemed like years – the trio devise a test that can help the Lion. But first they need to find him...

Locally Lost

The three continue along the yellow-brick road searching for the Lion. The dark forest gradually transitions into an enchanting meadow. Waist-tall grass grows as far as they can see in all directions. Lone boulders of uniform size, shape, and color appear in the distance away from the road. The meadow is uncannily flat with no discernible hills or valleys.

On the morning of their fifth day in the meadow they come to a fork in the yellow-brick road. Here the road splits off in three directions. At the left side of each road there is a sign reading *Lower Boundz* pointing away from the intersection. Curiously, there is a fourth sign pointing back along the road the companions followed here. The trio examine each of the four identical roads and signs. "Even the landscape in the distance looks the same," Dorothy observes. "Which way do we go?" "Not the way we came," says the Tinman shattering the sign pointing back the way they came with his ax.

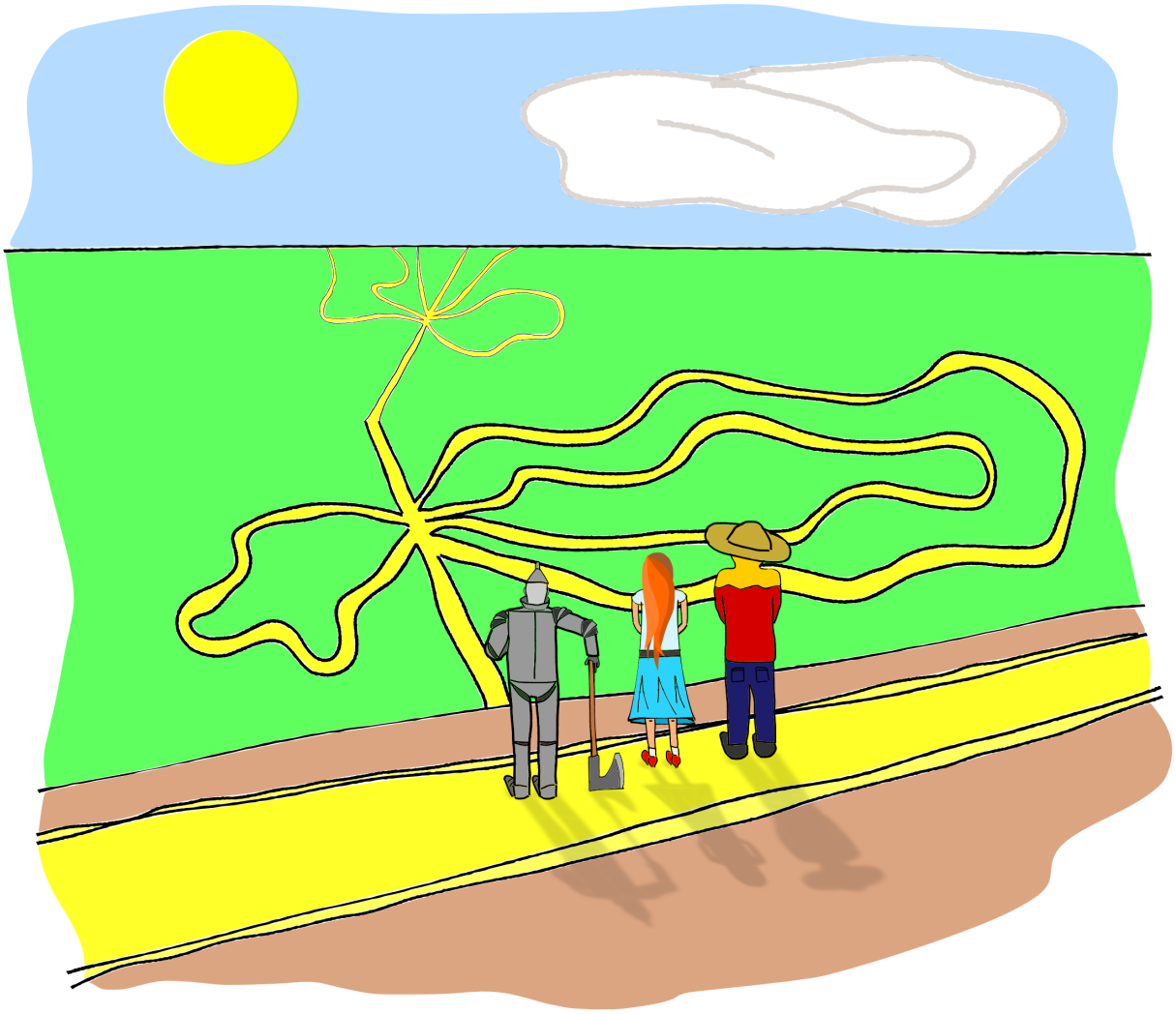
"We may as well go straight," says Dorothy after some time. Dorothy and the Tinman begin to walk down that road. The Scarecrow momentarily hesitates, and then follows. They walk for several days along the road, never passing anything of note. In the afternoon of the third day they again reach a crossroads that splits off in three directions. "I think we've been here before," the Scarecrow points at pieces of a sign that lay smashed by the left side of one of the forks. "We've been walking in a circle..." they sigh in unison. "At least we've eliminated some possibilities," says the Tinman hopefully as he knocks down two wrong signs. The three proceed away from the crossroads following the one remaining

sign.

The next day they reach another crossroads, here the road forks into seven parts each paired with now unconvincing signs. They choose one to follow, but the Scarecrow again hesitates, and, sure enough, the path loops back to the same crossroads. After several more false starts they reach a new crossroads. Exasperated, the companions sit down by the side of the road. "If history is any lesson, most of these roads loop back here, one goes back where we came," Dorothy gestures to the sign the Tinman had already destroyed, "and one actually goes towards the wizard." "How can we decide which way to go?" Squinting off into the distance, the Tinman points out, "Locally, everything about these roads appears exactly the same; we have no global information about the road network." Dorothy and the Tinman begin to irritably argue about their predicament.

After some time, the Scarecrow silently starts off down one of the roads. Eventually, during a lull in the debate Dorothy and the Tinman notice that the Scarecrow is gone and run to catch up. "Where are you going?" they say breathlessly when they reach him. He stops and turns towards them. "You've hardly said a word since we reached the first crossroads," Dorothy continues, "is there something wrong?" The Scarecrow looks down, and mumbles, "I... At each crossroads I felt which direction we should go... I don't know how... I've been right both times... But I'm not very smart... I didn't want to..." He trails off. "We may as well go your way, I don't have any better ideas," the Tinman callously concludes the discussion.

They reach a new crossroad the next day, apparently making progress. They follow the now more confident Scarecrow out of the crossroads. This repeats several times, never looping back to where they've been before. After about a week the yellow-brick road leads treacherously up a mile-tall escarpment. They pause after the long climb and look back at their path through the meadow. They see the yellow road reflect brightly in the afternoon sun, along with the long path they followed and the many spindled crossroads looping back



on themselves. "I don't know how you did it, but you got us through that strawman," the Tinman apologizes, "you could tell which path was a cycle."

"You're pretty intelligent after all Scarecrow; you're smarter than an AC^0 circuit." The strawman blushes. "And, indubitably, street-smart," Dorothy adds, and they all laugh.

Following the Yellow-Brick Road

Much like the preceding tales, we begin our story in the middle of a long intellectual journey towards a central and elusive goal in complexity theory:

Exhibit explicit problems that cannot be solved by small circuits.

In this dissertation we deal with two types of circuits: Boolean and arithmetic. Each circuit is structured by an underlying directed acyclic graph whose leaf nodes correspond to inputs that are either Boolean variables, or arithmetic variables and constant field elements. The other nodes of the graph represent *gates*. In the Boolean setting these are usually AND, OR and NOT, and in the arithmetic case + and ×. Each gate naturally computes a Boolean function or a polynomial, respectively, of its children. The root of the graph represents the output or value computed by the circuit.

The *circuit complexity* of a problem is the number of gates and wires a circuit requires to compute a given function as a function of the number of input variables. By comparing the number of functions on n bits (2^{2^n}) with the number of size-s circuits computing n-bit functions ($2^{O(s \log s)}$) it follows that there exists some language that requires exponential-size circuits. However, this argument only tells us that some such language exists, but does not show us how to *explicitly* (succinctly) describe it. It is widely believed that a stronger property holds:

Conjecture 1. *There exist* explicit *problems that no small circuit family can compute.*

One instance of an explicit language that we believe is hard to compute is Boolean circuit satisfiability (SAT), that is, the problem of determining whether a given Boolean circuit has a satisfying assignment. Since this problem is complete for non-deterministic polynomial time, NP, showing that it requires large circuits would immediately imply that NP $\neq$ P, answering a monumental open problem in theoretical computer science. On the other hand, if NP does have polynomial-size circuits it would collapse the entire polynomial hierarchy, PH, to the second level [KL82].

From the algebraic perspective it is believed that the Permanent polynomial is exponentially hard, i.e., that it requires exponential-size arithmetic circuits to compute. Intuitively,

the Permanent polynomial is an unsigned version of the Determinant:

$$\mathrm{Perm}(M) \doteq \sum_{\sigma \in \mathrm{Sym}_n} \prod_i M_{i,\sigma(i)},$$

where M is an n-by-n symbolic matrix and Sym_n is the symmetric group on n elements. The Permanent is VNP-complete, that is, it plays the analogous role to SAT for counting problems [Val79]. Moreover, the entire polynomial hierarchy reduces to the Permanent [Tod91].

Showing general and unconditional circuit lower bounds for either of these canonical problems has proved notoriously difficult; and, for this reason, much research has focused on lower bounds for restricted circuit models. We mention a few such results now.

For Boolean circuits, no super-linear lower bounds are known for problems in NP in the most general setting. There has been considerable progress for classes related to constant-depth circuits. Perhaps the most well-known is a deep result from the 80s which shows that PARITY, the problem of determining whether the number of ones in a sequence of n bits is odd, cannot be computed, even approximately, by constant-depth Boolean circuits of sub-exponential size [Ajt83, FSS84, Yao85, Cai89, Hås86]. Recently, Williams showed that there are problems in NEXP, non-deterministic exponential time, that do not have polynomial-size constant-depth Boolean circuits with modulo gates, ACC^0 [Wil11]. There are also a number of results that show that classes within and related to the polynomial hierarchy do not have size n^c circuits for any constant c (e.g., [Kan82, Cai01]).

For arithmetic circuits, Baur and Strassen showed that the polynomial $\sum_{i=1}^{n} x_i^r$ requires $\Omega(n \log r)$ size on general arithmetic circuits [BS83]. Raz demonstrated that both the Determinant and Permanent polynomials of dimension n require multilinear formulas of size $2^{\Omega(\log^2 n)}$ [Raz09]. A formula is *multilinear* if every gate computes a polynomial that is linear in each variable.

These results, and others we did not mention, are quite far from the full generality of Conjecture 1. A number of barriers have arisen along the path to this goal, that, perhaps, explain the difficulty: relativization [BGS75], natural proofs [RR97], and algebrization [AW09]. Informally, these barriers are a manifestation of the fact that arguing the conjecture will require more advanced proof techniques (and understanding) than we currently have.

In the following paragraphs we give a brief description of our main results. The remaining sections of this chapter provide further context and motivation, and the body of this dissertation formally substantiates the claims made in our theorems.

Polynomial Identity Testing is the fundamental problem of testing whether a given multivariate polynomial is identically zero. There is an efficient natural randomized algorithm, as noted by Dorothy and the others in the first episode. Identity testing is closely tied to arithmetic circuit lower bounds. In particular, an efficient deterministic identity test, in a sufficiently general setting, implies strong circuit lower bounds. In the first part of this dissertation, Section 1.1 and Chapter 3, we discuss new deterministic identity tests for multilinear constant-read formulas, that is, formulas where each variable may occur only a constant number of times. We give tests both in the blackbox and non-blackbox setting for this type of formula, where blackbox means that the test does not have access to the structure of the formula. The blackbox setting is much like the situation with the Lion's fear polynomial in the first episode, as opposed to the non-blackbox setting which is similar to how the Tinman's formula was visible to Dorothy and the Scarecrow. In addition, we can extend our tests to a class of polynomials that significantly generalize and encompass several prior works. This is joint work with Dieter van Melkebeek and Ilya Volkovich and first appeared at Conference on Computational Complexity (CCC'11) [AvMV11].

Locality is a property of logical formulas that expresses that the formula cannot distinguish between two inputs that appear the same up to some distance parameter; this is similar

to the difficulty the companions had in the second episode while trying to distinguish the correct road to follow among all the roads that appeared locally the same. Locality can be used to quickly argue separations between sets of logical formulas. In Section 1.2 and Chapter 4 we tightly characterize the locality of a logical system corresponding to AC^0, the class of languages solvable by families of constant-depth polynomial-size Boolean circuits, and in doing so give a meta-theorem for proving that certain graph properties cannot be computed in AC^0. In the second story the Scarecrow was able to sense a variant of cycle checking which is a very non-local property, and hence the others noted he was more powerful than AC^0. Our proof hinges on the strong lower bound for PARITY mentioned earlier. Locality may prove a useful tool for separating some of the circuit classes between AC^0 and L (deterministic logspace) by tightly bounding the locality of the circuit classes in this vicinity. This is joint work with Dieter van Melkebeek, Nicole Schweikardt, and Luc Segoufin, and first appeared at the International Colloquium on Automata, Languages and Programming (ICALP'11) [AvMSS11].

Kernelization is the process of transforming an instance of one problem into an equivalent instance of another problem where the size of the latter instance depends only on a single parameter. In Section 1.3 and Chapter 5 we look at a problem that was motivated by the study of kernelization. Our result is a tight bound on the size of a set system that arose in the development of [DvM10]. This result is yet unpublished.

1.1 Deterministic Polynomial Identity Testing

Polynomial identity testing (PIT) denotes the elementary problem of deciding whether a given polynomial identity holds. More precisely, we are given an arithmetic circuit or formula F on n inputs over a given field $\mathbb{F}$, and we wish to know whether all the coefficients of the formal polynomial P, computed by F, vanish. Due to its basic nature, PIT shows

up in many constructions in theory of computing. Particular problems that reduce to PIT include integer primality testing [AB03] and finding perfect matchings in graphs [Lov79].

PIT has a very natural randomized algorithm – pick values for the variables uniformly at random from a small set S, and accept iff P evaluates to zero on that input. If $P \equiv 0$ then the algorithm never errs; if $P \not\equiv 0$ then by Schwartz-Zippel [Sch80, Zip79, DL78] the probability of error is at most $d/|S|$, where d denotes the total degree of P. This results in an efficient randomized algorithm for PIT. The algorithm works in a blackbox fashion in the sense that it does not access the representation of the polynomial P, rather it only examines the value of P at certain points (from $\mathbb{F}$ or an extension field of $\mathbb{F}$).

Despite the simplicity of the above randomized algorithm and much work over the course of thirty years no efficient deterministic algorithm is known when the polynomial is given as an arithmetic formula.

Is there an efficient deterministic identity test for arithmetic formulas?

This question is fundamental to our understanding of circuit lower bounds: Efficiently derandomizing identity testing implies elusive circuit/formula lower bounds [KI04, KvMS09, AvM10]. In fact, an efficient deterministic identity test for depth-four arithmetic formulas implies a lower bound for general arithmetic circuits [AV08]. There are also a number of partial converses that show that lower bounds can imply deterministic identity tests [KI04, DSY08].

The powerful connections with circuit lower bounds have energized much recent effort towards derandomizing identity testing for restricted classes of arithmetic formulas, in particular for constant-depth formulas. For depth two formulas several deterministic polynomial-time blackbox algorithms are known [BOT88, KS01, Agr03, AM10, BHLV09]. For depth three the state-of-the-art is a deterministic polynomial-time blackbox algorithm when the fanin of the top gate is fixed to any constant [SS11]. The same is known for depth

four but only when the formulas are multilinear [SV11]. We refer to the excellent survey papers [Sax09, SY10] for more information.

Another natural restriction are arithmetic formulas in which each variable appears only a limited number of times. We call such formulas *read-k*, where k denotes maximum number of times each variable may appear. PIT for read-once formulas is trivial in the non-blackbox setting as there can be no cancellation of monomials. Shpilka and Volkovich considered a special type of bounded-read formulas, namely formulas that are the sum of k read-once formulas. For such formulas and constant k they established a deterministic polynomial-time non-blackbox algorithm as well as a deterministic blackbox algorithm that runs in quasi-polynomial time, i.e., in time $s^{O(\log s)}$ on formulas of size s [SV08, SV09].

Results

We present a deterministic polynomial-time identity test for multilinear constant-read formulas, as well as a deterministic quasi-polynomial-time blackbox algorithm for these formulas.[1]

Theorem 1. *For each $k \in \mathbb{N}$ there is a deterministic polynomial identity test for multilinear read-k formulas of size s that runs in time* $\mathrm{poly}(s)$*. In addition, there is a deterministic blackbox test that runs in time $s^{O(\log s)}$.*

Note that Theorem 1 extends the class of formulas that Shpilka and Volkovich could handle since a sum of read-once formulas is always multilinear. This is a *strict* extension – in Section 3.3.1.3 we exhibit an explicit read-2 formula with n variables that requires $\Omega(n)$ terms when written as a sum of read-once formulas. The separating example also shows that the efficiency of the identity test in Theorem 1 cannot be obtained by first expressing

[1]Note that the complete formal statements of results discussed in this section can be found in Section 3.1.

the given formula as a sum of read-once formulas and then applying the known algorithms [SV09] for sums of read-once formulas to it.

Shpilka and Volkovich actually proved their result for sums of a somewhat more general type than read-once formulas, namely read-once formulas in which each leaf variable is replaced by a low-degree univariate polynomial in that variable. We can handle a further extension in which the leaf variables are replaced by *sparse multivariate* polynomials. We use the term *sparse-substituted* formula for a formula along with substitutions for the leaf variables by multivariate polynomials that are each given as a list of terms (monomials). We call a sparse-substituted formula *read-k* if each variable appears in at most k of those multivariate polynomials.

We can even further extend our identity tests by introducing a relaxed notion of multilinearity for sparse-substituted formulas that requires only that for every multiplication gate of the original formula the different input branches of the gate are variable disjoint. We call such sparse-substituted formulas *structurally-multilinear*. Note that this definition allows the resulting substituted polynomials to be non-multilinear.

Theorem 2. *For each $k \in \mathbb{N}$ there is a deterministic polynomial identity test for structurally-multilinear sparse-substituted read-k formulas that runs in time $s^{O(\log t)}$, where s denotes the size of the formula, and t the maximum number of terms a substitution consists of. In addition, there is a deterministic blackbox test that runs in time $s^{O(\log st)}$.*

We observe that any multilinear depth-four alternating formula with an addition gate of fanin k as the output can be written as the sum of k sparse-substituted read-once formulas, where the read-once formulas are single monomials and the substitutions correspond to multilinear depth-two formulas. This implies that our blackbox algorithm also extends the work by Karnin et al. [KMSV10], who established a deterministic quasi-polynomial-time blackbox algorithm for multilinear formulas of depth four. Thus, our results can be seen

as unifying identity tests for sums of read-once formulas [SV09] with identity tests for depth-four multilinear formulas [KMSV10] while achieving comparable running times in each of those restricted settings.

We can improve the running time of our blackbox algorithm in the case where the formulas have small depth. In particular, we obtain a polynomial-time blackbox algorithm for constant-read constant-depth formulas.

1.2 Locality

Expressibility of logics over finite structures plays an important role in various areas of computer science. In *descriptive complexity*, logics over finite structures are used to characterize complexity classes [Imm99]. For example, existential second-order logic can describe exactly those graph problems that belong to the complexity class NP. In *automated verification*, one uses logics as specification languages to describe properties of hardware and software systems, and one needs to balance the expressivity of the logics used with the feasibility of the model checking task (cf., e.g., [CGP99]).

The classical inexpressibility arguments for logics over finite structures (i.e., Ehrenfeucht-Fraïssé games) often involve nontrivial combinatorics. The notion of *locality* has been proposed as an alternative that allows one to contain much of the hard work in generic results, and keep the specific applications simple. Roughly speaking, a query is local if one only needs to look at a small, localized part of the structure in order to answer the query. If one can show that every query in a given logic has a certain degree of locality, and the query at hand does not, then one can conclude that the query is not expressible in the logic. For example, one can show that for every first-order query on graphs, there exists a constant r such that the result of the query only depends on the neighborhood up to distance r of the vertices that are part of the query (cf., e.g., the textbook [Lib04]). On the other hand, it is

easy to see that the connectivity of two vertices in a graph is not determined solely by the restriction to those neighborhoods. Therefore, connectivity is not expressible in first-order logic.

Apart from inexpressibility proofs, locality is also used as a tool for obtaining algorithmic meta theorems, i.e., results stating that if a problem is expressible in a certain logic on a certain class of structures, then it can be solved algorithmically within certain resource bounds (c.f., [GK11] for a recent overview on this topic).

This motivates the following question:

To what extent are logics local?

We show how to use *circuit lower bounds* to establish *upper bounds on the locality radius* of certain logics. In particular, we consider a logic that corresponds to the complexity class AC^0 of all languages that can be decided by nonuniform families of polynomial-size constant-depth circuits. By exploiting the known lower bounds for parity and related problems on constant-depth circuits [Ajt83, FSS84, Yao85, Cai89, Hås86], we obtain an upper bound for the locality radius of queries expressible in that logic. This provides a simple and generic means of proving inexpressibility results for this type of formulas, hence extending the power of the original lower bound to a larger class of queries. It also gives a simple and general way of showing that many graph queries cannot be computed in AC^0. We also give examples showing that our upper bounds are essentially tight.

The logic we consider is the extension of order-invariant first-order logic with arbitrary numerical predicates. The notion of *order-invariance* was introduced a while ago to capture the data independence principle in databases, cf., [AHV95, Lib04]: An implementation of a database query may exploit the order in which the elements are stored on a disk, but only in such a way that the result of the query does not depend on this order. Order-invariant first-order queries are exactly those first-order queries that can make use of an order predicate $<$

but such that the answer is independent of the interpretation of $<$ as long as it is a linear order on the domain of the structure. In our extension, on top of the order predicate $<$, we also allow the use of *arbitrary numerical predicates* that are induced by the order. This means that the formula has access to numerical predicates, like addition, multiplication, and powering, all defined with respect to the order $<$, e.g., if a, b, and c are the 1^{st}, 2^{nd}, and 3^{rd} elements of the universe with respect to $<$, then $a + b = c$, $a \cdot b = b$, and $a^c = a$. Further, we require that the result of a query not depend on the actual choice of the linear order when all numerical predicates are interpreted consistent with the linear order. We denote this logic[2] as *Arb-invariant* FO. In terms of graph queries, Arb-invariant FO expresses precisely those computable in the complexity class AC^0.

Results

In order to state our results, we need to introduce the two standard notions of locality, known as *Gaifman locality* and *Hanf locality*. Both are based on the distance measure on the elements of a structure when viewed as the vertices of a graph in which two elements are connected by an edge whenever they appear together in a tuple of one of the structure's relations. The latter graph is referred to as the *Gaifman graph* of the structure. In a nutshell, Gaifman locality means that a query cannot distinguish between two tuples having the same neighborhood type in a given structure, while Hanf locality means that a query cannot distinguish between two structures having the same (multi-)set of neighborhood types. Here, the neighborhood type of a tuple refers to the equivalence class under isomorphism of the substructure induced by the elements up to distance r from the tuple, where r is a parameter. It is known that Hanf locality implies Gaifman locality, modulo a constant factor loss in the distance parameter r. We refer to Section 4.3 for the formal treatment of

[2]Strictly speaking, Arb-invariant FO is a "logical system" rather than a "logic", as the syntax is undecidable.

these notions.

A well-known result shows that first-order logic exhibits constant locality with respect to both notions, i.e., every FO query is Gaifman and Hanf local with a constant parameter r depending on the query [Gai82, Han65]. In the presence of an extra linear order that is part of the structure, all neighborhoods of positive radius degenerate to the entire domain, so all queries are trivially 1-local. Locality becomes meaningful again in *order-invariant* FO, where the formulas can make use of an order, but the structure does not contain the order and the semantics are independent of the order. It is shown in [GS00] that order-invariant FO queries are Gaifman local with a constant parameter r depending on the query. The status of Hanf locality for order-invariant FO is still open in general; it is only known for structures like strings and trees [BS09].

When we allow arbitrary numerical predicates, constant locality no longer holds, even if we require Arb-invariance. In fact, we show that the level of Gaifman locality of Arb-invariant FO queries can be polylogarithmic in the number of elements of the structure, but no worse than that: Arb-invariant FO is Gaifman $(\log n)^{O(1)}$-local in the following sense.

Theorem 3. *Every Arb-invariant* FO *formula is Gaifman* $(\log n)^c$*-local for some constant* c *depending on the formula, and for every constant* c *there exists an Arb-invariant* FO *formula that is not Gaifman* $(\log n)^c$*-local.*

The upper bound in Theorem 3 means that for any query in Arb-invariant FO and any large enough number n, if a structure has n elements and if two tuples of that structure have the same neighborhood up to distance $(\log n)^c$, then they cannot be distinguished by the query. The lower bound part of Theorem 3 is realized by variations of the connectivity example mentioned before.

As easy consequences of the upper bound in Theorem 3 one obtains, e.g., that the following graph queries are not computable in AC^0: Does a node x lie on a cycle? Are two

nodes x and y connected by a path? Do nodes x and y have the same distance to node z? Does node x belong to a connected component that is acyclic?

Theorem 3 provides an essentially complete picture of the Gaifman locality of Arb-invariant FO. Similar to the case of order-invariant FO, the Hanf locality of Arb-invariant FO queries is still open in general, but if we restrict our attention to structures that represent strings, we can establish Hanf locality with the same bounds as in Theorem 3. In the following statement, Arb-invariant FO(*Succ*) refers to Arb-invariant queries over string structures.

Theorem 4. *Every Arb-invariant* FO(*Succ*) *formula is Hanf* $(\log n)^c$*-local for some constant* c *depending on the formula, and for every constant* c *there exists an Arb-invariant* FO(*Succ*) *formula that is not Hanf* $(\log n)^c$*-local.*

The upper bound in Theorem 4 means the following, where we use r to denote $(\log n)^c$: For any Arb-invariant FO query over strings and any large enough number n, if two strings of length n have the same prefix of length $2r$, the same suffix of length $2r$, and the same multiset of substrings of length $2r + 1$, then they cannot be distinguished by the query. Since Hanf locality implies Gaifman locality, the lower bound in Theorem 4 can be viewed as a strengthening of the lower bound part of Theorem 3.

We also present an application of our locality results to the study of regular languages. It is known that the class of definable regular languages does not grow when we move from FO to order-invariant FO [BS09], but does grow when we proceed to addition-invariant FO, i.e., Arb-invariant FO where the only numerical predicate used is addition [SS10b]. The larger class coincides with FO(*Succ*, *lm*), i.e., the extension of FO(*Succ*) with predicates determining the length of a string modulo some constant. Based on our locality results, we can show that the class does not grow further when we allow the use of arbitrary numerical predicates, i.e., when we proceed from addition-invariant FO to Arb-invariant FO.

Theorem 5. *A regular language is definable in Arb-invariant* FO(*Succ*) *iff it is definable in* FO(*Succ*, *lm*).

1.3 Kernelization

A *parameterized problem* is a language extended with a parameter k: $L \subseteq \{0,1\}^* \times \mathbb{N}$. One natural instance of a parameterized problem is k-VERTEXCOVER, the problem of deciding whether a given graph G on n vertices has a subset S of the vertices of size at most k such that each edge in G is incident to some vertex in S. See the survey [GN07] for other examples and more background.

A *kernelization procedure* is an algorithm that takes an instance of a parameterized problem and maps it to an instance of the same problem where the size of the instance depends only on the parameter and the running time of the procedure is polynomial in the input size. k-VERTEXCOVER has a simple kernelization procedure: Select a vertex in the graph with degree greater than k, put the vertex into the cover and remove the vertex from the graph. Such vertices must appear in the cover if it exists; otherwise the cover will have size at least $k+1$. Repeating this procedure at most k times will reduce the graph to a vertex cover instance with $O(k^2)$ vertices and edges, or determine that a k-cover is impossible. This reduction runs in time polynomial in the input size, and produces a size $O(k^2)$ kernel.

At this point, it is natural to ask whether anything better can be done. Dell and Van Melkebeek answer this question by showing that for vertex cover instances on n-vertex graphs *no* deterministic polynomial-time mapping reduction exists to instances of size $O(n^c)$ (for any language L), for any $c < 2$, unless the polynomial hierarchy collapses [DvM10]. In an earlier weaker version of this result [DvM09] use set systems that realize lines in the following sense.

For some field $\mathbb{F} = \mathrm{GF}(p)$, consider the *line* $L_{a,b}$ in $\mathbb{F}^2$ for $a, b \in \mathbb{F}$ defined by $L_{a,b} \doteq \{(x, a + bx) \mid x \in \mathbb{F}\}$. Let $\mathcal{L}$ be the family of lines $L_{a,b}$ for all $a, b \in \mathbb{F}$. We say that a set family $\mathcal{F}$ over $\mathbb{F}^2$ *realizes* $\mathcal{L}$ if for every line $L \in \mathcal{L}$ there exist two sets $Q, S \in \mathcal{F}$ such that $Q \cap S = L$.

Now, suppose we have a set family $\mathcal{F}$ with $|\mathcal{F}| = p^d$ that realizes $\mathcal{L}$. Dell and Van Melkebeek show that vertex cover does not have kernels of size n^c for any $c < \frac{2}{d}$, unless the polynomial hierarchy collapses [DvM09].

Results

We show essentially optimal bounds on d. We describe such a set family $\mathcal{F}$ of size $O(p^{3/2})$ that realizes $\mathcal{L}$. This rules out the kernelization of generic vertex cover instances to size n^c for any $c < \frac{4}{3}$. We then argue that a family of this size is necessary. This implies that using the above approach and set systems to rule out kernels of size $n^{\frac{4}{3}}$ or larger is not possible.

Theorem 6. *Let $\mathbb{F} \doteq \mathrm{GF}(p)$, for $a, b \in \mathbb{F}$ $L_{a,b} \doteq \{(x, a + bx) \mid x \in \mathbb{F}\}$, and $\mathcal{L} \doteq \cup_{a,b \in \mathbb{F}} L_{a,b}$. Then the number of sets sufficient and necessary to realize $\mathcal{L}$ is $\Theta(p^{1.5})$.*

The upper bound is fairly straightforward; the lower bound follows via a nontrivial geometric argument. Observe that the size of such a set family is bounded above by p^2 (pick $\mathcal{F}$ to be all of $\mathcal{L}$) and bounded below by p (because $|\mathcal{F}|^2$ must be at least $|\mathcal{L}| = p^2$).

2 Notation and Basics

We could, of course, use any notation we want; do not laugh at notations; invent them, they are powerful. In fact, mathematics is, to a large extent, invention of better notations.

— Richard Feynman

In this chapter we first describe some general notation then proceed to detail the various models of computation that we consider in this dissertation: arithmetic formulas, Boolean circuits, and first-order formulas.

2.1 General Notation

Let $\mathbb{F}$ denote a field, finite or otherwise, and let $\bar{\mathbb{F}}$ denote its algebraic closure. For a prime p, $\mathrm{GF}(p)$ is the Galois field of order p. For the most part we use the basic properties of fields implicitly, see, for example, the textbook [Isa09] for more background information. For an integer n, let $[n]$ denote the set $\{1, 2, \ldots, n\}$. We use the notation $|\cdot|$ to represent the size of various objects, like sets and binary strings. We use the following shorthand for asymptotic notation: We write $p = \mathrm{poly}(f_1, f_2, \ldots, f_r)$ to indicate that there exists a constant c such that $p = O((f_1 \cdot f_2 \cdot \ldots \cdot f_r)^c)$.

2.2 Arithmetic Formulas

Let $\mathbb{F}[x_1, \ldots, x_n]$ denote the polynomial ring in the variables x_1 through x_n over the field $\mathbb{F}$. In this dissertation we will be considering only *finite* polynomials $P \in \mathbb{F}[x_1, \ldots, x_n]$. We often denote variables interchangeably by their index or by their label: i versus x_i, and

$[n] = \{1, 2, \ldots, n\}$ versus $\{x_1, ..., x_n\}$. We frequently drop the index and refer to a generic variable as $x \in \{x_1, \ldots, x_n\}$.

For a subset of the variables $X \subseteq \{x_1, ..., x_n\}$ and an assignment $\bar{\alpha} \in \mathbb{F}^{|X|}$, $P|_{X \leftarrow \bar{\alpha}}$ denotes the polynomial P with the variables in X substituted by the corresponding values in $\bar{\alpha}$. When $X = \{x_1, \ldots, x_n\}$ is the full set of variables we shorten this substitution to $P(\bar{\alpha})$.

We say that a polynomial P *depends* on a variable x_i if there are two elements $\bar{\alpha}, \bar{\beta} \in \bar{\mathbb{F}}^n$ differing only in the i^{th} coordinate for which $P|_{[n] \leftarrow \bar{\alpha}} \neq P|_{[n] \leftarrow \bar{\beta}}$. We denote by $\mathrm{var}(P)$ the set of variables that P depends on.

One means of describing the complexity of evaluating a polynomial is by the size of an arithmetic circuit that computes it.

An *arithmetic circuit* is a rooted directed acyclic graph whose *inputs* (nodes with no incoming edges) are labeled with variables in $\{x_1, \ldots, x_n\}$ or by elements of $\mathbb{F}$. The *gates* (that is, those nodes with incoming edges) are labeled with addition $+$ or multiplication $\times$. The singular root node with no outgoing edges is called the output gate of the circuit. Naturally, each gate in an arithmetic circuit computes a polynomial of the nodes that point to it and a polynomial $\mathbb{F}^n \to \mathbb{F}$ of the input variables. We interchangeably use the notions of a gate and the polynomial in the input variables computed by that gate.

An *arithmetic formula* is simply an arithmetic circuit where the *fan-out* (or the number of out-going edges) of any gate is at most one. We allow generalized input, addition, and multiplication gates, where the result can be multiplied by and/or added to a constant. Formally, for arbitrary constants $\alpha, \beta \in \mathbb{F}$, input and arithmetic gates produce the following output, where the g_i's are the polynomials computed by the gate's children.

- Input gates: $\alpha \cdot x + \beta$.
- Addition gates: $g = \alpha \cdot (\sum_i g_i) + \beta$.

- Multiplication gates: $g = \alpha \cdot (\prod_i g_i) + \beta$.

The *size* of an arithmetic formula is the number of wires together with the total bit length of all the constant fields in the formula. The *depth* of an arithmetic formula is the length of the longest path from the output gate to any input. Except when we are discussing constant depth formulas we will assume that the fan-in of multiplication and addition gates is two; it is not difficult to see that formulas with arbitrary fan-in can be transformed into formulas with fan-in two at the cost of a constant factor increase in formula size. Note that a given formula can be evaluated over an assignment to the variables from the base field (or an extension field) in time polynomial in the size of the formula and the bit length of the assignment. See the excellent survey [SY10] for more background information on arithmetic circuits and formulas, including basic properties and algorithms.

We can define an analogous notion of complexity for Boolean languages.

2.3 Boolean Circuits

A *Boolean circuit* C is a rooted directed acyclic graph whose inputs are labeled with Boolean variables $\{x_1, \ldots, x_n\}$ or their negations $\{\neg x_1, \ldots, \neg x_n\}$. The gates of C are labeled with either $\wedge$ or $\vee$. Each gate of the circuit C naturally defines a function from $\{0,1\}^n$ to $\{0,1\}$. The *depth* of a circuit is the length of a longest path from the root of the circuit to one of its inputs. The *size* of a circuit is the number of wires it contains.

A *circuit family* $\mathcal{C}$ is a sequence $(C_n)_{n\in\mathbb{N}}$ such that for all $n \in \mathbb{N}$, C_n is a circuit over n input variables. We say that a circuit C *accepts* a string w if evaluating C on w outputs 1, otherwise $C(w) = 0$ and we say that C *rejects* w. We say that a *Boolean language* $L \subseteq \{0,1\}^*$ is accepted by a family of circuits $(C_n)_{n\in\mathbb{N}}$ if for all $n \in \mathbb{N}$ and for all binary strings w of length n, $C_n(w) = 1$ iff $w \in L$.

A language L is in AC^0 if there exists a family of circuits accepting L that have constant depth and size polynomial in their input length n. A sequence of celebrated results shows that AC^0 is not even able to compute *parity*, that is, whether the number of ones in its input is even or odd [Ajt83, FSS84, Yao85, Hås86, Cai89]. In fact, we use the following somewhat stronger promise version. For a binary string $w \in \{0,1\}^*$, let $|w|_1$ denote the number of ones in w.

Lemma 2.1 (Implicit in [Hås86, Theorem 5.1]). *For any $d \in \mathbb{N}$, there are constants $\alpha > 0$ and $n_0 > 0$ such that for all $n \geq n_0$ there is no circuit of depth d and size $2^{\alpha n^{1/(d-1)}}$ that accepts all inputs $w \in \{0,1\}^{2n}$ with $|w|_1 = n$ and rejects all inputs with $|w|_1 = n+1$.*

To discuss circuits over graphs we must first discuss finite structures, which are intuitively, a natural generalization of graphs.

2.4 Finite Model Theory and First-Order Logic

A *relational schema* is a set of symbols each with an associated arity. A *structure* M over a relational schema τ is a *finite* set $dom(M)$, the *domain*, containing all the *elements* of M, together with an interpretation $R^M \subseteq dom(M)^k$ of each relation symbol $R \in \tau$ of arity k. We call a structure over a relational schema τ a *τ-structure*. The *size* of a structure M is the cardinality of its domain $dom(M)$.

For a set S of elements in the domain of M, $M_{|S}$ denotes the *induced substructure of M on S*. That is, $M_{|S}$ is the structure whose domain is S and whose relations are the relations of M restricted to those tuples containing only elements in S.

We say that two τ-structures M and M' are *isomorphic*, $M \cong M'$, if there exists a bijection $\pi : dom(M) \to dom(M')$ such that for each k-ary relation symbol $R \in \tau$, $(a_1, a_2, \ldots, a_k) \in R^M$ iff $(\pi(a_1), \pi(a_2), \ldots, \pi(a_k)) \in R^{M'}$. We write $\pi : M \cong M'$ to indicate

that π is an isomorphism that maps M to M'. If $\bar{a}$ and $\bar{b}$ are tuples (of the same length) of elements of $dom(M)$ and $dom(M')$, respectively, then we write $(M, \bar{a}) \cong (M', \bar{b})$ to indicate that there is an isomorphism $\pi : M \cong M'$ which maps $\bar{a}$ to $\bar{b}$. All *classes* of structures considered in this dissertation are closed under isomorphisms.

A *k-ary query on τ-structures* is a mapping q that associates with each τ-structure M a relation $q(M) \subseteq dom(M)^k$, and that is closed under isomorphism in the following sense: If $(M, \bar{a}) \cong (M', \bar{b})$, then $\bar{a} \in q(M)$ iff $\bar{b} \in q(M')$.

Queries over finite structure are naturally computed by logical formulas.

2.4.1 First-Order Logic

One standard class of logical formulas are *first-order formulas*. We denote by FO(τ) the first-order logic with respect to the schema τ. This is the set of logical formulas whose atoms are formed based on the relation symbols in τ, the equality symbol $=$, and an infinite sequence of variables $(x_1, x_2, \ldots)$, and that is closed under Boolean connectives ($\wedge, \vee$, and $\neg$) and existential and universal quantifications ($\exists$ and $\forall$). We use the standard syntax and semantics for FO (cf., e.g., [Lib04]). We write $M \models \phi(\bar{a})$ or $(M, \bar{a}) \models \phi(\bar{x})$ to express that the tuple $\bar{a}$ of elements in $dom(M)$ makes the formula $\phi(\bar{x})$ true in M. A formula $\phi(\bar{x})$ with k free variables defines the k-ary query that associates with every τ-structure M the set of k-tuples $\bar{a} \in dom(M)^k$ for which $M \models \phi(\bar{a})$. Sometimes, we will say that $\phi(\bar{x})$ is a *k-ary formula*. A *sentence* is a formula that has no free variables.

One measure of logical formula size is *alternation depth*. The alternation depth of a formula is the maximum, over all paths from the root of a formula to its atoms, of the number of alternating blocks of quantifiers along the path.

Queries may also be computed by Boolean circuits.

2.4.2 Representing Structures and Queries as Strings

In order to enable circuits to act on structures and compute queries, we need to specify how to represent a τ-structure M and a k-tuple $\bar{a} \in dom(M)^k$ as a bit-string. Our results are robust with respect to the details of the encoding. For concreteness, we use the following scheme based on characteristic sequences.

Let $<$ be a linear order on $dom(M)$. Let $R_1, \ldots, R_s$ be a list of the relation symbols in τ and let $r_1, \ldots, r_s$ be the arities of these symbols. For each $R_i \in \tau$, we denote by $enc_<(R_i^M)$ the bit-string of length $|dom(M)|^{r_i}$ whose jth bit is 1 iff the jth smallest element in $dom(M)^{r_i}$ w.r.t. the lexicographic order associated with $<$ belongs to the relation R_i^M. Similarly, for each component a_i of the k-tuple $\bar{a}$, we let $enc_<(a_i)$ be the bit-string of length $|dom(M)|$ whose jth bit is 1 iff a_i is the jth smallest element of $dom(M)$ w.r.t. $<$. Finally, we let

$$enc_<(M, \bar{a}) \;:=\; enc_<(R_1^M) \cdots enc_<(R_s^M)\, enc_<(a_1) \cdots enc_<(a_k)$$

be the *binary encoding of* $(M, \bar{a})$ *w.r.t.* $<$.

The above encoding presumes a linear order $<$ on $dom(M)$. For ordered structures, i.e., structures with an associated order on their domain, the choice of $<$ is fixed. For unordered structures – the ones we care about – we consider all possible linear orders and let

$$\mathrm{Rep}(M, \bar{a}) \;:=\; \{enc_<(M, \bar{a}) \;:<\text{ is a linear order on } dom(M)\}$$

denote the set of all binary encodings of $(M, \bar{a})$. Note that $\mathrm{Rep}(M, \bar{a}) = \mathrm{Rep}(M', \bar{b})$ iff $(M, \bar{a}) \cong (M', \bar{b})$.

For a circuit family $\mathcal{C} = (C_m)_{m \in \mathbb{N}}$ to compute a k-ary query q on τ-structures, we require that it produces the correct result for all possible representations. In other words, for all τ-structures M, all k-tuples $\bar{a} \in dom(M)^k$, and all strings $\Gamma \in \mathrm{Rep}(M, \bar{a})$, we have that

$C_{|\Gamma|}(\Gamma) = 1$ iff $\bar{a} \in q(M)$. Note that for every fixed M and $\bar{a}$, all representations in $\mathrm{Rep}(M, \bar{a})$ have the same length, so the same circuit of the family $\mathcal{C}$ acts on all of them.

3 Deterministic Polynomial Identity Tests

Perhaps it's impossible to wear an identity without becoming what you pretend to be.

— Orson Scott Card

In this chapter we discuss our deterministic polynomial identity tests for bounded-read multilinear formulas and related generalizations, both in the blackbox and non-blackbox settings. In Section 3.1 we give the formal statements of our results and an overview of the techniques we employ. In Section 3.2 we further discuss related work. In Section 3.3 we introduce our notation and formally define the classes of arithmetic formulas that we study. Section 3.3 also reviews some properties of tools that we use from prior work. In Section 3.4 we present a structural property of non-zero arithmetic formulas that witnesses their non-zeroness and give a new result, in this context, that improves on the parameters of prior work. In Section 3.5 we develop our main technical workhorse, the Fragmentation Lemma, in a step-wise fashion – for read-once formulas, read-k formulas, and sparse-substituted formulas – and the Shattering Lemma that is based on it.

We develop our blackbox and non-blackbox identity tests in parallel. Each is built using two reductions. In Section 3.6 we reduce PIT for structurally-multilinear sparse-substituted read-$(k+1)$ formulas to PIT for structurally-multilinear sparse-substituted $\sum^2$-read-k formulas. In Section 3.7 we reduce PIT for structurally-multilinear sparse-substituted $\sum^2$-read-k formulas to PIT for structurally-multilinear sparse-substituted read-k formulas. In Section 3.8 we prove our two main theorems for identity testing structurally-multilinear sparse-substituted constant-read formulas: a deterministic polynomial-time non-blackbox algorithm and a deterministic quasi-polynomial-time blackbox algorithm. We end with a

specialization of our approach that gives a deterministic polynomial-time blackbox algorithm for multilinear constant-read constant-depth formulas. In Section 3.9 we describe some avenues for further research and some recent developments.

3.1 Overview

In this subsection we give an overview of our results and techniques. For clarity we focus on the case of multilinear constant-read formulas. At the end of this section we briefly discuss the complexities that the extension to structurally-multilinear sparse-substituted formulas entails. We begin with the formal statement of our main theorem (Theorem 1 from the introduction).

Theorem 7 (PIT for Multilinear Bounded-Read Formulas). *There exists a deterministic polynomial identity testing algorithm for multilinear formulas that runs in time* $s^{O(1)} \cdot n^{k^{O(k)}}$*, where* s *denotes the size of the formula,* n *the number of variables, and* k *the maximum number of times a variable appears in the formula. There also exists a deterministic blackbox algorithm that runs in time* $n^{k^{O(k)}+O(k \log n)}$ *and queries points from an extension field of size* $O(n^2)$*.*

As mentioned earlier, polynomial identity testing is trivial for read-once formulas. Our overall approach for multilinear constant-read formulas is a recursive one in which we reduce to instances with smaller read-value and/or fewer variables until we reach a trivial case. Our reduction alternates between two steps and uses as an intermediate stage formulas that are the sum of two multilinear read-k formulas. We refer to such formulas as multilinear $\sum^2$-read-k formulas.

Step 1. Reduce PIT for multilinear read-$(k+1)$ formulas to PIT for multilinear $\sum^2$-read-k formulas and PIT for multilinear read-$(k+1)$ formulas on half the number of variables.

Step 2. Reduce PIT for multilinear $\sum^2$-read-k formulas to PIT for multilinear read-k formulas.

A key concept in our tests is the notion of a structural witness: A *structural witness* is a means of exhibiting a formula is nonzero by examining its structure (rather than finding a particular point where the formula is nonzero). We develop a technique, which we call *shattering*, that exposes structural witnesses using a few well-chosen partial derivatives. This allows us to bring the known structural witnesses for depth-three formulas (namely, [DS07, SS10a], as seen through [KMSV10]) to bear on multilinear $\sum^2$-read-k formulas, and enables us to realize Step 2 in both the blackbox and non-blackbox settings. Our shattering technique builds on a simpler technique, which we call *fragmentation*, that breaks up a formula into the product of small factors using a single partial derivative (this generalizes an idea of [KMSV10]), and which we also use to realize Step 1 in the blackbox setting. A key technical difficulty lies in showing how fragmentation enables shattering. Ultimately, one can view our approach as unifying the techniques developed for sums of read-once formulas (the SV-generator from [SV09]) and multilinear depth-four formulas by using the novel techniques mentioned above to explore the structure of bounded-read formulas.

In general, a blackbox PIT algorithm for a class $\mathcal{F}$ of formulas is equivalent to the construction of a low-degree polynomial mapping G on few variables such that $F \circ G$ is nonzero for every nonzero $F \in \mathcal{F}$. We refer to such a mapping as a *hitting set generator* for $\mathcal{F}$, and say that G *hits* every $F \in \mathcal{F}$. Testing F on all elements in the image of G when the input variables to G range over some small set produces an identity test. The converse is also true – identity tests imply hitting set generators (see e.g. [SV09]).

We now discuss the two steps and their key ingredients in more detail, with a focus on the role of fragmentation.

3.1.1 Fragmenting Multilinear Formulas

Our fragmentation technique for multilinear formulas involves partial derivatives with respect to well-chosen variables. In the simple case of read-once formulas F, the idea boils down to the following observation: Taking the partial derivative with respect to the median variable x on which F depends in leaf order, yields a nonzero formula $\partial_x F$ that is the product of subformulas each of which depends on at most half the variables. For a general multilinear read-$(k+1)$ formula on n variables, a similar procedure yields the following.

Lemma 3.1 (Simplified Fragmentation Lemma). *Given a nonzero multilinear read-$(k+1)$ formula F on n variables, there exists a variable x such that $\partial_x F$ is nonzero and can be written as the product of subformulas on at most $n/2$ variables each, and possibly one other formula that is the derivative of a $\sum^2$-read-k subformula.*

The Fragmentation Lemma helps us in realizing the blackbox version of Step 1 as follows. A hitting set generator for a class $\mathcal{F}$ of formulas also hits products of formulas from $\mathcal{F}$. Multilinear formula classes (e.g., read-$(k+1)$ or $\sum^2$-read-k) are closed under the action of partial derivatives, because for any variable at most one input to each gate may depend on that variable and, hence, the product rule is simplified. Thus, by the Fragmentation Lemma, a hitting set generator that hits multilinear $\sum^2$-read-k formulas on n variables as well as multilinear read-$(k+1)$ formulas that depend on at most $n/2$ of the n variables, also hits some nonzero partial derivative of any nonzero multilinear read-$(k+1)$ formula on n variables. Adding an independent random field element turns such a hitting set generator into one that hits every multilinear read-$(k+1)$ formula on n variables. A logarithmic

number of applications of this transformation then turns a hitting set generator for n-variate $\sum^2$-read-k formulas into one for n-variate read-$(k+1)$ formulas.

We also use the Fragmentation Lemma as a building block to establish our Shattering Lemma. This is the most involved step in our construction. Once we have our Shattering Lemma, we employ two more ingredients: the SV-generator and structural witnesses for depth-three formulas. At a high level, the Shattering Lemma allows us to transform multilinear read-k formulas into depth-three formulas for which structural witnesses are known, and the latter enables us to apply the SV-generator and realize Step 2. We first introduce these additional ingredients, then explain how they combine with the Shattering Lemma, and finally sketch how to obtain shattering from fragmentation.

3.1.2 SV-Generator

Shpilka and Volkovich [SV09] defined a hitting set generator G_w that interpolates all 0-1-vectors of weight at most w (i.e., it contains all such vectors in its image) and has some additional closure properties. Their approach for sums of a constant number of read-once formulas is based on two facts.

Fact 3.1 ([SV09]). *Let $\mathcal{D}_\ell$ denote the class of nonzero polynomials that are divisible by at least ℓ distinct variables. G_w is a hitting set generator for any class $\mathcal{F}$ of multilinear polynomials that*

1. *is closed under zero-substitutions and*
2. *is disjoint from $\mathcal{D}_\ell$ for every $\ell > w$.*

Fact 3.2 ([SV09]). [1] *Let $F_i \in \mathbb{F}[x_1, \ldots, x_n] = \mathbb{F}[\bar{x}]$ be read-once formulas and $F = \sum_{i=1}^{k} F_i$ be a nonzero formula. Let $\bar{\sigma}$ be a point in $\mathbb{F}^n$ where none of the nonzero first-order partial derivatives of the F_i's vanish. Then $F(\bar{x} + \bar{\sigma}) \notin \mathcal{D}_\ell$ for any $\ell \geq 3k$.*

For a formula F as in Fact 3.2, consider applying Fact 3.1 to the class $\mathcal{F}$ consisting of $F(\bar{x} + \bar{\sigma})$ and all its zero-substitutions, for some fixed $\bar{\sigma}$. The first condition of Fact 3.1, the closure under zero-substitutions of $\mathcal{F}$, holds by construction. As for the second condition, consider a formula F' obtained by substituting into F the corresponding components of $\bar{\sigma}$ for some subset $X \subseteq \{x_1, \ldots, x_n\}$ of the variables. For any variable $x_i \notin X$, we have that $\frac{\partial F'}{\partial x_i}(\bar{\sigma}) = \frac{\partial F}{\partial x_i}(\bar{\sigma})$. Thus, if $\bar{\sigma}$ satisfies the hypothesis of Fact 3.2 for F, then it also satisfies that hypothesis for any partially substituted F' of the above type. By Fact 3.2, this shows that $F'(\bar{x} + \bar{\sigma}) \notin \mathcal{D}_\ell$ for $\ell \geq 3k$. Noting that $F'(\bar{x} + \bar{\sigma})$ coincides with $F(\bar{x} + \bar{\sigma})$ where all variables in X have been substituted by zero, this means that $\mathcal{F}$ satisfies the second condition of Fact 3.1. We conclude that for any $\bar{\sigma}$ satisfying the condition of Fact 3.2 G_{3k} hits $F(\bar{x} + \bar{\sigma})$, and hence $G_{3k} + \bar{\sigma}$ hits F.

In addition, since the partial derivatives $\partial_x F_i$ are read-once formulas and PIT for read-once formulas is trivial, we can efficiently find a shift $\bar{\sigma}$ satisfying the conditions of Fact 3.2 when given access to the formula F – select values for the components of $\bar{\sigma}$ one by one so as to maintain non-zeroness of the nonzero partial derivatives under that setting. This is how Shpilka and Volkovich obtained their polynomial-time non-blackbox test [SV08]. Alternately, one can use a hitting set generator $\mathcal{G}$ for read-once formulas to generate a shift $\bar{\sigma}$ satisfying the conditions of Fact 3.2. Fact 3.1 then shows that $\mathcal{G} + G_{3k}$ is a hitting set generator for sums of k read-once formulas. This is how Shpilka and Volkovich obtained their quasi-polynomial-time blackbox test [SV09].

[1] Shpilka and Volkovich refer to this fact as a hardness of representation result and use the term "justifying assignment" for $\bar{\sigma}$.

Although more involved, our approach follows the same strategy for Step 2, i.e., to reduce PIT for multilinear $\sum^2$-read-k formulas to PIT for multilinear read-k formulas. We use Fact 3.1 as is, and develop the following equivalent of Fact 3.2 for sums of (two) multilinear read-k formulas.

Lemma 3.2 (Informal Simplified Key Lemma). *Let F be a $\sum^2$-read-k formula, and $\bar{\sigma}$ a point where none of the nonzero partial derivatives of small order of any subformula of F vanish. Then $F(\bar{x} + \bar{\sigma}) \notin \mathcal{D}_\ell$ for any ℓ that is sufficiently large with respect to k.*

Note that the condition of the Key Lemma involves higher-order derivatives, whereas the corresponding condition in Fact 3.2 only uses first-order derivatives. Nevertheless, the important properties are preserved: (i) the condition implies that the conclusion holds for $F(\bar{x} + \bar{\sigma})$ as well as for all its zero-substitutions, and (ii) the condition states that $\bar{\sigma}$ is a common nonzero of some nonzero multilinear read-k formulas which we can easily compute from F.

Thus, given access to F and to an oracle to a polynomial identity test for multilinear read-k formulas, we can efficiently construct a shift $\bar{\sigma}$ such that G_w hits $F(\bar{x} + \bar{\sigma})$ for w sufficiently large with respect to k. This gives our non-blackbox reduction from PIT on multilinear $\sum^2$-read-k formulas to PIT on multilinear read-k formulas. In the blackbox setting, we can generate the shift $\bar{\sigma}$ we need using a hitting set generator $\mathcal{G}$ for multilinear read-k formulas, resulting in $\mathcal{G} + G_w$ as a hitting set generator for multilinear $\sum^2$-read-k formulas.

Shpilka and Volkovich show Fact 3.2 by arguing that applying a sequence of partial derivatives and nonzero substitutions to F reduces the degree of the terms in $\mathcal{D}_\ell$ and zeroes some of the F_i's. If $\mathcal{D}_\ell$ remains non-trivial after all F_i's are zeroed, the fact is proved. The bound they derive on ℓ depends on how quickly the fanin k is reduced relative to the number of operations performed by the argument. Strong structural properties of *read-once* formulas

make it relatively easy to argue that few partial derivatives and substitutions suffice to zero any particular read-once formula. For formulas of *arbitrary read* these properties are not readily present. In order to prove the Key Lemma, we employ our fragmentation technique to bring the known structural witnesses for depth-three formulas to bear on multilinear constant-read formulas.

3.1.3 Structural Witnesses

Derandomizing polynomial identity testing means coming up with deterministic procedures that exhibit witnesses for nonzero circuits. The most obvious type of witness consists of a point where the polynomial assumes a nonzero value; such witnesses are used in blackbox tests. For restricted classes of circuits one may hope to exploit their structure and come up with other types of witnesses. The prior deterministic PIT algorithms we mentioned [KS08, SV08, KS09, SV09, SS09, KMSV10, SV11] follow the latter general outline. More specifically, they exhibit a measure for the complexity of the restricted circuit that can be efficiently computed when given the circuit as input, and prove that (i) restricted circuits that are zero have low complexity, and (ii) restricted circuits of low complexity are easy to test. This framework immediately yields a non-blackbox PIT algorithm for the restricted class of circuits, and in several cases also forms the basis for a blackbox algorithm. Complexity measures that have been successfully used within this framework are the rank of depth-three circuits [DS07, KS08, SV08, KS09, SV09, SS09, KMSV10] and, very recently, the sparsity of multilinear depth-four circuits [SV11].

To derive their results for multilinear depth-four formulas, Karnin et al. [KMSV10] established the following structural witness for formulas F that are the sum of "split" formulas. A "split" formula is the product of subformulas that each only depend on a fraction of the variables.

Fact 3.3 (Informal and implicit in [KMSV10]). *A simple and minimal*[2] *sum of a constant number of sufficiently split*[3] *multilinear formulas cannot be zero.*

For the precise quantitative version of Fact 3.3 we refer to Section 3.4, where we also give a new self-contained proof that yields slightly better parameters than the original one. For the moment it suffices to say that the conditions of simplicity and minimality are relatively easy to deal with.

We use Fact 3.3, not to directly construct our PIT algorithm as in earlier works, but to establish the Key Lemma (Lemma 3.2). The connection between the two is as follows. Let F be a sum of a constant number of multilinear formulas. Note that $F(\bar{x}+\bar{\sigma}) \in \mathcal{D}_\ell$ iff there exists a nonzero formula Q and an index set I of size ℓ such that $F(\bar{x}+\bar{\sigma}) - Q \cdot \prod_{i \in I} x_i \equiv 0$. Since F is multilinear, Q cannot depend on any variable in I. Thus there is an assignment to the variables not in I that does not zero Q (it turns out that substituting $\bar{\sigma}$ works), and hence we can assume that $F'(\bar{x}+\bar{\sigma}) - a \cdot \prod_{i \in I} x_i \equiv 0$ for some nonzero scalar a. Fact 3.3 shows that the latter cannot happen for $\ell > 0$ if each of the summands of F' is (i) sufficiently split and (ii) not divisible by any variable. For the shifted formula $F'(\bar{x}+\bar{\sigma})$ the latter condition is met if the summands of F' do not vanish at $\bar{\sigma}$. Thus, in order to establish the Key Lemma, all that remains is to transform multilinear $\sum^2$-read-k formula $F'(\bar{x}+\bar{\sigma}) \in \mathcal{D}_\ell$ into a sum $F''(\bar{x}+\bar{\sigma})$ of a constant number of sufficiently split multilinear formulas such that $F''(\bar{x}+\bar{\sigma}) \in \mathcal{D}_{\ell'}$ for some $\ell' > 0$. Moreover, the transformation should be sufficiently simple so that the condition that none of the summands of F' vanish at $\bar{\sigma}$ translates into a simple condition about $\bar{\sigma}$ and the original formula F. Repeated applications of the Fragmentation Lemma allow us to do so (for ℓ sufficiently large compared to k) in a process we refer to as "shattering".

[2] *Simplicity* means that there is no non-trivial factor that is common to all summands of F. *Minimality* means that no non-trivial subset of the summands of F sums to zero. Intuitively, these two conditions mean that there is no "easy" way to make F less complex.

[3] Karnin et al.[KMSV10] refer to "split" formulas as "compressed".

3.1.4 Shattering Multilinear Formulas

For the purpose of exposition, let us consider the case $k = 1$, i.e., let $F = F_1 + F_2$ be the sum of two read-once formulas. The Fragmentation Lemma applied to F_i gives a formula $\partial_x F_i$ that is a product of subformulas on at most half of the variables each. When we greedily apply the Fragmentation lemma to a factor which depends on the most variables, $O(1/\alpha)$ applications suffice to ensure that each of the remaining factors depend on at most a fraction α of the variables. If we denote by P the set of variables we used for the partial derivatives, multilinearity implies that the product of all those factors equals $\partial_P F_i$. Thus, the formula $F' \doteq \partial_P F$ is the sum of two split multilinear formulas. Moreover, if $F(\bar{x} + \bar{\sigma}) \in \mathcal{D}_\ell$ then $F'(\bar{x}+\bar{\sigma}) \in \mathcal{D}_{\ell'}$ for $\ell' = \ell - |P|$, which is positive as long as ℓ is sufficiently large compared to $1/\alpha$. Let F'_i coincide with $\partial_P F_i$, so the condition that F'_i does not vanish at $\bar{\sigma}$ is equivalent to $\partial_P F_i$ not vanishing at $\bar{\sigma}$. This is how higher-order derivatives enter the conditions of the Key Lemma.

In the cases where $k \geq 2$ the shattering process becomes more complicated as it no longer holds that all the factors produced by the Fragmentation Lemma depend on at most half the number of variables – the one $\sum^2$-read-$(k-1)$ factor may depend on more.

We handle this situation by explicitly expanding the $\sum^2$-read-$(k-1)$ formula into the sum of two read-$(k-1)$ formulas and propagating the sum up to the top addition gate of F'. This increases the top fanin of F' and duplicates some of the variable occurrences. However, by restricting to the variables that only appear in the $\sum^2$-read-$(k-1)$ formula and then further restricting the variables to the largest group that appear the exact same number of times in the larger of the two terms, we can ensure that the sum of the read-values of the children of F' does not increase. As a result, we need to apply this expansion operation no more than $2k$ times, and the top fanin of F' never grows above $2k$. Since we only apply the operation when the $\sum^2$-read-$(k-1)$ formula depends on many variables, the subset of

the variables to which we restrict remains large. This result of this process is qualitatively captured in the following lemma.

Lemma 3.3 (Informal Simplified Shattering Lemma). *For any $\sum^2$-read-k formula F, there exist disjoint sets of indices P and V, with P small and V relatively large, such that $\partial_P F$ can be written as $\sum_{j=1}^{m} F'_j$ where $m \leq 2k$ and each F'_j is split with respect to V and is the product of subformulas of partial derivatives of F_1 and F_2.*

Note that the formula F' given by the Shattering Lemma may depend on variables outside of V, and that the F'_j's are only split with respect to V, i.e., they are the products of factors that each only depend on a fraction of the variables of V but may depend on many variables outside of V. The formula to which we apply Fact 3.3 is obtained from F' by setting the variables outside of V appropriately. If neither the projections nor any of the F_j vanish at $\bar{\sigma}$, we can conclude that $F(\bar{x} + \bar{\sigma}) \notin \mathcal{D}_\ell$ for any ℓ larger than the number of partial derivatives we needed for the shattering. Since the variables always appear as subformulas, the condition in the statement of the Key Lemma suffices.

3.1.5 Extension to Multilinear Sparse-Substituted Formulas

To extend our results to multilinear sparse-substituted formulas, only a few modifications are needed. Such a formula consists of a multilinear formula in which each leaf variable is replaced by a sparse multilinear polynomial in such a way that all multiplication gates of the original formula remain variable disjoint. The main extension happens in the Fragmentation Lemma. A combination of partial derivatives and zero-substitutions similar to the techniques used in [KMSV10] allows us to fragment the sparse substitutions. For substitutions that consist of at most t terms, this results in an overall multiplicative increase in the number of such operations by $\log t$. This factor propagates to the exponent of the running time of our algorithms. Making this argument rigorous gives the following theorem.

Theorem 8 (Extension to Multilinear Sparse-Substituted Formulas).
There exists a deterministic polynomial identity testing algorithm for multilinear sparse-substituted formulas that runs in time $s^{O(1)} \cdot n^{k^{O(k)}(\log(t)+1)}$, where s denotes the size of the formula, n the number of variables, k the maximum number of substitutions in which a variable appears, and t the maximum number of terms a substitution consists of. There also exists a deterministic blackbox algorithm for multilinear sparse-substituted formulas that runs in time $n^{k^{O(k)}(\log(t)+1)+O(k \log n)}$ and queries points from an extension field of size $O(n^2)$.

Note Theorem 7 is a specialization of Theorem 8 obtained by setting $t = 1$.

3.1.6 Extension to Structurally-Multilinear Sparse-Substituted Formulas

Our arguments thus far hinge on multilinearity, for two main reasons. First, we heavily use partial derivatives, and partial derivatives do not increase multilinear formula size. Second, the factors of multilinear formulas are variable disjoint.

We can relax the multilinearity condition somewhat, namely to structural multilinearity. The latter only requires that the multiplication gates of the original formula be variable disjoint under the sparse-substitution, but the sparse polynomials may be non-multilinear.

In the non-blackbox case, the extension to structurally-multilinear sparse-substituted formulas follows by a simple transformation from general sparse substitutions to multilinear sparse substitutions that preserves (non-)zeroness. This transformation is based on the fact that in structurally-multilinear formulas each variable power can be treated as a distinct variable since the structurally-multilinear condition forces all such products to be formed in the sparse-substituted polynomials. In the blackbox case, the extension is more difficult. We argue that a generalization of the Key Lemma holds for structurally-multilinear sparse-substituted formulas. We show that the transformation from non-blackbox case can be used

within the proof to reduce to the multilinear sparse-substituted version of the Key Lemma.

In both cases the effect of this extension on the running times of the algorithms is minimal, increasing the base of the exponent from n to dn.

Theorem 9 (Extension to Structurally-Multilinear Formulas). *There exists a deterministic polynomial identity testing algorithm for structurally-multilinear sparse-substituted formulas that runs in time* $s^{O(1)} \cdot (dn)^{k^{O(k)}(\log(t)+1)}$, *where* s *denotes the size of the formula,* n *the number of variables,* k *the maximum number of substitutions in which a variable appears,* t *the maximum number of terms a substitution consists of, and* d *the maximum degree of individual variables in the substitutions. There also exists a deterministic blackbox algorithm for structurally-multilinear sparse-substituted formulas that runs in time* $(dn)^{k^{O(k)}(\log(t)+1)+O(k \log n)}$ *and queries points from an extension field of size* $O(dn^2)$.

Note that this is the formal statement of Theorem 2 from the introduction and that Theorem 8 is a specialization of Theorem 9 obtained by setting $d = 1$.

3.2 Related Work

For depth-three formulas with constant top fanin Kayal and Saxena [KS07] present a non-blackbox algorithm that runs in polynomial time; Karnin and Shpilka [KS08] give a blackbox algorithm that runs in quasi-polynomial time, i.e., in time $2^{\text{polylog}\, s}$ on inputs of size s; for the special case when the field is the rationals or the reals polynomial-time blackbox algorithms are known [KS09, SS09]. Saxena and Seshadhri resolve this line of research, by giving a polynomial-time blackbox algorithm for depth-3 constant-top-fanin formulas over arbitrary fields [SS11].

For depth four Karnin et al. [KMSV10] established a deterministic quasi-polynomial-time blackbox algorithm for multilinear formulas. Parallel to our work, Saraf and Volkovich

[SV11] obtained a deterministic polynomial-time blackbox algorithm for multilinear formulas of depth four. There are also a few incomparable results for rather specialized classes of depth-four formulas [Sax08, AM10, SV09].

There is some prior work on depth-three constant-read formulas that does not require multilinearity. In particular, [KS08] gives a $n^{2^{O(k^2)}}$ blackbox identity testing algorithm for read-k depth-three formulas. Later work in [SS09] implies an improved running time of $n^{2^{O(k)}}$ for this algorithm. Note that Saxena and Seshadhri's $(dn)^{O(k)}$ time blackbox identity test for depth-three formulas with degree d and top fanin k [SS11] that we mentioned before subsumes these depth-three constant-read because one can efficiently transform a constant-read depth-three formula into a constant-top-fanin depth-three formula.

3.3 Background

In this section we first review some notation and basic properties of polynomials and arithmetic formulas. We then describe an ingredient we need from prior work, namely, the SV-generator, along with some related observations and extensions.

3.3.1 Polynomials and Arithmetic Formulas

We consider several restricted classes of arithmetic formulas. An arithmetic formula is *multilinear* if every gate of the formula computes a polynomial that has degree at most one in every variable. This means that only one child of a multiplication gate may depend on a particular variable. However, more than one child may contain occurrences of some variable. For example, the formula

$$(x_1 - x_2) \cdot ((x_1 + x_3) - x_1)$$

is multilinear, and although the second factor has occurrences of x_1 it does not depend on x_1.

We also consider the restriction that each variable occurs only a bounded number of times.

Definition 3.1 (read-k formula). *For $k \in \mathbb{N}$, a* read-k *formula is an arithmetic formula that has at most k occurrences of each variable. For a subset $V \subseteq [n]$, a* read$_V$-k *formula is an arithmetic formula that has at most k occurrences of each variable in V (and an unrestricted number of occurrences of variables outside of V).*

Observe that for $V = [n]$ the notion of read$_V$-k coincides with read-k.

Given a read-k formula, we can transform it in polynomial time into an equivalent formula in *standard form* that has at most $O(kn)$ gates and where constants only occur in the α and β of gates. The transformation preserves multilinearity and number of reads. The transformation is included in the next subsection for completeness. In the blackbox setting we can assume without loss of generality that the underlying read-k formula is in standard form. Our non-blackbox algorithms will always implicitly start by running the transformation.

We can build more complex formulas by adding several formulas together.

Definition 3.2 ($\sum^m$-read-k formula). *For $k, m \in \mathbb{N}$, a $\sum^m$-*read-k *formula is the sum of m read-k formulas.*

Note that any $\sum^m$-read-k formula is a read-(km) formula.

Finally, we also consider the above types of formulas in which variables can be replaced by sparse polynomials. We call a polynomial is *t-sparse* if it consists of at most t terms.

Definition 3.3 (sparse-substituted formula). *A* sparse-substituted *formula is an arithmetic formula where each leaf is replaced by a sparse multivariate polynomial given as a list of terms. Further,*

1. *if every variable occurs in at most k of the sparse polynomials, we say that the formula is* read-k *, and*

2. *if for every multiplication gate g and every variable x there is at most one multiplicand of g that depends on x, we say that the formula is* structurally multilinear*.*

A sparse-substituted formula is multilinear if every gate (including the substituted input gates) computes a multilinear polynomial. This is equivalent to all multiplication gates in the backbone formula being variable-disjoint, and the sparse substitutions being multilinear. The corresponding interpretation of structural multilinearity is that the multiplication gates in the backbone formula are variable-disjoint, but the substituted sparse polynomials may not be multilinear. Thus, structural multilinearity is more general than multilinearity. For brevity we often drop the quantifier "sparse-substituted" when discussing structurally-multilinear formulas.

3.3.1.1 Standard Form of Read-k Formulas

Given a read-k formula, it can be transformed into a standard form read-k formula where constants only occur in the α and β of gates and the formula has at most $O(kn)$ gates.

Proposition 3.1. *There is an algorithm that transforms a given read-k formula F on n variables into an equivalent read-k formula F', such that F' has at most kn gates. Moreover, the algorithm preserves multilinearity and runs in time polynomial in the size of F.*

Proof. Without loss of generality assume has F fanin at most 2. Consider the following simplification rules for a gate g in F.

1. Suppose g has one child. Then $g = \alpha \cdot g' + \beta$. If g' is a constant, replace g with the constant $\alpha \cdot g' + \beta$. If g' is an input variable do nothing. If g' is another gate, where $g' = \alpha' \cdot g'' + \beta'$, replace g with $(\alpha\alpha') \cdot g'' + (\alpha\beta' + \beta)$, explicitly computing the new constants.

2. Suppose g has two children. Then $g = \alpha \cdot (g_1 \text{ op } g_2) + \beta$. If both children are constant replace g with the constant it computes. If one child is constant, without loss of generality, g_1, and op $= +$, replace g with $\alpha \cdot g_2 + (\alpha g_1 + \beta)$; if op $= \times$, replace g with $(\alpha g_1) \cdot g_2 + \beta$; otherwise do nothing.

Note that the simplification rules preserve multilinearity. Repeatedly apply these rules until a fixed point is reached, say F'. Inspection of the rules gives that g is always replaced by an equivalent formula. Therefore $F' \equiv F$. Since non-constant parts of the formula are never duplicated, F' is read-k. If g is an internal gate with only one child, it is eliminated; if g is an internal gate with two children at least one of which is a constant, g is changed by these rules. Thus, if a fixed point is reached, all internal gates must have two children that are not constants.

This implies that all the original constants have been moved into the α's and β's of the gates and inputs. Therefore F' can be viewed as a binary tree where the at most kn inputs are leaves. Thus, the total number of gates in F' is at most kn. This means that there are at most $2kn$ gates and input pairs (α, β).

Since the total number of gates and constants decreases with each step, this process can repeat at most the size of F many times before the fixed point is reached. ■

The previous lemma only bounds the number of gates in the resulting formula. The constants α and β at each gate and inputs in F' are bounded in bit-length by the size of the original formula F. This means that evaluating the formula F' only incurs cost polynomial in the size of F.

The standard form can be easily generalized to sparse-substituted formulas, because the internal gate structure is the same where the sparse polynomials are treated as inputs. The standard form can also be specialized to bounded depth formulas where the gate fanin is unbounded.

3.3.1.2 Partial Derivatives

Partial derivatives of multilinear polynomials can be defined formally over any field $\mathbb{F}$ by stipulating the partial derivative of monomials consistent with standard calculus, and imposing linearity. The well-known sum, product, and chain rules then carry over. For a multilinear polynomial $P \in \mathbb{F}[x_1, ..., x_n]$ and a variable x_i, we can write P as $P = Q \cdot x_i + R$, where $Q, R \in \mathbb{F}[x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$. In this case the partial derivative of P with respect to x_i is $\frac{\partial P}{\partial x_i} = Q$. We often shorten this notation to $\partial_{x_i} P$. Observe that $R = P|_{x_i \leftarrow 0}$.

For a multilinear read-k formula F, $\partial_x F$ is easily obtained from F, and results in a formula with the same or a simpler structure than F. Start from the output gate and recurse through the formula, applying at each gate the sum or product rule as appropriate. In the case of an addition gate $g = \alpha \cdot \sum_i g_i + \beta$, we have that $\partial_x g = \alpha \cdot \sum_i \partial_x g_i$. Thus, we essentially recursively replace each of the children by their partial derivative. The structure of the formula is maintained, except that some children may disappear because they do not depend on x. In the case of a multiplication gate $g = \alpha \cdot (\prod_i g_i) + \beta$, the derivative $\partial_x g$ is a sum of products, namely $\partial_x g = \alpha \cdot \sum_i (\prod_{j \neq i} g_j) \cdot \partial_x g_i$. However, by the multilinearity condition at most one of the terms in the sum is nonzero because at most one g_i can depend on x. Thus, we leave the branches g_j for $j \neq i$ untouched, recursively replace g_i by its partial derivative, and set $\beta = 0$. The structure of the formula is again maintained or simplified. Overall, the resulting formula $\partial_x F$ is multilinear and read-k. See Figure 3.1 for an example with each $\alpha = 1$ and $\beta = 0$. Similarly, the partial derivatives of multilinear $\sum^m$-read-k formulas are multilinear $\sum^m$-read-k formulas.

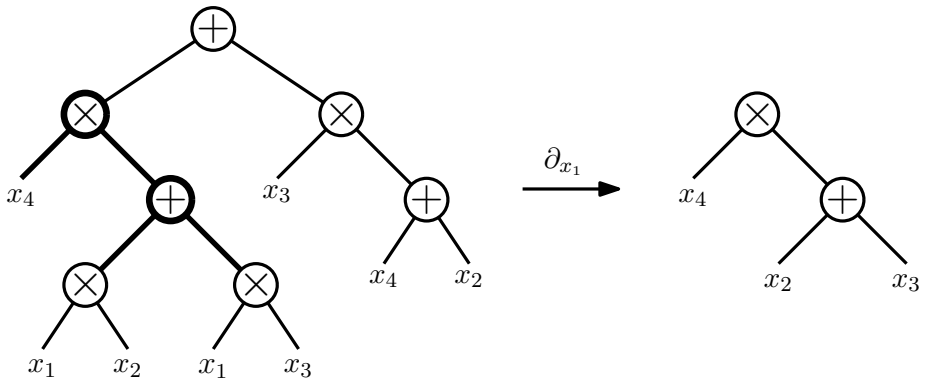


Figure 3.1: An example of taking the partial derivative of a multilinear read-2 formula.

To handle the case of structurally-multilinear formulas we extend the notion of partial derivative: $\partial_{x,\alpha} F \doteq F|_{x \leftarrow \alpha} - F|_{x \leftarrow 0}$ for some $\alpha \in \mathbb{F}$. Provided the size of $\mathbb{F}$ is more than the degree of x in the formula F, there exists some $\alpha \in \mathbb{F}$ such that $\partial_{x,\alpha} F \not\equiv 0$ iff F depends on x. For this more general definition the analogs of the sum and product rules follow for structurally-multilinear formulas. Given a structurally-multilinear formula F, $\partial_{x,\alpha} F$ can be computed by a structurally-multilinear formula with no larger size or read.

3.3.1.3 Example Separating Read-2 and $\sum$-Read-Once Formulas

In this subsection we give an example which separates multilinear read-2 formulas from $\sum$-read-once formulas to demonstrate that the identity tests of [SV08, SV09] for the latter type cannot carry over to our more general setting.

We follow an approach similar to that which [SV08, SV09] use to show "hardness of representation" results for sums of read-once formulas. Consider some multilinear read-2 polynomial H_k which is purportedly computable by the sum of less than k read-once formulas, i.e., $H_k \equiv \sum_{i=1}^{k-1} F_i$. We argue that for an appropriate choice of H_k, some combination of partial derivatives and substitutions is sufficient to zero at least one of the branches F_i while not degrading the hardness of H_k by too much. Since H stays hard we can complete the

argument by induction. In the base case H_1 is nonzero, so it requires at least one read-once formula to compute. This intuition is formalized in the following lemma.

Lemma 3.4. *For any non-trivial field $\mathbb{F}$ and each $k \in \mathcal{N}$ define*

$$H_k \doteq \prod_{i=1}^{2k-1} (x_{1,i}x_{2,i}x_{3,i} + x_{1,i} + x_{2,i} + x_{3,i}).$$

H_k is a multilinear read-2 formula which depends on $6k-3$ variables. Moreover, H_k is not computable by the sum of less than k read-once formulas.

Proof. Observe that for all $i \in \mathcal{N}$, $(x_{1,i}x_{2,i}x_{3,i} + x_{1,i} + x_{2,i} + x_{3,i})$ is a multilinear read-2 formula. Therefore for all $k \in \mathcal{N}$, H_k is a multilinear read-2 formula. We prove the second half of the claim by induction. When $k = 1$, H_1 is nonzero and hence the claim holds trivially. Now consider the induction step. Suppose the contrary: There exists a sequence of at most $k-1$ read-once formulas $\{F_i\}$ such that $H_k = \sum_{i=1}^{k-1} F_i$.

Consider F_{k-1}. Suppose there exists a pair of variables y, z such that $\partial_{y,z} F_{k-1} \equiv 0$. These operations modify at most two factors of H_k but do not zero them. Therefore $\partial_{y,z} H_k = H' \cdot H_{k-1}$ for some nonzero multilinear read-2 formula H' that depends on four variables and is variable disjoint from H_{k-1} (abusing notation to relabel the variables). Since $H' \not\equiv 0$ and multilinear, there exists $\bar{\alpha} \in \{0,1\}^4 \subseteq \mathbb{F}^4$ such that $H'(\bar{\alpha}) = c \neq 0$. This means that $\partial_{y,z} H_k|_{\mathrm{var}(H') \leftarrow \bar{\alpha}} = c \cdot H_{k-1}$. Hence H_{k-1} can be written as $\sum_{i=1}^{k-2} c^{-1} \partial_{y,z} F_i|_{\mathrm{var}(H') \leftarrow \bar{\alpha}}$, which contradicts the induction hypothesis. Therefore we can assume that for all pairs of variables y and z, $\partial_{y,z} F_{k-1} \not\equiv 0$.

This together with the read-once property of F_{k-1} implies the that least common ancestor of any pair of variables in F_{k-1} must exist and must be a multiplication gate. This also implies that F_{k-1} depends on all variables in H_k. Consider some variable y. Now, since $k > 1$ there must exist a variable z such that the least common ancestor of y and z in F_{k-1}

is the first multiplication gate above y which depends on a variable other than y. Because F_{k-1} is a read-once formula we can write $\partial_z F_{k-1} = (y - \alpha) \cdot F'_{k-1}$ for some $\alpha \in \mathbb{F}$ and a read-once formula F'_{k-1} which is independent of y and z. Therefore $(\partial_z F_{k-1})|_{y \leftarrow \alpha} \equiv 0$. By inspection we see that for all variables y, z and $\alpha \in \mathbb{F}$, $(\partial_z H_k)|_{y \leftarrow \alpha} = H' \cdot H_{k-1}$ for some nonzero multilinear read-2 formula H' which is variable disjoint from H_{k-1}. By the argument in the previous paragraph we may again conclude by contradicting the induction hypothesis. ∎

This implies the following corollary.

Corollary 3.1. *There exists a multilinear read-2 formula in n variables such that all k sums of read-once formula computing it require $k = \Omega(n)$.*

3.3.1.4 Polynomial Identity Testing and Hitting Set Generators

Arithmetic formula identity testing denotes the problem of deciding whether a given arithmetic formula is identically zero as a formal polynomial. More precisely, let F be an arithmetic formula on n variables over the field $\mathbb{F}$. The formula F is identically zero iff all coefficients of the formal polynomial that F defines vanish. For example, the formula $(x-1)(x+1) - (x^2 - 1)$ is identically zero but the formula $x^2 - x$ is nonzero (even over the field with 2 elements).

There are two general paradigms for identity testing algorithms: *blackbox* and *non-blackbox*. In the non-blackbox setting, the algorithm is given the description of the arithmetic formula as input. In the blackbox setting, the algorithm is allowed only to make queries to an oracle that evaluates the formula on a given input. Observe that non-blackbox identity testing reduces to blackbox identity testing because the description of a formula can be used to efficiently evaluate the formula on each query the blackbox algorithm makes. There is one caveat – in the blackbox case the algorithm should be allowed to query inputs from a

sufficiently large field. This may be an extension field if the base field is too small. Otherwise, it is impossible to distinguish a polynomial that is functionally zero over $\mathbb{F}$ but not zero as a formal polynomial, from the formal zero polynomial (e.g., consider the formula $x^2 - x$ restricted to the field with two elements).

Blackbox algorithms for a class $\mathcal{P}$ of polynomials naturally produce a *hitting set*, i.e., a set H of points such that each nonzero polynomial $P \in \mathcal{P}$ from the class does not vanish at some point in H. In this case we say that H *hits* the class $\mathcal{P}$, and each P in particular. To see the connection, observe that when a blackbox algorithm queries a point that is nonzero it can immediately stop. Conversely, when the result of every query is zero, the algorithm must conclude that the polynomial is zero; otherwise, it fails to correctly decide the zero polynomial.

A related notion is that of a *hitting set generator*. Formally, a polynomial map $\mathcal{G} = (\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n)$ where each $\mathcal{G}_i \in \mathbb{F}[y_1, y_2, \ldots, y_\ell]$ is a hitting set generator (or generator for short) for a class $\mathcal{P}$ of polynomials, if for each nonzero polynomial $P \in \mathcal{P}$, $\mathcal{G}$ *hits* P, that is the composition of P with $\mathcal{G}$ (often denoted $P \circ \mathcal{G}$ or $P(\mathcal{G})$) is nonzero. Suppose that $\mathcal{G}$ hits a class of polynomials $\mathcal{P}$, then $\mathcal{G}$ can be used to construct a blackbox identity test for $P \in \mathcal{P}$ by collecting all elements in the image of $\mathcal{G}$ when we let the input variables to $\mathcal{G}$ range over some small set.

Proposition 3.2. *Let $\mathcal{P}$ be a class of n-variate polynomials of total degree at most d. Let $\mathcal{G} \in (\mathbb{F}[y_1, ..., y_\ell])^n$ be a generator for $\mathcal{P}$ such that the total degree of each polynomial in $\mathcal{G}$ is at most $d_{\mathcal{G}}$. There is a deterministic blackbox polynomial identity testing algorithm for $\mathcal{P}$ that runs in time $O((d \cdot d_{\mathcal{G}})^\ell)$ and queries points from an extension field of size $O(d \cdot d_{\mathcal{G}})$.*

Proof. Let P be a nonzero polynomial in $\mathcal{P}$. Since $\mathcal{G}$ is a generator for $\mathcal{P}$, the polynomial $P \circ \mathcal{G} \in \mathbb{F}[y_1, y_2, \ldots, y_\ell]$ is nonzero. The total degree of $P \circ \mathcal{G}$ is at most $d \cdot d_{\mathcal{G}}$. By the Schwartz-Zippel Lemma [Sch80, Zip79, DL78] any set $V^\ell \subseteq \mathbb{E}^\ell$, where $|V| \geq d \cdot d_{\mathcal{G}} + 1$, and

$\mathbb{E}$ is an extension field of $\mathbb{F}$, contains a point at which $P \circ \mathcal{G}$ does not vanish. Note that the extension field $\mathbb{E} \supseteq \mathbb{F}$ must be sufficiently large to support the subset V of the required size. The algorithm tests P at all points in $\mathcal{G}(V^\ell)$ and outputs zero iff all test points are zero.■

Note that this approach is only efficient when $\ell \ll n$ and the degrees are not too large.

Hitting set generators and a hitting sets are closely related. By Proposition 3.2 a hitting set generator implies a hitting set. It is also known that a hitting set generator can be efficiently constructed from a hitting set using polynomial interpolation [SV09].

3.3.2 SV-Generator

One example of such a generator is the one Shpilka and Volkovich obtained by interpolating the set H_w^n of all points in $\{0,1\}^n$ with at most w nonzero components. The resulting generator G_w is a polynomial map of total degree n on $2w$ variables. Shpilka and Volkovich [SV09] showed that it hits $\sum^k$-read-once formulas for $w \geq 3k + \log n$. Karnin et al. [KMSV10] also used it to construct a hitting set generator for multilinear depth-four formulas with bounded top fanin.

For completeness, we include the definition of the generator G_w.

Definition 3.4 (SV-Generator [SV09]). *Let $a_1, \ldots, a_n$ denote n distinct elements from a field $\mathbb{F}$, and for $i \in [n]$ let $L_i(x) \doteq \prod_{j \neq i} \frac{x - a_j}{a_i - a_j}$ denote the corresponding Lagrange interpolants. For every $w \geq 1$, define*

$$G_w(y_1, ..., y_w, z_1, ..., z_w) \doteq \left(\sum_{j=1}^{w} L_1(y_j) z_j, \sum_{j=1}^{w} L_2(y_j) z_j, ..., \sum_{j=1}^{w} L_n(y_j) z_j \right).$$

Let $(G_w)_i$ denote the i^{th} component of G_w.

For intuition it is helpful to view the action of G_1 on a random element of $\mathbb{F}^2$ as selecting a

random variable (via the value of y_1[4]) and a random value for that variable (via the value of z_1). Since the SV-generator is a polynomial map it is natural to define the sum of two copies of the generator by their component-wise sum and to furthermore view G_w as the sum of w independent choices of variables and values. For this reason and convenience of notation we take the convention that whenever an instance of the SV-generator G_w is added to another polynomial map $\mathcal{G}$ (including to another instance of the SV-generator) we will assume that G_w is defined over a new set of variables, that is, $\mathrm{var}(G_w) \cap \mathrm{var}(\mathcal{G}) = \emptyset$. With this convention in mind, the SV-generator has a number of useful properties that follow immediately from its definition. We list the ones we use.

Proposition 3.3 ([KMSV10, Observations 4.2, 4.3]). *Let w_1, w_2 be positive integers.*

1. $G_{w_1}(\bar{y}, \bar{0}) \equiv \bar{0}$.
2. $G_{w_1}|_{y_{w_1} \leftarrow a_i} = G_{w_1 - 1} + z_{w_1} \cdot \bar{e}_i$, *where $\bar{e}_i$ is the 0-1-vector with a single 1 in position i.*
3. $G_{w_1} + G_{w_2} = G_{w_1 + w_2}$.

The first item states that zero is in the image of G. The second item shows how to make a single output component (and no others) depend on a particular z_j. The final item shows that sums of independent copies of G are equivalent to a single copy of G with the appropriate parameter w (note that the seed variables to the generator can be reindexed without loss of generality). Proposition 3.3 implies the following.

Proposition 3.4. *Let $P = \sum_{i=0}^{d} P_i x_k^i$, where each $P_i \in \mathbb{F}[x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n]$ are polynomials independent of the variable x_k. Suppose the polynomial map $\mathcal{G}$ hits P_j for some $j > 0$, then $P(\mathcal{G} + G_1)$ is non-constant.*

[4]Note: This is not completely accurate because for values outside the a_i's the generator does not uniquely select a variable.

Proof. Consider $\mathcal{G} + G_1$. Without loss of generality, the variables in G_1 are y_1 and z_1, and by convention are disjoint from the variables in $\mathcal{G}$. Set the seed variable y_1 to the constant a_k. By Proposition 3.3, Part 2, $G_1(y_1, z_1)|_{y_1 \leftarrow a_k} = z_1 \cdot \bar{e}_k$.

Now, consider P composed with $\mathcal{G} + G_1(a_k, z_1)$ and write:

$$P(\mathcal{G} + G_1(a_k, z_1)) = \sum_{i=0}^{d} P_i(\mathcal{G})((\mathcal{G})_k + z_1)^i.$$

By hypothesis $P_j(\mathcal{G}) \not\equiv 0$ for some $j > 0$, fix j to be the maximum such index. Since $\mathcal{G}$ is independent of z_1, $P_j(\mathcal{G}) \not\equiv 0$, and j is maximal: $P(\mathcal{G} + G_1(a_k, z_1))$ has a monomial which depends on z_1^j that cannot be canceled. Therefore $P(\mathcal{G} + G_1(a_k, z_1))$ is non-constant and hence $P(\mathcal{G} + G_1)$ is as well. ∎

This proposition implies the following lemma: If $\mathcal{G}$ is a generator that hits some partial derivative of a polynomial, then $\mathcal{G} + G_1$ hits the polynomial itself.

Lemma 3.5. *Let P be a polynomial, x be a variable, and $\alpha \in \mathbb{F}$. If $\mathcal{G}$ hits a nonzero $\partial_{x,\alpha} P$, then $P(\mathcal{G} + G_1)$ is non-constant.*

Proof. Write P as a univariate polynomial in x:

$$P = \sum_{i=0}^{d} P_i x^i,$$

where the polynomials P_i do not depend on x. By definition

$$\partial_{x,\alpha} P = P|_{x \leftarrow \alpha} - P|_{x \leftarrow 0} = \sum_{i=1}^{d} P_i \alpha^i.$$

Our hypothesis $(\partial_{x,\alpha} P)(\mathcal{G}) \not\equiv 0$ then implies that there is a $j > 0$ such that $P_j(\mathcal{G}) \not\equiv 0$. Applying Proposition 3.4 completes the proof. ∎

We now use this lemma to argue that the SV-generator hits sparse polynomials. Consider a sparse polynomial F. For any variable x that does not divide F, either at least half of the terms of the sparse-substituted polynomial depend on x, or at least half of the terms the do not. In the former situation setting x to zero eliminates at least half of the terms in F; in the latter situation taking the partial derivative with respect to x has the same effect. Combining this with Lemma 3.5 and the properties of G completes the argument.

Lemma 3.6. *Let F be a non-constant sparse polynomial with t terms, then $F(G_{\lceil \log t \rceil + 1})$ is non-constant.*

Proof. Assume, without loss of generality, that there are no duplicate monomials present in F. We proceed by induction on t.

Suppose $t = 1$. By hypothesis F consists of a single non-constant monomial. Because the components G_1 are non-constant we can conclude that $F(G_1)$ is non-constant.

Now consider the induction step for $t > 1$. Let $w \doteq \lceil \log \frac{t}{2} \rceil + 1$.

Case 1: Suppose there exists a variable $x_i \in \mathrm{var}(F)$ such that at most half of the terms depend on x_i. Then there is an $\alpha \in \bar{\mathbb{F}}$ such that $\partial_{x_i,\alpha} F \not\equiv 0$ and has at most $\frac{t}{2}$ terms. By induction $(\partial_{x_i,\alpha} F)(G_w) \not\equiv 0$. By Lemma 3.5, $F(G_w + G_1)$ is non-constant. Applying Proposition 3.3, Part 3, completes the case.

Case 2: Otherwise, for each variable $x_i \in \mathrm{var}(F)$ more than half the terms in F depend on x_i. There are two cases.

1. Suppose there exists a variable $x_i \in \mathrm{var}(F)$ such that $F|_{x_i \leftarrow 0}$ is non-constant. Consider $F(G_w + G_1(y_{w+1}, z_{w+1}))$. Set $y_{w+1} \leftarrow a_i$ and $z_{w+1} \leftarrow -(G_w)_i$. Write $F = F_{x_i} \cdot x_i + F|_{x_i \leftarrow 0}$, for some sparse polynomial F_{x_i} which may depend on x_i. By Proposition 3.3,

Part 2:

$$\begin{aligned} F(G_w + G_1(a_i, -(G_w)_i)) =& F|_{x_i \leftarrow 0}(G_w) \\ &+ F_{x_i}(G_w + G_1(a_i, -(G_w)_i)) \cdot \underbrace{((G_w)_i - (G_w)_i)}_{\equiv 0} \\ =& F|_{x_i \leftarrow 0}(G_w). \end{aligned}$$

By induction, the RHS of the above equation is non-constant, and hence $F(G_w + G_1)$ is non-constant.

2. Otherwise, for all $x_i \in \text{var}(F)$, $F|_{x_i \leftarrow 0}$ is a constant. We can assume without loss of generality that F is not divisible by any variable because such a variable can be factored out and independently hit by G_{w+1}. Therefore, for each $x_i \in \text{var}(F)$ at least one term of F does not depend on x_i. Combining this fact with the hypothesis of the case implies, without loss of generality, that F has a nonzero constant term c. We can write $F = F' + c$ for a non-constant sparse polynomial F' with $t - 1$ terms. By induction $F'(G_{w+1})$ is non-constant. Hence $F(G_{w+1})$ is non-constant.

This completes the proof. ∎

Before we argue the last necessary property of G, we state one additional definition.

Definition 3.5. *For $\ell \in \mathbb{N}$, let $\mathcal{D}_\ell$ denote the class of nonzero polynomials that are divisible by a multilinear monomial on ℓ variables, i.e., the product of ℓ distinct variables. We use M_ℓ to denote the monomial $\prod_{i=1}^{\ell} x_i$.*

We require a property of G that is implicit in [SV08, SV09], namely Fact 3.1 from the introduction. Informally it states: If a class of polynomials is disjoint from $\mathcal{D}_\ell$, and is closed under zero substitution, then the SV-generator hits this class of polynomials.

Lemma 3.7 (Implicit in [SV09, Theorem 6.2]). *Let $\mathcal{P}$ be a class of polynomials that is closed under zero-substitutions. If $\mathcal{P}$ is disjoint from $\mathcal{D}_\ell$ for every $\ell > w$, the map G_w is a hitting set generator for $\mathcal{P}$.*

Proof. Fix $P \in \mathbb{F}[x_1, \ldots, x_n]$ in $\mathcal{P}$, and let d denote the maximum degree of individual variables in P. Let $S \subseteq \bar{\mathbb{F}}$ with $|S| = d + 1$ and $0 \in S$. Define the set H_w^n to be the set of vectors in $S^n \subseteq \mathbb{F}^n$ with at most w nonzero components. The set H_w^n is in the image of G_w. To see this, consider $\bar{x} \in H_w^n$. Let $\{c_i\}_{i=1}^r$ be the index set of at most w nonzero components of $\bar{x}$. We can set the seed to G_w so that G_w evaluates to $\bar{x}$: For $i \in [r]$, set $y_i \leftarrow a_{c_i}$ (that is, the constant selecting the c_i^{th} component) and $z_i \leftarrow x_i$; set $y_i = z_i \leftarrow 0$ for $i > r$. Then $G_w(\bar{y}, \bar{z}) = \bar{x}$.

Since the image of G_w contains H_w^n, it is sufficient to prove that $P|_{H_w^n} \equiv 0$ implies $P \equiv 0$. For the given value of d, we prove the latter statement by induction on n. If $n \leq w$, then H_w^n is all of S^n. Since P has individual degree at most d, there is point in S^n which witnesses the non-zeroness of P. Therefore, $P|_{H_w^n} \equiv 0$ implies $P \equiv 0$, completing the base case.

Now, consider $n > w$ and suppose that $P|_{H_w^n} \equiv 0$. For some $j \in [n]$, let $P' \doteq P|_{x_j \leftarrow 0}$. By the closure under zero-substitutions of $\mathcal{P}$, $P' \in \mathcal{P}$. Since H_w^{n-1} is a projection of $H_w^n \cap \{\bar{x} \in S^n | x_j = 0\}$, we have that $P'|_{H_w^{n-1}} \equiv 0$. The individual degree of P' is at most d, and P' depends on at most $n - 1$ variables. By the induction hypothesis $P|_{x_j \leftarrow 0} = P' \equiv 0$ and therefore $x_j | P$. The above argument works for any $j \in [n]$, so $x_j | P$ for all $j \in [n]$. Hence, $(\prod_{i=1}^n x_i) | P$. We have that $P = Q \cdot \prod_{i=1}^n x_i$ for some polynomial Q. Since $P \in \mathcal{P}$ and $\mathcal{P} \cap \mathcal{D}_n = \emptyset$ for $n > w$, we conclude that $Q \equiv 0$. Thus $P \equiv 0$, completing the proof. ∎

3.4 Structural Witnesses

Structural witnesses allow us to determine that certain formulas are nonzero without exhibiting a point where they do not vanish. The notion was introduced for depth-three

formulas in [DS07] and later also applied to multilinear depth-four formulas in [KMSV10, SV11].

For their application to multilinear depth-four formulas Karnin et al. [KMSV10] consider multilinear formulas of the form $F = \sum_{i=1}^{m} F_i$ where the F_i's factor into subformulas each depending only on a fraction α of the variables. In such a case we call the formula F α-split. For technical reasons we present a more general definition that requires "splitness" with respect to a restricted set of variables.

Definition 3.6 (α-split). *Let $F = \sum_{i=1}^{m} F_i \in \mathbb{F}[x_1, \ldots, x_n]$, $\alpha \in [0,1]$, and $V \subseteq [n]$. We say that F is* α-split *if each F_i is of the form $\prod_j F_{i,j}$ where $|var(F_{i,j})| \leq \alpha n$. F is* α-split with respect to V *(in shorthand, α-split$_V$) if $|var(F_{i,j}) \cap V| \leq \alpha |V|$ for all i, j.*

For $V = [n]$, the two definitions coincide. Note in the definition of split we do not require that $\mathrm{var}(F) = [n]$.

To state the structural result Karnin et al. use, we also need the following terminology. An additive top-fanin-m formula $F = \sum_{i=1}^{m} F_i$ is said to be *simple* if the greatest common divisor (gcd) of the F_i's is in $\mathbb{F}$. F is said to be *minimal* if for all non-trivial subsets $S \subsetneq [m]$, $\sum_{i \in S} F_i \not\equiv 0$. The following formalization quantifies Fact 3.3 from the introduction.

Lemma 3.8 (Structural Witness for Split Multilinear Formulas).
For $R(m) = (m-1)^2$ the following holds for any multilinear formula $F = \sum_{i=1}^{m} F_i$ on $n \geq 1$ variables with $\cup_{i \in [m]} var(F_i) = [n]$. If F is simple, minimal, and α-split for $\alpha = (R(m))^{-1}$, then $F \not\equiv 0$.

Although not critical for our results, we point out that Lemma 3.8 shaves off a logarithmic factor in the bound for $R(m)$ obtained by Karnin et al. They show how to transform a split, simple and minimal, multilinear formula $F = \sum_{i=1}^{m} F_i$ into a simple and minimal depth-three formula $F' = \sum_{i=1}^{m} F'_i$, and then apply the so-called rank bound [DS07, SS09, SS10a] to F'

in order to show that $F \not\equiv 0$. We follow the same outline, but use a new structural witness for the special type of multilinear depth-three formulas F' that arise in the proof, rather than the rank bound Karnin et al. use.

The special type of multilinear depth-three formulas we consider is of the form $F = \sum_{i=1}^{m} F_i$ where each F_i is the product of univariate linear polynomials (of the form $\alpha \cdot x + \beta$) on distinct variables. Along the lines of [SV11], we show that a simple and minimal formula of that form where at least one F_i depends on more than $m-2$ variables, is nonzero.

Lemma 3.9 (Structural Witness for Univariate Multilinear Depth-3 Formulas). *Let $m \geq 2$ and $F = \sum_{i=1}^{m} F_i = \sum_{i=1}^{m} \prod_{j=1}^{d_i} L_{ij}$ be a multilinear depth-three formula where each L_{ij} is a univariate polynomial. If F is simple and minimal, and $|var(F_i)| > m-2$ for some $i \in [m]$, then $F \not\equiv 0$.*

We provide the proof of Lemma 3.9 in Section 3.4. For completeness, we now show how it implies the structural witness that we use in our identity test.

Proof (of Lemma 3.8). Without loss of generality write: $F = \sum_{i=1}^{m} F_i = \sum_{i=1}^{m} \prod_{j=1}^{d_i} P_{ij}$, where the P_{ij} are irreducible. We can construct a set $U \subseteq [n]$ such that for all i, j, $|U \cap \mathrm{var}(P_{ij})| \leq 1$ and there exists $\ell \in [m]$ for which $|U \cap \mathrm{var}(F_\ell)| \geq \frac{1}{\alpha \cdot m} > m-2$.

Construct U greedily as follows. Begin with U empty. While there is a variable x such that all the P_{ij}'s that depend on x depend on no variables currently in U, add x to U. Each added variable x excludes at most $(\alpha n) \cdot b_x$ variables from consideration, where b_x is the number of branches of F that depend on x. This procedure can continue as long as $\sum_{x \in U} \alpha n b_x < n$. This implies that we can achieve $b \doteq \sum_{x \in U} b_x \geq \frac{1}{\alpha}$. Observe that we may also write $b = \sum_{i=1}^{m} |U \cap \mathrm{var}(F_i)|$. By averaging we see that there exists an $\ell \in [m]$ such that $|U \cap \mathrm{var}(F_\ell)| \geq \frac{1}{\alpha \cdot m} > m-2$, as claimed.

Assigning all variables outside of U linearizes each P_{ij} – in fact, each P_{ij} becomes a univariate linear function – and turns F into a depth-three formula F' with an addition

gate on top of fanin m. Moreover, as we will argue, a typical assignment $\bar{\beta}$ from $\bar{\mathbb{F}}$ to the variables outside of U keeps F' (1) simple, (2) minimal, and (3) ensures that $|\mathrm{var}(F'_\ell)|$ is more than $m-2$. The structural witness for univariate multilinear depth-three formulas (Lemma 3.9) implies that $F' \not\equiv 0$, and therefore that $F \not\equiv 0$.

All that remains is to establish the above claims about a typical assignment $\bar{\beta}$ to the variables in $[n] \setminus U$:

1. To argue simplicity, we make use of the following property of multilinear polynomials P and Q: If P is irreducible and depends on a variable x, then $P|Q$ iff $P|_{x\leftarrow 0} \cdot Q - P \cdot Q|_{x\leftarrow 0} \equiv 0$.[5] Since F is simple, for every irreducible subformula P_{ij} that depends on some $u \in U$, there is branch, say $F_{i'}$, such that P_{ij} does not divide $F_{i'}$. Thus, by the above property, $P_{ij}|_{U\leftarrow 0} \cdot F_{i'} - P_{ij} \cdot F_{i'}|_{U\leftarrow 0} \not\equiv 0$. Let P'_{ij} be the result of applying $\bar{\beta}$ to P_{ij}, and define $F'_{i'}$ and F' similarly. A typical assignment $\bar{\beta}$ keeps $P_{ij}|_{U\leftarrow 0} \cdot F_{i'} - P_{ij} \cdot F_{i'}|_{U\leftarrow 0}$ nonzero and P'_{ij} dependent on u. Since P'_{ij} remains irreducible as a univariate polynomial, the above property implies that P'_{ij} does not divide $F'_{i'}$. Therefore, F' is simple.

2. Minimality is maintained by a typical assignment since if $\sum_{i\in S} F_i$ is a nonzero polynomial for all $\emptyset \subsetneq S \subsetneq [m]$, then the same holds after a typical partial assignment $\bar{\beta}$.

3. Finally, for any $u \in U$ there exists at least one P_{ij} for which $u \in \mathrm{var}(P_{ij})$. Since a typical assignment to the variables in P_{ij} other than u turns P_{ij} into a non-constant linear function of u and $|U \cap \mathrm{var}(F_\ell)| > m-2$, we conclude that F'_ℓ depends on more than $m-2$ variables under a typical assignment $\bar{\beta}$. ∎

[5]Here is a short proof of this property: If $P|Q$, then $Q = P \cdot Q'$ where Q' does not depend on x by multilinearity and hence $P|_{x\leftarrow 0} \cdot Q - P \cdot Q|_{x\leftarrow 0} \equiv P|_{x\leftarrow 0} \cdot (P \cdot Q') - P \cdot (P|_{x\leftarrow 0} \cdot Q') \equiv 0$. If P does not divide Q, then because P is irreducible and depends on x, P does not divide $P|_{x\leftarrow 0} \cdot Q$, and we conclude the required identity cannot hold.

Proof of Lemma 3.9

We now return to the proof of Lemma 3.9, whose outline goes as follows.

We argue the contra-positive of the lemma, that is, if F is a formula of the stated form from Lemma 3.9, and simple, minimal, and *zero*, then $|\mathrm{var}(F_\ell)| \leq m-2$ for all $\ell \in [m]$. We proceed by induction on m. Consider such a simple, minimal, and zero formula F. Suppose that some branch, say F_m, depends on more than $m-2$ variables. Since F is simple, if $(x-\alpha)$ divides F_m, it does not divide all other F_ℓ's. We may therefore assume it does not divide F_1. Setting $x \leftarrow \alpha$ zeroes the branch F_m, but not F_1. The formula $F' \doteq F|_{x\leftarrow\alpha}$ may not be simple or minimal. Consider a minimal zero subformula of F', called F'', containing $F_1|_{x\leftarrow\alpha}$. We argue that the size of $\mathrm{var}(F_1)$ can be bounded by a combination of the number of variables in: (i) $\frac{F_1|_{x\leftarrow\alpha}}{\gcd(F'')}$, and (ii) the gcd of the branches of F''. The former may be bounded by induction. To bound the latter we argue a bound on the number of variables in the gcd of a non-trivial subformula of F (with fanin at least 2). This is done by including the gcd as a summand in a formula with fanin less than m. Since the gcd is a product of univariate polynomials, we use induction to derive the stated bound on $|\mathrm{var}(F_1)|$.

The above argument can be repeated for each F_ℓ with $\ell \in [m-1]$ to get the bound the lemma states for those branches. Since all branches except F_m depend on few variables, there exists a set of variables $V \subseteq \mathrm{var}(F_m)$ that is not fully contained in any other branch F_ℓ. This implies that the partial derivative of F with respect to V zeroes all branches except F_m. This in turn, means that $\partial_V F \equiv \partial_V F_m \not\equiv 0$, contradicting the fact that $F \equiv 0$, and completing the proof.

When $m = 2$ the simplicity and zeroness of F implies that neither branch depends on any variables. In the induction step at least one branch is eliminated at the cost of at most one variable. This intuitively implies the bound of $m-2$. We now formalize the above outline.

Proof (of Lemma 3.9). The proof is by induction on m. If $m = 2$ then F can only be a sum of two constants. Hence, the base case holds. Assume $m \geq 3$. We need the following proposition.

Proposition 3.5. *Let $m \geq 2$ and $F = \sum_{i=1}^{m} F_i = \sum_{i=1}^{m} \prod_{j=1}^{d_i} L_{ij}$ be a multilinear depth-three formula where each L_{ij} is a univariate polynomial, let $H \doteq \gcd(F_1, F_2, \ldots, F_t)$ for some $2 \leq t \leq m-1$. If F is simple and minimal, then $|var(H)| \leq m - t - 1$.*

Proof. Denote $V = \mathrm{var}(H)$ and $F_i = H \cdot f_i$ for $i \in [t]$. We can write

$$F = H \cdot (f_1 + f_2 + \ldots + f_t) + F_{t+1} + \ldots + F_m.$$

As F is multilinear H must be variable disjoint with all f_i's. As in the proof of Lemma 3.8, we can fix the variables outside V such that the resulting formula $F' = H \cdot \alpha + F'_{t+1} + \ldots + F'_m$ remains simple and minimal, and $\alpha \neq 0$. A typical assignment will suffice. We obtain a formula satisfying the conditions of Lemma 3.9 with top fanin $m' = m - t + 1$. Note that $2 \leq m' \leq m - 1$. Therefore, we can apply the induction hypothesis to get that $|\mathrm{var}(H)| \leq m' - 2 = m - t - 1$. ∎

We now return to the proof of the lemma.

Suppose that the lemma does not hold. Then there is some branch that depends on more than $m-2$ variables. Therefore, we assume without loss of generality that $|\mathrm{var}(F_m)| > m-2$. Now consider some other branch, say F_1. Since F is simple and minimal, by Proposition 3.5, $|\mathrm{var}(\gcd(F_1, F_m))| \leq m - 3$. Therefore there is a univariate factor of F_m which does not divide F_1. Let this factor be $(x - \alpha)$.

Consider $F' \doteq F|_{x \leftarrow \alpha}$, and observe that $F_m|_{x \leftarrow \alpha} \equiv 0$, but $F_1|_{x \leftarrow \alpha} \not\equiv 0$. By assumption F' is a multilinear formula computing the zero polynomial. Let F'' be a minimal zero subformula of F' that contains the summand $F_1|_{x \leftarrow \alpha}$. Assume without loss of generality

that

$$F'' \doteq \sum_{i=1}^{t} F_i'' = \sum_{i=1}^{t} F_i|_{x \leftarrow \alpha},$$

for some t. From the above discussion $2 \leq t \leq m-1$. We would to like to upper bound $|\mathrm{var}(F_1'')|$. By multilinearity we may write

$$|\mathrm{var}(F_1'')| = \left|\mathrm{var}\left(\frac{F_1''}{\gcd(F_1'', \cdots, F_t'')}\right)\right| + |\mathrm{var}(\gcd(F''))|\,.$$

We bound these two terms in turn. Consider the first term. This formula $\frac{F''}{\gcd(F'')}$ is simple by construction; in addition, it is minimal because F'' is minimal. Consequently, by the induction hypothesis on $\frac{F''}{\gcd(F'')}$ we get that: $\left|\mathrm{var}\left(\frac{F_1''}{\gcd(F'')}\right)\right| \leq t-2$.

Now consider the second term. We have that

$$|\mathrm{var}(\gcd(F''))| \leq |\mathrm{var}\left(\gcd(F_1, F_2, \ldots, F_t)\right)| \leq m-t-1,$$

where the former inequality follows because the F_i are multilinear products of univariate polynomials, and the latter inequality follows by the minimality of F, and Proposition 3.5 applied to F with t. By putting everything together we have that:

$$\begin{aligned}|\mathrm{var}(F_1)| \leq 1 + |\mathrm{var}(F_1'')| &\leq 1 + \left|\mathrm{var}\left(\frac{F_1''}{\gcd(F'')}\right)\right| + |\mathrm{var}(\gcd(F''))| \\ &\leq 1 + (t-2) + (m-t-1) = m-2.\end{aligned}$$

■

We have concluded that $|\mathrm{var}(F_1)| \leq m-2$. Moreover, since the above argument is generic, the bound applies to all F_ℓ, for $\ell \in [m-1]$. Thus, we have $|\mathrm{var}(F_\ell)| \leq m-2$, for $\ell \in [m-1]$.

Form a set V of variables in $\mathrm{var}(F_m)$ by selecting for each $\ell \in [m-1]$ a variable that occurs in $\mathrm{var}(F_m)$ but not in $\mathrm{var}(F_\ell)$. This is possible because F is multilinear and F_m depends on more variables than any other branch. Because F_m is a multilinear product

of linear univariate polynomials, $\partial_V F_m \not\equiv 0$. However, $\partial_V F_\ell \equiv 0$ for all $\ell \in [m-1]$. From this we conclude that $\partial_V F \equiv \partial_V F_m \not\equiv 0$, contradicting the fact that $F \equiv 0$. Therefore $\mathrm{var}(F_m) \leq m-2$ and the proof is complete.

3.5 Fragmenting and Shattering Formulas

In this section we describe a means of splitting up or *fragmenting* multilinear sparse-substituted formulas. We build up towards this goal by first fragmenting read-once formulas, and then multilinear read-k formulas. We conclude by extending our fragmentation technique to work for sparse-substituted formulas, proving our Fragmentation Lemma (Lemma 3.12).

We view the Fragmentation Lemma as an atomic operation that breaks a read-k formula into a product of easier formulas, at the cost of a few partial derivatives and zero-substitutions (the more such operations performed the longer the seed length of the eventual generator). By greedily applying the Fragmentation Lemma and using some other ideas we are able to *shatter* multilinear sparse-substituted $\sum^m$-read-k formulas, that is, simultaneously split all the top-level branches so that they are the product of factors that each only depend on a fraction of the variables. The Shattering Lemma (Lemma 3.14) is the main result of this section and formalizes Lemma 3.3 from the introduction.

3.5.1 Fragmenting Read-Once Formulas

Let F be a read-once formula. Consider a traversal of the variables of F in leaf order. For the median variable x in this traversal, $\partial_x F$ is $\frac{1}{2}$-split. This is because the path from the root of F to x partitions the formula into halves that can only combine at the top multiplication gate of $\partial_x F$. Moreover, by only considering variables on which F depends, we can make sure that $\partial_x F \not\equiv 0$ assuming F is non-constant. We make this intuition more formal in the

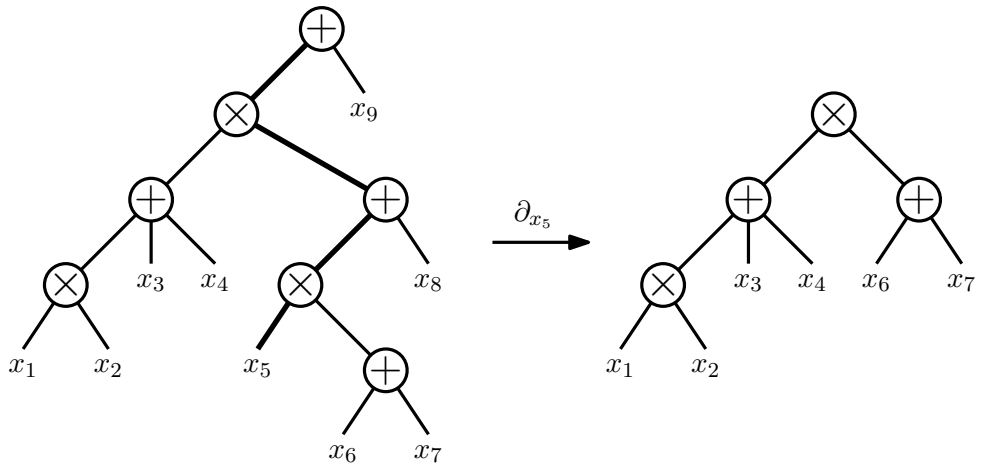


Figure 3.2: An example of fragmenting a read-once formula.

following lemma and refer to Figure 3.2 for an example. For generality, we state the lemma with respect to a restricted variable set $V \subseteq [n]$. Recall that a formula F is α-split$_V$ if it is the product of formulas each depending on fewer than $\alpha|V|$ variables from V (and possibly more variables outside of V).

Lemma 3.10 (Fragmenting Read-Once Formulas). *Let $\emptyset \subsetneq V \subseteq [n]$ and let F be an n-variable multilinear read$_V$-once formula that depends on at least one variable in V. There exists a variable $x \in V$ such that $\partial_x F$ is nonzero and is the product of subformulas of F that each depend on at most $\frac{|V|}{2}$ variables from V (and possibly more variables outside of V).*

Proof. Without loss of generality, it can be assumed that V only contains variables on which F depends. Let S be a sequence of the variables in V produced by an in-order traversal of F. Select the variable x, which has no more than $|V|/2$ of the variables in V to the left or right of it in the traversal S. Then $\partial_x F$ is nonzero (since F depends on x) and can be written in the required form. To see the latter, follow the procedure for computing $\partial_x F$ described in Section 3.3.1.2 by tracing the path in F from the root to the leaf labeled x. By the sum rule, ∂_x eliminates the diverging addition branches along this path as those

branches to not depend on x. Multiplicative branches that do not depend on x are split off as factors. None of those factors of $\partial_x F$ can depend on more than $|V|/2$ variables from V. This is because the variables within each factor cannot span both sides of the formula around the path from the output gate to x. ■

3.5.2 Fragmenting Multilinear Read-k Formulas

While illustrating the basic idea of fragmenting, Lemma 3.10 is insufficient for our purposes. A key reason the proof of Lemma 3.10 goes through is that in read-once formulas each addition gate has children that are variable disjoint. This property allows the argument to recurse into a single addition branch. In read-k formulas this is no longer the case. Our solution is to follow the largest branch that depends on a variable that is only present within that branch. This allows us to mimic the behavior of the read-once approach as long as such a branch exists. Once no such branch exists, each child of the current gate cannot contain all the occurrences of any variable x. This means that these children are read-$(k-1)$ formulas. Taking a partial derivative with respect to a variable that only occurs within the current gate eliminates all diverging addition branches above the gate. This makes the resulting formula multiplicative in all the unvisited (and small) multiplication branches. This intuition can be formalized in the following lemma, which generalizes Lemma 3.1 from the introduction.

Lemma 3.11 (Fragmenting Multilinear Read-k Formulas). *Let $\emptyset \subsetneq V \subseteq [n]$, $k \geq 2$, and let F be an n-variable multilinear read$_V$-k formula that depends on at least one variable in V. There exists a variable $x \in V$ such that $\partial_x F$ is nonzero and is the product of*

1. *subformulas of F each depending on at most $\frac{|V|}{2}$ variables from V (and possibly more variables outside of V), and*
2. *at most one $\sum^2$-read$_V$-$(k-1)$ formula, which is the derivative with respect to x of some subformula of F.*

Proof. Assume without loss of generality that V only contains variables on which F depends, and that the children of multiplication gates are variable disjoint with respect to V.

If none of the variables in V occur k times in F, any choice of variable $x \in V$ does the job. So, let us assume that is at least one variable in V occurs k times. We use Algorithm 1 to select the variable x, where we assume without loss of generality that F has fanin two.

The algorithm recurses through the structure of F, maintaining the following invariant: The current gate g being visited contains below it k occurrences of some variable in V. Setting g to be the output gate of F satisfies this invariant initially.

If g is a multiplication gate (steps (1)-(3)), recurse to the child of g that depends on more than $\frac{|V|}{2}$ of the variables in V and contains k occurrences of some variable in V. If no such child exists, return a variable from V that occurs k times in g. Such a variable must exist by the invariant.

If g is an addition gate (steps (4)-(6)), and at least one of its children, g_i, has a variable in V that occurs k times in g_i, recurse to g_i. Otherwise, both children of g are read$_V$-$(k-1)$ formulas. Select a variable $x \in V$ that occurs k times in g ending the recursion.

In the partial derivative $\partial_x F$, all unvisited addition branches along the path from the output gate of F to the final g have been eliminated. Also, all unvisited multiplication branches along the path become factors of $\partial_x F$ together with $\partial_x g$. More formally, $\partial_x F =$

$(\partial_x g) \cdot \prod_i F_i$, where the F_i are the unvisited multiplication branches. The F_i are read$_V$-k formulas that depend on at most $\frac{|V|}{2}$ variables from V, because the largest branch is always taken at multiplication gates. When the process stops at a multiplication gate, no large child contains all occurrences of some variable in V. In this case, there is at most one factor of $\partial_x g$ that depends on more than $\frac{|V|}{2}$ variables from V. If this factor exists it is a read$_V$-$(k-1)$ formula (and thus also a $\sum^2$-read$_V$-$(k-1)$ formula). The remaining factors are read$_V$-k formulas depending on at most $\frac{|V|}{2}$ variables from V. When the process stops at an addition gate, $\partial_x g$ is a $\sum^2$-read$_V$-$(k-1)$ formula that may depend on many variables from V. In either case, the resulting $\partial_x F$ meets the requirements of the lemma. Note that since we assumed F depends on all variables in V, $\partial_x F \not\equiv 0$. ∎

Lemma 3.10 can be viewed as a simplified version of Lemma 3.11 for the case $k = 1$.

Algorithm 1 SPLIT(g, k, V) - A read$_V$-k fragmenting algorithm

---

Input: $k \geq 2, n \in \mathbb{N}$, $\emptyset \subsetneq V \subseteq [n]$ and g is an n-variate, multilinear read$_V$-k formula that depends on all variables in V. There exists a variable in V that occurs k times in g.

Output: $x \in V$, such that $\partial_x g$ meets the conditions of Lemma 3.11

1: **case** $g = \alpha(g_1 \cdot g_2) + \beta$

2: **if** $\exists i \in \{1, 2\}, x \in V$ where x occurs k times in g_i and $|\text{var}(g_i) \cap V| > \frac{|V|}{2}$ **then**

3: **return** SPLIT(g_i, k, V)

4: **case** $g = \alpha(g_1 + g_2) + \beta$

5: **if** $\exists i \in \{1, 2\}, x \in V$ where x occurs k times in g_i **then**

6: **return** SPLIT(g_i, k, V)

{Otherwise}

7: **return** $x \in V$ that occurs k times in g.

3.5.3 Fragmenting Sparse-Substituted Formulas

In this subsection we extend our fragmenting arguments to work for sparse-substituted formulas.

First consider a multilinear sparse-substituted read-once formula F. The idea is to apply the argument from Lemma 3.10 and the chain rule to locate a variable x such that $\partial_x F$ is *almost* fragmented. By this we mean that each of the factors of $\partial_x F$ depends on at most half of the variables except the factor that was originally a sparse polynomial that depends on x. The sparse polynomial, say f, may depend on too many variables. In that case we perform further operations so that f factors into small pieces. Through a sequence of partial derivatives and zero-substitutions we eliminate all but one term in f. This implies that the sparse polynomial and hence the overall resulting formula F' is $\frac{1}{2}$-split. To perform the additional step, observe that for any variable x, either at most half of the terms in f depend on x or at most half do not. In the former case, taking the partial derivative with respect to x eliminates at least half of the terms; setting x to 0 has the same effect in the latter case. Repeating this process a number of times logarithmic in the maximum number of terms eliminates all but one of the terms, resulting in a trivially split formula.

This is the intuition behind the sparse-substituted extension of Lemma 3.10 and corresponds to the first part of the next lemma. The second part is the sparse-substituted extension of Lemma 3.11 and follows from that lemma by a simple observation.

Lemma 3.12 (Fragmentation Lemma). *Let $\emptyset \subsetneq V \subseteq [n]$, $k \geq 1$, and let F be an n-variable multilinear sparse-substituted read$_V$-k formula that depends on at least one variable in V. Let t denote the maximum number of terms in each substituted polynomial.*

1. *If $k = 1$ there exist disjoint sets of variables $P, Z \subseteq V$ with $|P \cup Z| \leq (\log(t) + 1)$ such that $\partial_P F|_{Z \leftarrow 0}$ is nonzero and is a product of factors that each depend on at most $\frac{|V|}{2}$ variables from V (and possibly more variables outside of V). Moreover, the factors are subformulas of F and at most one formula of the form $\partial_P f|_{Z \leftarrow 0}$ where f is one of the sparse substitutions.*

2. *Otherwise, there exists a variable $x \in V$ such that $\partial_x F$ is nonzero and is the product of*

   a) *subformulas of F each depending on at most $\frac{|V|}{2}$ variables from V (and possibly more variables outside of V), and*

   b) *at most one multilinear sparse-substituted $\sum^2$-read$_V$-$(k-1)$ formula, which is the derivative with respect to x of some subformula of F.*

Proof. We argue the two parts separately.

Part 1. Assume without loss of generality that F has fanin 2, depends on all variables in V, and that the children of multiplication gates are variable disjoint with respect to V.

Use Lemma 3.10 to select a variable, x, such that $\partial_x F$ is the product of multilinear sparse-substituted read$_V$-once formulas on at most $\frac{|V|}{2}$ variables from V and possibly a single sparse polynomial on more than $\frac{|V|}{2}$ variables from V (which originally depended on x). If the large sparse factor is not present, the lemma is complete with $P = \{x\}$ and $Z = \emptyset$. Therefore, assume otherwise.

Let f be the large sparse polynomial factor of $\partial_x F$. A sparse polynomial can be thought of as a sparse sum of terms over the variables in V where the coefficients are sparse

polynomials in $\mathbb{F}[[n]\backslash V]$ (the constant term counts). If the number of terms in f is less than two, f can be represented as a product of multilinear sparse-substituted read$_V$-once formulas, each depending on a single variable from V. Also, for each variable in V that f depends on, we can assume that variable is present in at least one term, but not all of them. Otherwise, that variable can be pulled out as a factor multilinear sparse-substituted read$_V$-once formula. Therefore, we can assume that the number of terms in f is at least two and every variable in V that f depends on is present in at least one term but not in every term. Thus, for each variable $y \in \mathrm{var}(f) \cap V$, at least one of $f|_{y \leftarrow 0}$ or $\partial_y f$ has at most half as many terms as f. Since f has at most t terms, at most $\log t$ many partial derivatives and zero-substitutions are sufficient to eliminate all but one of the terms in f. Therefore, there are choices for disjoint sets P' and Z such that $\partial_{P'} f|_{Z \leftarrow 0}$ becomes a term over V (which is the product of univariate read$_V$-once formulas). Choosing $P \doteq \{x\} \cup P'$ and Z, lifts the required property to $\partial_P F|_{Z \leftarrow 0}$. Since F is multilinear, the operations we perform ensure that $\partial_P F|_{Z \leftarrow 0} \not\equiv 0$.

Part 2. Here the proof is essentially the same as the proof of Lemma 3.11. Since $k \geq 2$, the argument always halts at an internal gate and never reaches a sparse-substituted input. Only the number of occurrences of each variable is relevant to the decisions the argument makes. This implies that the argument does not change when sparse-substituted formulas are considered (and is even independent of the sparsity parameter). Thus, this part of the proof is immediate as a corollary to the proof of Lemma 3.11. ∎

Observe that the cost of applying the Fragmentation Lemma to a read-once formula is $\log(t) + 1$ partial derivatives and zero-substitutions, whereas applying it to a formula that is not read-once requires only a single partial derivative (though the promised result is weaker in this case).

It is useful to have a version of Part 2 of the Fragmentation Lemma generalized to

structurally-multilinear formulas. The argument is the same as for the earlier version except that in addition to selecting an appropriate x, we must pick an $\alpha \in \mathbb{F}$, such that $\partial_{x,\alpha} F \doteq F|_{x \leftarrow \alpha} - F|_{x \leftarrow 0}$ is nonzero. The directed partial derivative comes in here because $\partial_x F \doteq F|_{x \leftarrow 1} - F|_{x \leftarrow 0}$ may be zero even when F depends on x, because F is not multilinear.

Lemma 3.13 (Fragmentation Lemma for Structurally-Multilinear Formulas). *Let $\emptyset \subsetneq V \subseteq [n]$, $k \geq 2$, and let F be an n-variable structurally-multilinear sparse-substituted read$_V$-k formula that depends on at least one variable in V. Let t denote the maximum number of terms in each substituted polynomial. There exists a variable $x \in V$ and $\alpha \in \bar{\mathbb{F}}$ such that $\partial_{x,\alpha} F$ is nonzero and is the product of*

1. *subformulas of F each depending on at most $\frac{|V|}{2}$ variables from V (and possibly more variables outside of V), and*
2. *at most one structurally-multilinear sparse-substituted $\sum^2$-read$_V$-$(k-1)$ formula, which is the derivative with respect to x and α of some subformula of F.*

Proof. Repeat the proof of Lemma 3.12, Part 2, but add the following step. After selecting an appropriate variable x that F depends on, select an α such that $\partial_{x,\alpha} F \not\equiv 0$. By the Schwartz-Zippel Lemma, such an α exists within the algebraic closure $\bar{\mathbb{F}}$ of $\mathbb{F}$. Note that, in fact, if $|\mathbb{F}|$ is larger than the degree of x in F such an α is present in $\mathbb{F}$. ∎

3.5.4 Shattering Multilinear Formulas

The previous subsections establish a method for fragmenting multilinear sparse-substituted read-k formulas. We now apply the Fragmentation Lemma (Lemma 3.12) to shatter multilinear sparse-substituted $\sum^m$-read-k formulas. Recall that shattering is the act of simultaneously splitting all branches of a $\sum^m$-read-k formula. When $k = 1$, that is, in the case of multilinear sparse-substituted $\sum^m$-read-once formulas, applying the Fragmentation

Lemma greedily to a factor that depends on the largest number of variables suffices to shatter a multilinear sparse-substituted $\sum^m$-read-once formula to an arbitrary level. To obtain an α-split formula in the end, we need $O(m\frac{(\log(t)+1)}{\alpha})$ partial derivatives and zero-substitutions.

In the case of arbitrary read-value $k > 1$ the Fragmentation Lemma is not immediately sufficient for the task. As in the read-once case, we can apply the lemma greedily to a largest factor of a read-k branch to α-split the branch within at most $\frac{2}{\alpha}$ applications. However, this is assuming that Case 2b of the Fragmentation Lemma never occurs where the $\sum^2$-read-$(k-1)$ factor depends on more than half (possibly all) of the variables. When this case occurs the fragmentation process fails to split the formula into pieces each depending on few variables. To resolve the issue, we leverage the fact that this troublesome factor is both large and a $\sum^2$-read-$(k-1)$ formula.

Consider a specific read-k formula F on n variables. Apply the Fragmentation Lemma to F. Suppose that Case 2b of the lemma occurs, producing a variable x, and that the corresponding $\sum^2$-read-$(k-1)$ factor of $\partial_x F$ depends on more than $\frac{n}{2}$ of the variables. Without loss of generality, $\partial_x F = H \cdot (H_1 + H_2)$, where H is a product of read-k formulas each depending on at most $\frac{n}{2}$ variables, and both H_1 and H_2 are read-$(k-1)$ formulas. Rewrite F by distributing the top level multiplication over addition:

$$F' \doteq (H \cdot H_1) + (H \cdot H_2) \equiv H \cdot (H_1 + H_2) = \partial_x F.$$

Let $V \doteq \mathrm{var}(H_1 + H_2)$. F' is explicitly a $\sum^2$-read$_V$-$(k-1)$ formula and a read$_V$-k formula. However, F' is almost certainly not a read-k formula. By further restricting to the largest set of variables that appear the exact same number of times in the larger of the two subformulas H_1 and H_2, we can argue the existence of a subset $V' \subseteq V$ that contains at least a $\frac{1}{2k}$ fraction of the variables in V such that the read of H_1 and H_2 with respect to V' sum to at most k. Note that prior to this restriction the upper bound on this sum is $2(k-1)$. This

action effectively breaks up the original formula F into two branches without increasing the sum of the read values of the branches. Since $|V| \geq \frac{n}{2}$, the set V' is at most a factor $4k$ smaller than n, and the number of branches increased by one.

This operation can be performed at most $k-1$ times on a read-k formula before either: (i) the attempted greedy splitting is successful, or (ii) the formula becomes the sum of k read-once formulas with respect to some subset V of $[n]$. In the latter case we are effectively in the situation we first described with $k = 1$, and all subsequent splittings will succeed. In either case we obtain a formula that is shattered with respect to a subset V that is at most a factor $k^{O(k)}$ smaller than n.

In summary, the Shattering Lemma splits multilinear sparse-substituted $\sum^m$-read-k formulas to arbitrary degree, albeit with some restriction of the variable set and an increase in top fanin. Moreover, each of the branches in the shattered formula are present in the original input formula, either as such or after taking some partial derivatives and zero-substitutions. This technical property follows from the properties of the Fragmentation Lemma and will be needed in the eventual application.

Lemma 3.14 (Shattering Lemma). *Let $\alpha : \mathbb{N} \to (0, 1]$ be a non-increasing function. Let $F \in \mathbb{F}[x_1, \ldots, x_n]$ be a formula of the form $F = c + \sum_{i=1}^{m} F_i$, where c is a constant, and each F_i is a non-constant multilinear sparse-substituted read-k_i formula. Let t denote the maximum number of terms in each substituted polynomial. There exist disjoint subsets $P, Z, V \subseteq [n]$ such that $\partial_P F|_{Z \leftarrow 0}$ can be written as $c' + \sum_{i=1}^{m'} F'_i$, where c' is a constant, and*

- $m' \leq k \doteq \sum_{i=1}^{m} k_i$,
- *each F'_i is multilinear and $\alpha(m'+2)$-split$_V$,*
- $|P \cup Z| \leq (k - m + 1) \cdot \frac{4k}{\alpha(k+2)} \cdot (\log(t) + 1)$, *and*
- $|V| \geq \left(\frac{\alpha(k+2)}{8k}\right)^{k-m} \cdot n - \frac{8k}{\alpha(k+2)} \cdot (\log(t) + 1)$.

Moreover, the factors of each of the F'_i's are of the form $\partial_{\tilde{P}} f|_{Z \leftarrow 0}$, where f is some subformula of some F_j and $\tilde{P} \subseteq P$.

Proof. We iteratively construct disjoint subsets $P, Z, V \subseteq [n]$, maintaining the invariant that $\partial_P F|_{Z \leftarrow 0}$ can be written as $F' \doteq c' + \sum_{i=1}^{m'} F'_i$ where (1) each F'_i is a read$_V$-k'_i formula and c' is a constant, (2) $m' \leq k$, (3) $\sum_{i=1}^{m'} k'_i \doteq k' \leq k$, and (4) each F'_i is the product of factors of the form $\partial_{\tilde{P}} f|_{Z \leftarrow 0}$ where f is some subformula of some F_j and $\tilde{P} \subseteq P$. Setting $P \leftarrow \emptyset, Z \leftarrow \emptyset, V \leftarrow [n]$ and $F' \leftarrow F$ realizes the invariant initially. The fact that $m' \leq k$ follows because each F_i is non-constant.

The goal of our algorithm is to $\alpha(m'+2)$-split$_V$ the formula F'. Each iteration (but the last) consists of two phases: a splitting phase, and a rewriting phase. In the splitting phase we attempt to split F' by greedily applying the Fragmentation Lemma (Lemma 3.12) on each of the branches F'_i. The splitting phase may get stuck because of a $\sum^2$-read$_V$-$(k'_i - 1)$ subformula that blocks further splitting. If not and the resulting F' is sufficiently split, the algorithm halts. Otherwise, the algorithm enters the rewriting phase where it expands the

subformula that blocked the Fragmentation Lemma and reasserts the invariant, after which the next iteration starts. A potential argument shows that the number of iterations until a successful splitting phase is bounded by $k - m$. We first describe the splitting and rewriting phases in more detail, then argue termination and analyze what bounds we obtain for the sizes of the sets P, Z, and V.

Splitting. Assume that F' is not $\frac{\alpha(m'+2)}{2}$-split$_V$, otherwise halt. Let F'_{ij} be a subformula of F' that depends on the most variables in V out of all the factors of the F'_i's. Apply the Fragmentation Lemma (Lemma 3.12) with respect to the set $V \cap \mathrm{var}(F'_{ij})$ to produce sets $P', Z' \subseteq [n]$. By the Fragmentation Lemma exactly one of the following holds: (i) the factors of $\partial_{P'} F'_{ij}|_{Z' \leftarrow 0}$ depend on at most $\frac{|V \cap \mathrm{var}(F'_{ij})|}{2}$ variables in V, or (ii) $\partial_{P'} F'_{ij}|_{Z' \leftarrow 0}$ has one multilinear sparse-substituted $\sum^2$-read$_V$-$(k'_i - 1)$ factor which depends on more than $\frac{|V \cap \mathrm{var}(F'_{ij})|}{2}$ variables in V.

Repeatedly perform this greedy application, adding elements to the sets P' and Z' until either case (ii) above occurs or $\partial_{P'} F'|_{Z' \leftarrow 0}$ is $\frac{\alpha(m'+2)}{2}$-split$_V$. In the former case we start a rewriting phase and modify $\partial_{P'} F'|_{Z' \leftarrow 0}$ before we re-attempt to split. In the latter case our goal has been achieved provided that $|P' \cup Z'| \leq \frac{|V|}{2}$: We can add P' to the set P we already had, similarly add Z' to Z, and replace V by $V' \doteq V \setminus (P' \cup Z')$. The assumption that $|P' \cup Z'| \leq \frac{|V|}{2}$ guarantees that $|V'| \geq \frac{|V|}{2}$. Since $\partial_P F|_{Z \leftarrow 0}$ (which equals $\partial_{P'} F'|_{Z' \leftarrow 0}$) is $\frac{\alpha(m'+2)}{2}$-split$_V$, the latter inequality implies that the formula is $\alpha(m'+2)$-split$_{V'}$. If the assumption that $|P' \cup Z'| \leq \frac{|V|}{2}$ does not hold, then outputting $V' = \emptyset$ will meet the size bound for that set and trivially make the formula $\partial_P F|_{Z \leftarrow 0}$ $\alpha(m'+2)$-split$_{V'}$.

The splitting phase maintains the invariant. Regarding part (4) of the invariant, observe that the factors produced by the Fragmentation Lemma are subformulas of the input to the Fragmentation Lemma (for which the invariant initially held).

Rewriting. We now describe the rewriting phase. Let V refer to the situation at the start

of the preceding splitting phase. Let F'_{ij} be the subformula the splitting phase blocked on, and let H_1 and H_2 denote the two branches of the multilinear sparse-substituted $\sum^2$-read$_V$-(k'_i-1) subformula of $\partial_x F'_i$ that caused the blocking Case 2b of the Fragmentation Lemma to happen. We have that $\partial_{P'} F'_{ij}|_{Z' \leftarrow 0} = H \cdot (H_1 + H_2)$, where H is some read$_V$-k'_i formula that is independent of the variables in $V \cap \text{var}(H_1 + H_2)$. Let $V' \doteq V \cap \text{var}(H_1 + H_2)$. Partition V' into sets $\{V'_0, ..., V'_{k'_i-1}\}$ based on the exact number of occurrences of each variable in H_1. Let V'' be any set from this partitioning excluding the set V'_0 (we will restrict the choice of V'' later). This implies that H_1 is read$_{V''}$-k'_{i1} and H_2 is read$_{V''}$-k'_{i2} for some integers k'_{i1} and k'_{i2} such that $k'_{i1}, k'_{i2} < k'_i$ and $k'_{i1} + k'_{i2} \leq k'_i$.

Rewrite $\partial_{P'} F'|_{Z' \leftarrow 0}$ as a top fanin $m' + 2$ formula by distributing multiplication over addition in the term $\partial_{P'} F'_i|_{Z' \leftarrow 0}$:

$$\partial_{P'} F'|_{Z' \leftarrow 0} \equiv (H \cdot H_1) + (H \cdot H_2) + \sum_{j \neq i} \partial_{P'} F'_j|_{Z' \leftarrow 0}. \tag{3.1}$$

Observe that $\sum_{j \neq i} \partial_{P'} F'_j|_{Z' \leftarrow 0}$ is a read$_{V''}$-$(\sum_{j \neq i} k'_j)$ formula as partial derivatives and substitutions do not increase the read-value, and $V'' \subseteq V$. The term $(H \cdot H_1) + (H \cdot H_2)$ may not be a read$_V$-k'_i formula, but it must be a read$_{V'}$-k'_i formula. It is explicitly the sum of a read$_{V''}$-k'_{i1} formula and a read$_{V''}$-k'_{i2} formula for some $k'_{i1}, k'_{i2} < k'_i$ with $k'_{i1} + k'_{i2} \leq k'_i$. The representation of $\partial_{P'} F'|_{Z' \leftarrow 0}$ in Equation (3.1) is therefore a read$_{V''}$-k' formula with top fanin $m' + 2$.

Set F' to be this representation of $\partial_{P'} F'|_{Z' \leftarrow 0}$. Merge branches that have become constant into a single constant branch. This maintains the invariant that $m' \leq k' \leq k$. Setting $V \leftarrow V''$ makes F' a top-fanin-$(m' + 1)$ read$_V$-k' formula. As for part (4) of the invariant, note that the subformula F'_{ij} which blocked the Fragmentation Lemma originally satisfied it during the splitting phase. This means that with respect to the additional partial derivatives and zero-substitutions performed for the attempted split, H_1 and H_2, as well as H, satisfy

the invariant as new factors of the branches F'_i. Thus, the new F' satisfies the full invariant. This completes the rewriting phase and one full iteration of the algorithm.

Correctness. We repeat the sequence of splitting and rewriting phases until a splitting phase runs till completion. In that case the algorithm produces disjoint sets $P, Z, V \subseteq [n]$ such that $\partial_P F|_{Z \leftarrow 0}$ can be written as a $\alpha(m'+2)$-split$_V$ formula with top fanin $m'+1 \leq k+1$.

Apart from the size bounds on the sets P, Z, and V, all that remains to establish correctness is termination. To argue the latter we use the following potential argument. Consider the sum $\sum_{i=1}^{m'} k'_i$ and view it as m' blocks of integer size, where k'_i is the size of the ith block. Over the course of the algorithm blocks can only stay the same, shrink, or be split in a nontrivial way. The latter is what happens in a rewriting phase. As soon as all blocks are of size at most 1, the splitting phase is guaranteed to run successfully because Case 2b of the Fragmentation Lemma cannot occur for read-once formulas, and the algorithm terminates. As we start out with m nontrivial blocks and a value of k for the sum, there can be no more than $k-m$ nontrivial splits. Therefore, there are no more than $k-m$ rewriting phases and $k-m+1$ splitting phases.

Analysis. We now bound the size of $P \cup Z$. We first analyze how many times the Fragmentation Lemma is applied in each splitting phase. The goal is to $\frac{\alpha(m'+2)}{2}$-split$_V$ each of the m' branches. To $\frac{\alpha(m'+2)}{2}$-split$_V$ one branch, $\frac{4}{\alpha(m'+2)}$ applications of the Fragmentation Lemma are sufficient, since each application reduces the intersection of the factors with V to at most half the original amount. Since the invariant maintains $m' \leq k$ and α is non-increasing, we can upper bound the number of applications of the Fragmentation Lemma during an arbitrary iteration by $\frac{4m'}{\alpha(m'+2)} \leq \frac{4k}{\alpha(k+2)}$. Each single application of the Fragmentation Lemma adds at most $(\log(t)+1)$ variables to P' and Z'. Since there are at most $k-m+1$ splitting phases, across all iterations at most $(k-m+1)(\log(t)+1)\frac{4k}{\alpha(k+2)}$ variables are added to $P \cup Z$.

We finish by lower bounding the size of V. Consider the change in $|V|$ over one combined splitting/rewriting iteration. We have that $|V'| \geq \frac{\alpha(m'+2)}{4}|V|$, because F' was not $\frac{\alpha(m'+2)}{2}$-split$_V$ before attempting to split F'_{ij} (so the largest number of variables in V that a factor depends on is at least $\frac{\alpha(m'+2)}{2} \cdot |V|$), F'_{ij} was chosen for its maximal dependence on variables from V, and $|\text{var}(H_1 + H_2) \cap V| \geq |\text{var}(F'_{ij}) \cap V|/2$. If we pick V'' to be the largest set from the partitioning $\{V'_0, V'_1, ..., V'_{k'_i-1}\}$ excluding V'_0, and we assume without loss of generality that $|\text{var}(H_1)| \geq |\text{var}(H_2)|$, we have that $|V''| \geq \frac{1}{2(k'_i-1)}|V'|$. Combining these inequalities and using the facts that α is non-increasing and $k \geq k'_i, m'$ gives:

$$|V''| \geq \frac{1}{2(k'_i-1)}|V'| \geq \frac{\alpha(m'+2)}{8(k'_i-1)}|V| \geq \frac{\alpha(k+2)}{8k}|V|.$$

This means that $|V|$ decreases by a factor of at most $\frac{8k}{\alpha(k+2)}$ in each combined splitting/rewriting iteration. At the end of the final splitting phase $|V'| \geq |V| - 2|P' \cup Z'|$ because V' is set to the empty set when $|P' \cup Z'| \geq \frac{|V|}{2}$. Recall that $|P' \cup Z'| \leq \frac{4k}{\alpha(k+2)}(\log(t)+1)$. Since there are at most $k - m$ combined splitting/rewriting iterations, this gives the following lower bound at the end:

$$|V| \geq \left(\frac{\alpha(k+2)}{8k}\right)^{k-m} \cdot n - \frac{8k}{\alpha(k+2)} \cdot (\log(t)+1). \qquad \blacksquare$$

3.6 Reducing Testing Read-$(k+1)$ Formulas to Testing $\sum^2$-Read-k Formulas

In this section we describe two methods of reducing identity testing structurally-multilinear read-$(k+1)$ formulas to identity testing structurally-multilinear $\sum^2$-read-k formulas. The first reduction is non-blackbox and is elementary. The second reduction is blackbox and makes use of the Fragmentation Lemma of the preceding section.

3.6.1 Non-Blackbox Reduction

Recall that we only need to deal with multilinear sparse-substituted formulas, because we can transform structurally-multilinear formulas into multilinear sparse-substituted formulas in a non-blackbox way while preserving (non-)zeroness.

The intuition for this reduction is somewhat similar to that for the Fragmentation Lemma. Consider a subformula g of a multilinear sparse-substituted read-$(k+1)$ formula F where g is of the form $g = g_1 + g_2$ and g_1 is read-$(k+1)$ but not read-k. There must be some variable x that appears $k+1$ times in g_1 and nowhere else in F. If g_1 actually depends on x, then g is nonzero. This is irrespective of whether g_2 is nonzero, because it is not possible for g_2 to cancel out the contribution of x present in g_1. In general, if all occurrences of a variable x are contained in an addition branch and that branch depends on x, the addition gate must be nonzero. The polynomials computed by gates above g can only be zero if the zero polynomial is multiplied in. Now, consider replacing g_1 with a fresh variable. The above reasoning shows that this transformation does not change whether the overall formula is nonzero. If a branch contains all occurrences of x but does not depend on x, setting x to 0 does not affect the value computed by the formula. This observation allows us to eliminate variables that are read $k+1$ times, and thereby transform a multilinear sparse-substituted read-$(k+1)$ formula into a multilinear sparse-substituted read-k formula without affecting the (non-)zeroness of the formula.

In order to execute the transformation, we need to be able to decide whether the subformula g_1 depends on the variable x. If we apply the transformation in a bottom-up fashion, by the time we need to make that decision the formula g_1 is multilinear, sparse-substituted and $\sum^2$-read-k. We can use a polynomial identity test for such formulas to check whether $\partial_x g_1 \equiv 0$. This is the idea behind the non-blackbox reduction from identity testing multilinear sparse-substituted read-$(k+1)$ formulas to identity testing multilinear

sparse-substituted $\sum^2$-read-k formulas.

Lemma 3.15 (Read-$(k+1)$ PIT $\leq \sum^2$-Read-k PIT – Non-Blackbox). *For an integer $k \geq 1$, given a deterministic identity testing algorithm for n-variate size-s multilinear sparse-substituted $\sum^2$-read-k formulas that runs in time $T(k, n, s, t)$, where t denotes the maximum number of terms in each substituted polynomial, there is a deterministic algorithm that tests n-variate size-s multilinear sparse-substituted read-$(k+1)$ formulas that runs in time $O(kn^2 \cdot T(k, n, s, t) + \text{poly}(k, n, s, t))$.*

Proof. Let F be a multilinear sparse-substituted read-$(k+1)$ formula. The goal of the algorithm is to transform the gates of F in a bottom-up fashion into read-k formulas while preserving the (non-)zeroness of F. The transformation also eliminates variables which gates do not depend on. Because F is multilinear, this second property ensures that multiplication gates are explicitly variable disjoint. Once F is transformed in this way, the identity test is immediate from the assumed identity testing algorithm.

As a first step we argue that the following transformation preserves the (non-)zeroness of F. Consider a gate g in F that contains all occurrences of some variable x and depends on x. Define $F_{-g}(\bar{x}, y)$ as the formula where the gate g has been replaced by a new variable y (note, $F_{-g}(\bar{x}, g) = F$). F_{-g} does not depend on x because all occurrences of x are in g. Then, because F_{-g} is a formula and y occurs only once, without loss of generality, write:

$$F \equiv F_{-g}(\bar{x}, g) \equiv P(\bar{x}) + Q(\bar{x}) \cdot g,$$

for two polynomials P and Q that do not depend on x. If $Q \equiv 0$, then F is independent of g, so $F_{-g} \equiv F$. If $Q \not\equiv 0$, then F is nonzero because g depends on x, but P does not depend on x, so the x component cannot be canceled. By definition, $F \not\equiv 0$ implies $F_{-g} \not\equiv 0$. Therefore we can conclude that for such gates g, $F \equiv 0$ iff $F_{-g} \equiv 0$.

Algorithm 2 Transforming a read-$(k+1)$ gate into an "equivalent" read-k gate

Input: $k \geq 1$, F is a multilinear sparse-substituted read-$(k+1)$ formula, g is a multilinear sparse-substituted read-$(k+1)$ subformula of F whose children are read-k formulas that depend on all variables that appear within them, and $\mathcal{A}$ is a deterministic identity testing algorithm for multilinear sparse-substituted $\sum^2$-read-k formulas.

Output: g is a read-k formula that depends on all variables that appear within it. Except for variables that do not appear in F_{-g}, the number of occurrences of a variable in g does not increase. The (non-)zeroness of F is unchanged with respect to the transformation of g.

1: **case** $g = \alpha(g_1 + g_2) + \beta$
2: **for all** $x \in \text{var}(g)$
3: **if** $\partial_x(g_1 + g_2) \equiv 0$ {Invoking the subroutine $\mathcal{A}$} **then**
4: Replace g by $g|_{x \leftarrow 0}$
5: **else if** x occurs $k+1$ times in g **then**
6: Replace g by a new variable y
7: **case** $g = \alpha(g_1 \cdot g_2) + \beta$
8: **if** $g_1 \equiv 0$ **OR** $g_2 \equiv 0$ {Invoking the subroutine $\mathcal{A}$} **then**
9: Replace g by β
10: **case** g is a sparse-substituted input
11: Simplify the list of terms

Now, process the gates g of F in a bottom-up fashion. Note that the algorithm realizes the following properties: (1) the (non-)zeroness of F does not change, (2) g is a multilinear sparse-substituted read-k formula, (3) except for variables that only appear in g, the number of occurrences of a variable in g does not increase, and (4) g depends on all variables that

appear in it. There are three possible cases for g.

1. Case $g = \alpha(g_1 + g_2) + \beta$. We first go over all original variables x that appear in g. Let x be such a variable. Since the children g_1 and g_2 of g are read-k, g is a multilinear sparse-substituted $\sum^2$-read-k formula. Compute $\partial_x g$ as a multilinear sparse-substituted $\sum^2$-read-k formula. This is easy because the multiplication gates within g_1 and g_2 are variable disjoint. Now, test whether $\partial_x g \equiv 0$ using the hypothesized identity testing algorithm. If $\partial_x g \equiv 0$, replace g by $g|_{x \leftarrow 0}$; otherwise, if x occurs $k+1$ times in g, replace g by a fresh variable y, if not, do nothing.

2. Case $g = \alpha(g_1 \cdot g_2) + \beta$. Because property (4) holds for the children of g, and g is multilinear, g is a read-k formula. Check whether g_1 or g_2 are identically zero, using the identity test for read-k formulas. If either formula is zero, replace g with β. Otherwise, do nothing.

3. Case g is a sparse-substituted input. In order to realize properties (1-4), all we need to do is to simplify the list of terms by collecting duplicate monomials and dropping them if the coefficient is zero.

For clarity, the algorithm is described in Algorithm 2. Overall, it transforms F from a multilinear sparse-substituted read-$(k+1)$ formula into a multilinear sparse-substituted read-k formula without affecting the (non-)zeroness. The $\sum^2$-read-k formula identity test is applied at most n times at each gate to determine the dependence on the original variables. Since F is in standard form, this takes at most $O(kn^2)$ identity tests overall. Adding polynomial-time for computing the standard form, traversing F, computing the partial derivatives, and doing the field arithmetic gives the running time claimed. ∎

3.6.2 Blackbox Reduction

Let F be a structurally-multilinear read-$(k+1)$ formula. We construct a generator for F using a generator $\mathcal{G}$ for structurally-multilinear $\sum^2$-read-k formulas. If F is a read-k formula, the assumed generator alone suffices. Otherwise, we apply the Fragmentation Lemma for structurally-multilinear sparse-substituted formulas (Lemma 3.13) to show that there is a partial derivative of F that has mostly small factors and, possibly, one factor that is a large structurally-multilinear $\sum^2$-read-k formula. In the former case the factors are small enough to be hit recursively, and in the latter case the factor is hit by the assumed generator for structurally-multilinear $\sum^2$-read-k formulas. The properties of the SV-generator G (Proposition 3.3 and Lemma 3.5) imply that if $\mathcal{G}$ is a generator for the partial derivative of a polynomial, then $\mathcal{G} + G_1$ is a generator for the original polynomial.

Lemma 3.16 (Read-$(k+1)$ PIT $\leq \sum^2$-Read-k PIT – Blackbox). *For an integer $k \geq 1$, let $\mathcal{G}$ be a generator for n-variate structurally-multilinear sparse-substituted $\sum^2$-read-k formulas, and let F be a nonzero n-variable structurally-multilinear sparse-substituted read-$(k+1)$ formula. Then $\mathcal{G} + G_{\log|var(F)|}$ hits F.*

Proof. First observe that if F is read-k, we are immediately done because $F \circ \mathcal{G} \not\equiv 0$ and $\bar{0}$ is in the range of G (by Proposition 3.3, Part 1).

The proof goes by induction on $|\mathrm{var}(F)|$. If $|\mathrm{var}(F)| = 0$, the lemma holds trivially as F is constant. If $|\mathrm{var}(F)| = 1$, F is a read-once formula, which is covered by the above observation. For the induction step, by the above observation we can assume that F is read-$(k+1)$ and not read-k. Therefore, F meets the conditions to apply the structurally-multilinear version of the Fragmentation Lemma (Lemma 3.13) with $V = \mathrm{var}(F)$. The lemma produces a variable $x \in \mathrm{var}(F)$ and $\alpha \in \bar{\mathbb{F}}$. The factors of $\partial_{x,\alpha} F$ all depend on at most $\frac{|\mathrm{var}(F)|}{2}$ variables and are read-$(k+1)$ formulas, except for at most one which is a $\sum^2$-read-k formula. The induction hypothesis gives that the former factors of $\partial_{x,\alpha} F$ are all

hit by $\mathcal{G} + G_{\log(|\mathrm{var}(F)|/2)}$. The latter factor (if it occurs) is hit by $\mathcal{G}$. Applying Lemma 3.5 gives that $\mathcal{G} + G_{\log(|\mathrm{var}(F)|/2)} + G_1$ hits F. Recalling Proposition 3.3, Part 3, implies that $\mathcal{G} + G_{\log|\mathrm{var}(F)|}$ hits F. ∎

3.7 Reducing Testing $\sum^2$-Read-k Formulas to Testing Read-k Formulas

In this section we present two methods of reducing identity testing structurally-multilinear sparse-substituted $\sum^2$-read-k formulas to identity testing structurally-multilinear sparse-substituted read-k formulas. We first develop those methods for *multilinear* rather than *structurally-multilinear* sparse-substituted formulas, and then show how to translate them to the latter setting.

Both reductions rely on a common theorem (Theorem 10) we prove in Section 3.7.2. Informally, that theorem says that for a nonzero multilinear sparse-substituted $\sum^2$-read-k formula F and a shift $\bar{\sigma}$ satisfying some simple conditions, the shifted formula $F(\bar{x} + \bar{\sigma})$ is hit by the SV-generator G_w with $w = k^{O(k)}(\log(t) + 1)$, where t denotes the maximum number of terms in each substituted polynomial.

Note that, since F is a nonzero polynomial, such a theorem is trivially true for a typical shift $\bar{\sigma}$, even with $w = 0$. The interesting part of the theorem is the simplicity of the conditions on $\bar{\sigma}$ that guarantee the hitting property. In particular, the properties needed of $\bar{\sigma}$ allow such a $\bar{\sigma}$ to be computed efficiently either by an identity test for multilinear sparse-substituted read-k formulas, or as an element in the range of a hitting set generator for such formulas.

In Section 3.7.1 we argue that small sums of specially shifted multilinear sparse-substituted read-k formulas cannot compute a term of high degree. This is the Key

Lemma for multilinear sparse-substituted formulas and is a formalization of Lemma 3.2 from the introduction. Using the Key Lemma and the hitting property of the SV-generator (Lemma 3.7), we prove, in Section 3.7.2, that the SV-generator hits small sums of specially shifted multilinear sparse-substituted read-k formulas. In Sections 3.7.3 and 3.7.4 we use Theorem 10 to argue reductions from identity testing multilinear sparse-substituted $\sum^2$-read-k formula to identity testing multilinear sparse-substituted read-k formulas in both the non-blackbox and blackbox settings.

In Section 3.7.5 we present a transformation $\mathcal{L}$ which maps structurally-multilinear read-k formulas to multilinear sparse-substituted read-k formulas while preserving the nonzeroness of the formula. Combining this transformation with the non-blackbox reduction for multilinear sparse-substituted formulas yields a non-blackbox reduction *directly* for structurally-multilinear sparse-substituted formulas. In the blackbox setting, the extension requires some more work. We first use the transformation to generalize the Key Lemma to structurally-multilinear sparse-substituted formulas, and then argue how the other ingredients transfer.

3.7.1 Proving the Key Lemma for Multilinear Formulas

In order to prove the Key Lemma, we first establish a similar lemma for *split* multilinear sparse-substituted formulas, and then apply the Shattering Lemma to lift the result to the bounded-read setting.

Let $F = \sum_{i=1}^{m} F_i$ be a sufficiently split multilinear sparse-substituted formula on n variables. By applying the structural witness for split formulas (Lemma 3.8) we can argue that if none of the F_i's are divisible by any variable then F cannot compute a term of the form $a \cdot M_n$, where a is a nonzero constant and, recall, M_n denotes the monomial $\prod_{i=1}^{n} x_i$. The idea is to consider the formula $F - a \cdot M_n$ and apply the structural witness to it in order

to show that it is nonzero. The non-divisibility condition and the natural properties of M_n immediately give simplicity. Minimality essentially comes for free because the argument is existential. The splitting required by the structural witness immediately follows from the splitting of F. Formalizing this idea yields the following lemma.

Lemma 3.17. *Let $F = \sum_{i=1}^{m} F_i$ be a multilinear sparse-substituted $\alpha(m+1)$-split formula on $n \geq 1$ variables, where $\alpha \doteq \frac{1}{R}$ and R is the function given by Lemma 3.8. If no F_i is divisible by any variable, then $F \not\equiv a \cdot \prod_{i=1}^{n} x_i$ for any nonzero constant a.*

Note that for a non-constant formula F on n variables to be $\alpha(m+1)$-split, n needs to be at least $1/\alpha(m+1)$.

Proof. Suppose for the sake of contradiction that $F \equiv a \cdot M_n$ for some nonzero constant a.

If there is some subsum of the branches of F that equals 0, eliminate all those branches. Not all branches of F may be eliminated in this way as this contradicts $a \cdot M_n \not\equiv 0$. Let $0 < m' \leq m$ be the remaining number of branches, and let F' denote the remaining branches. The formula $F' - a \cdot M_n$ is minimal and has top fanin $m' + 1$.

Now, suppose that there is some non-constant polynomial P that divides every remaining F_i. Since $F' \equiv a \cdot M_n$, then P also divides M_n. Because P is non-constant, some variable x divides P and hence divides each remaining F_i. This contradicts the hypothesized non-divisibility property of the F_i. Therefore $F' - a \cdot M_n$ is simple as a formula with top fanin $m' + 1$.

The previous two paragraphs establish that the $F' - a \cdot M_n$ is simple, minimal, and has top fanin $m' + 1$. Further, for every variable, there is some branch that depends on that variable, because the M_n branch depends on every variable. Observe that the M_n branch is trivially $\alpha(m'+1)$-split and every other branch is also $\alpha(m'+1)$-split as $m' \leq m$ and $\alpha \doteq \frac{1}{R}$ is decreasing. The structural witness for split formulas (Lemma 3.8) then implies

that $F' - a \cdot M_n \not\equiv 0$, and thus that $F \not\equiv a \cdot M_n$. This contradicts the initial assumption and concludes the proof. ∎

The property that the branches F_i are not divisible by any variable can be easily established by shifting the formula by a point $\bar{\sigma}$ that is a common nonzero of all the branches F_i. Indeed, if we pick $\bar{\sigma}$ such that $F_i(\bar{\sigma}) \neq 0$ then no variable can divide $F_i(\bar{x} + \bar{\sigma})$. This reasoning is formalized in the following corollary.

Corollary 3.2. *Let $F = \sum_{i=1}^{m} F_i$ be a multilinear sparse-substituted $\alpha(m+1)$-split formula on $n \geq 1$ variables, where $\alpha \doteq \frac{1}{R}$ and R is the function given by Lemma 3.8. If no F_i vanishes at $\bar{\sigma}$, then $F(\bar{x} + \bar{\sigma}) \not\equiv a \cdot \prod_{i=1}^{n} x_i$ for any nonzero constant a.*

Proof. Since the branches of F are $\alpha(m+1)$-split, the branches of $F(\bar{x} + \bar{\sigma})$ are also $\alpha(m+1)$-split. By assumption, $F_i(\bar{\sigma}) \neq 0$. Therefore, for each branch $i \in [m]$ and variable $x \in [n]$, $F_i(\bar{x} + \bar{\sigma})|_{x \leftarrow 0} \not\equiv 0$. This implies that no variables divide any branch $F_i(\bar{x} + \bar{\sigma})$. With this property established, apply Lemma 3.17 on $F(\bar{x} + \bar{\sigma})$ to conclude the proof. ∎

We now show how to lift Corollary 3.2 from split multilinear sparse-substituted formulas to sums of multilinear sparse-substituted bounded-read formulas. This yields our key lemma – that for such formulas F and a "good" shift $\bar{\sigma}$, $F(\bar{x} + \bar{\sigma})$ is not divisible by a term of large degree. For brevity, in the intuition below we discuss the simpler case of showing the formula is, instead, not identical to a term of large degree.

For the sake of contradiction suppose the opposite, i.e., that $F(\bar{x} + \bar{\sigma}) \equiv a \cdot M_n$ for some nonzero constant a and large n. Shatter F into $F' = \partial_P F|_{Z \leftarrow 0}$ using the Shattering Lemma (Lemma 3.14), and apply the same operations that shatter F to M_n. Observe that zero-substitutions are shifted into substitutions by $\bar{\sigma}$, and that $\partial_P M_n|_{Z \leftarrow (-\bar{\sigma})}$ is a nonzero term of degree $n - |P \cup Z|$ provided that no component of $\bar{\sigma}$ vanishes. After an appropriate substitution for variables outside of the set V from the Shattering Lemma, we obtain that

$F'(\bar{x} + \bar{\sigma}) \equiv a' \cdot M_V$ for some nonzero constant a' and $V \subseteq [n]$, where M_V denotes the product of the variables in V.

At this point we would like to apply Corollary 3.2 to derive a contradiction. However, we need to have that $|V| > 0$ and that $\bar{\sigma}$ is a common nonzero of all the branches of F'. The former follows from the bounds in the Shattering Lemma provided n is sufficiently large. To achieve the latter condition we impose a stronger requirement on the shift $\bar{\sigma}$ prior to shattering so that afterward $\bar{\sigma}$ is a common nonzero of the shattered branches. The Shattering Lemma tells us that the factors of the branches of the shattered formula are of the form $\partial_{\tilde{P}} f|_{Z \leftarrow 0}$ where f is some subformula of the F_i's and $\tilde{P} \subseteq P$. Therefore, we require that $\bar{\sigma}$ is a common nonzero of all such subformulas that are nonzero. This is what we mean by a "good" shift.

One additional technical detail is that we must apply a substitution to the variables outside of V that preserves the properties of $\bar{\sigma}$ and does not zero M_n. This step is in the same spirit as the argument in the proof of the structural witness for split formulas (Lemma 3.8), namely that a typical assignment suffices.

With these ideas in mind, the key lemma is as follows.

Lemma 3.18 (Key Lemma). *Let $F = c + \sum_{i=1}^{m} F_i$, where c is a constant, and each $F_i \in \mathbb{F}[x_1, \ldots, x_n]$ is a non-constant multilinear sparse-substituted read-k_i formula. If $\bar{\sigma}$ is a common nonzero of the nonzero formulas of the form $\partial_P f|_{Z \leftarrow 0}$ where f is a subformula of the F_i's and $|P \cup Z| \leq b \doteq (k - m + 1) \cdot 4k \cdot R(k + 2) \cdot (\log(t) + 1)$, then*

$$F(\bar{x} + \bar{\sigma}) \notin \mathcal{D}_\ell,$$

for $\ell \geq w \doteq (8k \cdot R(k + 2))^{k-m+1}(\log(t) + 1)$, where $k \doteq \sum_{i=1}^{m} k_i$, t denotes the maximum number of terms in each substituted polynomial, and R is the function given by Lemma 3.8.

Proof. Assume the contrary, without loss of generality, that $F(\bar{x}+\bar{\sigma}) \equiv Q \cdot M_\ell$ for some nonzero multilinear polynomial Q and $\ell \geq w$. If any variable divides Q, factor that variable out and increase ℓ by one. This way we can assume Q is not divisible by any variables.

We first argue that, without loss of generality, $\mathrm{var}(F_i) \subseteq [\ell]$ for all $i \in [m]$. Suppose that some F_i depends on a variable $x \notin [\ell]$. Replace F with $F|_{x \leftarrow \bar{\sigma}}$, and observe this is equivalent to substituting 0 for x in $F(\bar{x}+\bar{\sigma})$. We have $M_\ell|_{x\leftarrow 0} = M_\ell$, because M_ℓ does not depend on x, and $Q' \doteq Q|_{x\leftarrow 0} \not\equiv 0$, because x does not divide Q. The assignment $\bar{\sigma}$ remains a common nonzero of the stated type of formulas, now with F_i replaced by $F_i|_{x\leftarrow\bar{\sigma}}$. If Q' is divisible by any variables factor them out, and increase ℓ accordingly. Repeat this procedure until $\mathrm{var}(F_i) \subseteq [\ell]$ for all $i \in [m]$.

Note that these substitution may make some branches constant. In this case combine these constant branches into a single constant branch. Since all F_i were originally non-constant, the quantity $k-m$ has not increased.

Define $\alpha \doteq \frac{1}{R}$. Shatter F using Lemma 3.14. This produces the sets P, Z and V. Let $F' \doteq \partial_P F|_{Z\leftarrow 0}$. By the Shattering Lemma $F' = c' + \sum_{i=1}^{m'} F_i'$ is a multilinear sparse-substituted formula that has top fanin $m'+1 \leq k+1$, is $\alpha(m'+2)$-split$_V$, and each F_i' is a product of factors of formulas of the form $\partial_{\tilde{P}} f|_{Z\leftarrow 0}$ where f is a subformula of an F_i and $\tilde{P} \subseteq P$. Assume without loss of generality that each F_i' is nonzero. By the lemma, $|P \cup Z| \leq b$. By hypothesis, the subformulas of the above form do not vanish at $\bar{\sigma}$. These properties imply that $F_i'(\bar{\sigma}) \neq 0$ for each $i \in [m']$.

There is an assignment to the variables in $[\ell]\backslash V$ that: (1) preserves $\bar{\sigma}$ as a nonzero of the F_i''s on the remaining variables V, and (2) differs in every component from $\bar{\sigma}$. In fact, a typical assignment suffices. To see this, consider the polynomial:

$$\Phi \doteq \left(\prod_{i=1}^{m'} F_i'|_{V\leftarrow\bar{\sigma}}\right) \cdot \prod_{j\in([\ell]\backslash V)} (x_j - \sigma_j).$$

The polynomial Φ is nonzero because the F_i''s do not vanish at $\bar{\sigma}$. Thus, a nonzero assignment for Φ satisfies the requirements above. Pick $\bar{\beta}$ to be any such assignment.

Let $F'' \doteq F'|_{([\ell]\setminus V)\leftarrow\bar{\beta}}$, where the F_i'' are defined similarly. By the first property of $\bar{\beta}$, $F_i''(\bar{\sigma}) \neq 0$. By the second property of $\bar{\beta}$, $M_\ell|_{([\ell]\setminus V)\leftarrow(\bar{\beta}-\bar{\sigma})}$ is a nonzero term over the variables V. Then using the initial assumption write

$$F''(\bar{x}+\bar{\sigma}) \equiv F'(\bar{x}+\bar{\sigma})|_{([\ell]\setminus V)\leftarrow(\bar{\beta}-\bar{\sigma})} \equiv a \cdot M_\ell|_{([\ell]\setminus V)\leftarrow(\bar{\beta}-\bar{\sigma})} \equiv a' \cdot M_V,$$

for some nonzero constant a'. Now, $F'' \in \mathbb{F}[V]$ is a multilinear sparse-substituted $\alpha(m'+2)$-split$_V$ formula with top fanin $m'+1$, where no branch vanishes at $\bar{\sigma}$. Thus, we obtain a contradiction with Corollary 3.2 as long as $|V| > 0$. By the bound on $|V|$ given in the Shattering Lemma and then condition that $\ell \geq w$, the latter is the case for $w \geq (8k \cdot R(k+2))^{k-m+1}(\log(t)+1)$. ∎

3.7.2 Generator for Shifted Multilinear Formulas

In this subsection we show that the SV-generator hits small sums of specially shifted multilinear sparse-substituted bounded-read formulas. Our argument critically relies on the property given in Lemma 3.7 – that the SV-generator hits any class of polynomials that is closed under zero-substitutions and such that no term of high degree divides polynomials in the class.

In order to prove a usable theorem for our applications, we use the Key Lemma (Lemma 3.18) to construct a class of polynomials sufficient to apply Lemma 3.7. Let F be a formula, $\bar{\sigma}$ be a shift, and w be as in the statement of the Key Lemma. Consider $F(\bar{x}+\bar{\sigma})$. By the Key Lemma, $F(\bar{x}+\bar{\sigma}) \notin \mathcal{D}_n$, for $n \geq w$. Now consider substituting 0 for x in $F(\bar{x}+\bar{\sigma})$, this equivalent to substituting σ for x in F then shifting all other variables by $\bar{\sigma}$. This means that the preconditions of the Key Lemma are satisfied for $F|_{x\leftarrow\sigma}$, and

hence $F(\bar{x}+\bar{\sigma})|_{x\leftarrow 0} \notin \mathcal{D}_n$, for $n \geq w$. This argument can be repeated to get that each zero substitution of $F(\bar{x}+\bar{\sigma})$ is not in $\mathcal{D}_n$, for $n \geq w$. The set of polynomials which corresponds to all zero substitutions of $F(\bar{x}+\bar{\sigma})$ serves as the set $\mathcal{P}$ in the application of Lemma 3.7. This, in turn, implies that G_w hits $F(\bar{x}+\bar{\sigma})$, since it is a member of this set of polynomials.

Theorem 10. *Let $F = c + \sum_{i=1}^{m} F_i$, where c is a constant, and each F_i is a non-constant multilinear sparse-substituted read-k_i formula. If $\bar{\sigma}$ is a common nonzero of the nonzero formulas of the form $\partial_P f|_{Z\leftarrow 0}$ where f is a subformula of the F_i's and $|P \cup Z| \leq b \doteq (k-m+1)\cdot 4k \cdot R(k+2)\cdot(\log(t)+1)$, then*

$$F \not\equiv 0 \Rightarrow F(G_w + \bar{\sigma}) \not\equiv 0$$

for $w \geq (8k\cdot R(k+2))^{k-m+1}(\log(t)+1)$, where $k \doteq \sum_{i=1}^{m} k_i$, t denotes the maximum number of terms in each substituted polynomial, and R is the function given by Lemma 3.8.

Proof. Define the classes of formulas

$$\mathcal{F} \doteq \{F|_{S\leftarrow\bar{\sigma}} \mid S \subseteq [n]\}, \text{ and } \mathcal{F}' \doteq \{f(\bar{x}+\bar{\sigma}) \mid f \in \mathcal{F}\}.$$

Observe that $\mathcal{F}'$ is closed under zero substitutions because $f(\bar{x}+\bar{\sigma})|_{x\leftarrow 0} = f|_{x\leftarrow\sigma}(\bar{x}+\bar{\sigma})$ and $\mathcal{F}$ is closed under substitutions by $\bar{\sigma}$.

Without loss of generality each $f \in \mathcal{F}$ has at most one top level branch which is constant, since constant branches can be collected into a single constant branch without compromising any of the relevant properties of f. Observe that for each $f \in \mathcal{F}$, the $\bar{\sigma}$ remains a common nonzero subformulas of f under at least b partial derivatives and zero substitutions, because we are performing a partial substitution of $\bar{\sigma}$ itself. Therefore, for each $f \in \mathcal{F}$, the preconditions of Lemma 3.18 are met and hence $f(\bar{x}+\bar{\sigma}) \notin \mathcal{D}_n$, for $n \geq w$.

This implies that $\mathcal{F}'$ is disjoint from $\mathcal{D}_n$, for $n \geq w$. Lemma 3.7 then says that G_w hits $\mathcal{F}'$, and $F(\bar{x} + \bar{\sigma})$ in particular. ∎

3.7.3 Non-Blackbox Reduction

In this subsection we focus on giving a non-blackbox reduction from identity testing multilinear sparse-substituted $\sum^m$-read-k formulas to identity testing multilinear sparse-substituted read-k formulas. The first step of the reduction is to compute an appropriate shift $\bar{\sigma}$ using an identity testing algorithm for multilinear sparse-substituted read-k formulas. A technical complication is to ensure that the formula has gates that are explicitly multilinear, so that partial derivatives can be computed efficiently. This can also be done using an identity test for multilinear sparse-substituted read-k formulas. Once we have $\bar{\sigma}$, we simply evaluate $F(G_w + \bar{\sigma})$ on sufficiently many points and see whether we obtain a nonzero value.

Lemma 3.19 ($\sum^m$-Read-k PIT $\leq$ Read-k PIT – Non-Blackbox). *For any integer $k \geq 1$, given a deterministic identity testing algorithm for multilinear sparse-substituted read-k formulas that runs in time $T(k, n, s, t)$, there is a deterministic algorithm that tests multilinear sparse-substituted $\sum^m$-read-k formulas that runs in time*

$$k^2 m^2 n^{O(b)} \cdot T(k, n, s, t) + n^{O(w_{m,k} \cdot (\log(t)+1))} \operatorname{poly}(k, n, s, t),$$

where s denotes the size of the formulas, n the number of variables, and t the maximum number of terms in each substituted polynomial, $b \doteq ((k-1)m+1) \cdot 4km \cdot R(km+2) \cdot (\log(t)+1)$, $w_{m,k} \doteq (8km \cdot R(km+2))^{(k-1)m+1}$, and R is the function given by Lemma 3.8.

Proof. Let $F \doteq \sum_{i=1}^{m} F_i$, where each F_i is a multilinear sparse-substituted read-k formula. Let b be sufficient to apply Theorem 10 with the parameters $k_i = k$, m, and n.

Process each of the F_i from the bottom up, making the children of multiplication gates variable disjoint. To do this, at each gate g compute the set of variables that g depends on. This can be done using the hypothesized identity test on the first order partial derivatives of g with respect to each variable. These partial derivatives can be efficiently computed as the children have been previously processed to have variable disjoint multiplication gates. Set variables that g does not depend on to 0, though only within the subformula g. Note, that this does not affect the polynomial computed at each gate of F_i; it merely removes extraneous variable occurrences. As we can assume F to be in standard form, each F_i has at most $O(kn)$ gates and this step uses at most $O(kmn^2)$ applications of the identity test.

Let $\mathcal{F}$ be the set of all nonzero formulas of the form $\partial_P f|_{Z \leftarrow 0}$ where f is a subformula of one of the F_i's and $|P \cup Z| \leq b$. Notice that the elements of $\mathcal{F}$ are multilinear sparse-substituted read-k formulas because each f is of that type and that type of formulas is closed under partial derivatives and substitutions.

There are at most $O(kmn)$ gates in F, thus $|\mathcal{F}| = O(kmn^{b+1})$. The formulas in $\mathcal{F}$ can be efficiently enumerated. To see this, observe that for a choice of a gate g in F_i, and sets P and Z, the formula $\partial_P g|_{Z \leftarrow 0}$ can be computed in time polynomial in the size of F, because we preprocessed the multiplication gates of F to be variable disjoint. Further, we can determine efficiently whether each of these formulas is nonzero using the hypothesized identity test for multilinear sparse-substituted read-k formulas.

Define the polynomial, $\Phi \doteq \prod_{f \in \mathcal{F}} f$. Since $\Phi \not\equiv 0$, there is a point in a finite extension $\mathbb{E}^n \supseteq \mathbb{F}^n$ that witnesses the non-zeroness of Φ. We can use trial substitution to determine a point, $\bar{\sigma} \in \mathbb{E}^n$ where Φ is nonzero. F is multilinear and all the formulas in $\mathcal{F}$ are multilinear as well. This means that Φ has total degree at most $O(kmn^{b+2})$. By the Schwartz-Zippel Lemma we only need to test elements from a subset $W \subseteq \mathbb{E}$ of size at most the degree of Φ plus one (i.e., a set of size $O(kmn^{b+2})$). For each variable, in turn, determine a value from W that keeps Φ nonzero. Fix the variable to this value, and then move on to consider the

next variable. This uses $O(kmn^{b+3})$ identity tests on a partially substituted version of Φ. Each of these identity tests uses $O(kmn^{b+1})$ identity tests on the component read-k formulas. In total, our algorithm uses $O(k^2m^2n^{2b+4})$ identity tests on multilinear sparse-substituted read-k formulas to compute $\bar{\sigma}$. This can be completed in $O(k^2m^2n^{2b+4}T(k,n,s,t))$ time, using the assumed identity test.

Using Theorem 10 gives that $G_{w_{m,k}\cdot(\log t+1)}$ hits $F(\bar{x}+\bar{\sigma})$. Therefore, $F \equiv 0$ iff $F(G_w+\bar{\sigma}) \equiv 0$. By multilinearity, the formula F has degree at most n and, by definition, G_w has degree at most n. Applying Proposition 3.2 gives a test for $F(G_w+\bar{\sigma})$ that runs in time $O((n^2)^{2w})$. This completes the identity test. The cost of performing this part of the algorithm is at most $n^{O(w)} \cdot \text{poly}(k,m,s,t)$. Combining this with the preprocessing and the computation of $\bar{\sigma}$ gives the total running time claimed. ∎

3.7.4 Blackbox Reduction

We now describe a blackbox version of Lemma 3.19. The overall approach is the same as in Section 3.7.3, though the details are somewhat simpler. With Theorem 10 in hand, all that remains is to leverage a generator $\mathcal{G}$ for multilinear sparse-substituted read-k formulas to generate an appropriate shift $\bar{\sigma}$ and then apply the theorem to complete the reduction.

Let $F = \sum_{i=1}^{m} F_i$ be a multilinear sparse-substituted $\sum^m$-read-k formula. Let $\mathcal{F}$ be the set of all nonzero formulas of the form $\partial_P f|_{Z\leftarrow 0}$ where f is a subformula of the F_i's and P, Z are disjoint sets of variables with $|P \cup Z| \leq b$. $\mathcal{F}$ is composed of nonzero multilinear sparse-substituted read-k formulas. Therefore, $\mathcal{G}$ hits the product $\prod_{f\in\mathcal{F}} f$, and a suitable shift $\bar{\sigma}$ is in the image of $\mathcal{G}$. Applying Theorem 10 then gives that $\mathcal{G} + G_w$ is a generator for multilinear sparse-substituted $\sum^m$-read-k formulas, where w is bounded as in the theorem.

Lemma 3.20 ($\sum^m$-Read-k PIT $\leq$ Read-k PIT – Blackbox). *For an integer $k \geq 1$, let $\mathcal{G}$ be a generator for n-variate multilinear sparse-substituted read-k formulas. Then*

$\mathcal{G} + G_{w_{m,k}\cdot(\log(t)+1)}$ is a generator for n-variate multilinear sparse-substituted $\sum^m$-read-k formulas, where $w_{m,k} \doteq (8km \cdot R(km+2))^{(k-1)m+1}$, t denotes the maximum number of terms in each substituted polynomial, and R is the function given by Lemma 3.8.

Proof. Let F be a multilinear sparse-substituted $\sum^m$-read-k formula. Write $F \doteq \sum_{i=1}^m F_i$, where each F_i is a multilinear sparse-substituted read-k formula. Let $b \doteq ((k-1)m + 1) \cdot 4km \cdot R(km+2) \cdot (\log(t)+1)$ and $w \doteq w_{m,k} \cdot (\log(t)+1)$; in other words, sufficient parameters for applying Theorem 10 with m and $k_i = k$.

Let $\mathcal{F}$ be the set of all nonzero formulas of the form $\partial_P f|_{Z \leftarrow 0}$ where f is a subformula of the F_i's and P, Z are disjoint sets of variables with $|P \cup Z| \leq b$. Consider the polynomial $\Phi \doteq \prod_{f \in \mathcal{F}} f$. Note that $\Phi \not\equiv 0$, and that $f \in \mathcal{F}$ is multilinear sparse-substituted read-k formula with at most t terms in each substituted polynomial.

Since $\mathcal{G}$ is a generator for multilinear sparse-substituted read-k formulas and Φ is the product of multilinear sparse-substituted read-k formula; $\mathcal{G}$ hits Φ. There is a point with components in a finite extension $\mathbb{E} \supseteq \mathbb{F}$ that witnesses the non-zeroness of $\Phi \circ \mathcal{G}$. Let $\bar{\beta}$ be such a point and define $\bar{\sigma} \doteq \mathcal{G}(\bar{\beta})$. Thus $\Phi(\bar{\sigma}) \neq 0$. This implies that all formula in $\mathcal{F}$ do not vanish at $\bar{\sigma}$. By Theorem 10, $G_{w_{m,k}\cdot(\log(t)+1)}$ hits $F(\bar{x} + \bar{\sigma})$. Finally, since $\bar{\sigma}$ is in the image of $\mathcal{G}$, $\mathcal{G} + G_{w_{m,k}\cdot(\log(t)+1)}$ hits F, completing the reduction. ∎

3.7.5 From Multilinear to Structurally-Multilinear Formulas

In this subsection we exhibit a transformation $\mathcal{L}$ that takes a structurally-multilinear formula and produces a multilinear sparse-substituted formula while preserving (non-)zeroness. Combining this transformation with the non-blackbox reduction in Section 3.7.3 allows us to prove the non-blackbox part of Theorem 8 in Section 3.8.1. Additionally, the transformation induces a natural generalization of the condition (from Lemma 3.18) on $\bar{\sigma}$ as the common nonzero of some larger set of formulas. This allows us to generalize the Key

Lemma (Lemma 3.18) to structurally-multilinear formulas, and then argue an analog of the blackbox reduction from Section 3.7.4 for structurally-multilinear formulas.

We begin by formally defining the transformation $\mathcal{L}$. Next, we observe some useful properties of $\mathcal{L}$, and, finally, we use $\mathcal{L}$ to generalize the Key Lemma for structurally-multilinear formulas.

3.7.5.1 The Transformation $\mathcal{L}$

For set of variables $X \doteq \{x_1, \ldots, x_n\}$ we define $\mathfrak{X} \doteq \left\{x_\ell^j \mid \ell, j \geq 1\right\}$ to be the set of all positive powers of the variables in X. Consider a new set of variables $Y \doteq \{y_{\ell,j} \mid \ell, j \geq 1\}$, and observe that there is a bijection between $\mathfrak{X}$ and Y. The transformation $\mathcal{L}$ maps elements of $\mathfrak{X}$ into variables of Y in a natural way.

Definition 3.7 (The transformation $\mathcal{L}$).
Let $X \doteq \{x_1, \ldots, x_n\}$ and $Y \doteq \{y_{\ell,j} \mid \ell, j \geq 1\}$. Let $f \in \mathbb{F}[X]$ be a sparse-substituted formula.

- *For $\ell, j \geq 1$, let $\mathcal{L}_{\left\{x_\ell^j\right\}}(f)$ be the result of replacing every occurrence of exactly x_ℓ^j in each term of a sparse-substituted input of f by the variable $y_{\ell,j}$.*
- *Let A be a set of positive powers of variables in X. Let $\mathcal{L}_A(f)$ be the result of applying $\mathcal{L}_{\left\{x_\ell^j\right\}}$ to f for all $x_\ell^j \in A$. Furthermore, let $\mathcal{L}(f)$ denote the result of taking A to be the set of all positive powers of every variable in X, e.g., the result of replacing all positive powers of x_ℓ-variables by the corresponding $y_{\ell,j}$'s.*
- *For any set $P \subseteq Y$, let $X(P) \doteq \left\{x_\ell^j \mid y_{\ell,j} \in P\right\}$ be the preimages of the $y_{\ell,j}$'s under $\mathcal{L}$.*

For concreteness we give a few examples of the transformation $\mathcal{L}$ being applied to structurally-multilinear formulas:

$$\begin{aligned} f = x_1^2 x_3 &\mapsto \mathcal{L}(f) = y_{1,2} y_{3,1}, \\ f = (x_1^2 x_3 + x_1 x_3^6) \cdot (x_2^3 x_4 + 3) &\mapsto \mathcal{L}(f) = (y_{1,2} y_{3,1} + y_{1,1} y_{3,6}) \cdot (y_{2,3} y_{4,1} + 3). \end{aligned}$$

The following lemma demonstrates the connection between a formula f and its transformation $\mathcal{L}(f)$. The lemma exploits the fact that in a structurally-multilinear formula variables are never multiplied with themselves outside a sparse-substituted input. This implies that we can treat each degree of x_ℓ as if it is a distinct variable. Additionally, we observe that setting $x_\ell \leftarrow a$ in f, for some $a \in \mathbb{F}$, is equivalent to setting $\{y_{\ell,j} \leftarrow a^j \mid j \geq 1\}$ in $\mathcal{L}(f)$.

Lemma 3.21. *Let $f \in \mathbb{F}[X]$ be a structurally-multilinear sparse-substituted read-k formula. Let $P, Z \subseteq Y$ be two disjoint subsets of variables and let $\bar{\sigma} \in \mathbb{F}^n$ be an assignment. Then the following holds:*

1. *$\mathcal{L}(f)$ is a multilinear sparse-substituted read-k formula.*
2. *$f \equiv 0$ if and only if $\mathcal{L}(f) \equiv 0$.*
3. *$\partial_P(\mathcal{L}_{X(P \cup Z)}(f))|_{Z \leftarrow 0}$ does not depend on any $y_{\ell,j}$.*
4. *$(\partial_P(\mathcal{L}(f))|_{Z \leftarrow 0})\,|_{\left\{y_{\ell,j} \leftarrow \sigma_\ell^j \mid \ell, j \geq 1\right\}} = \left(\partial_P(\mathcal{L}_{X(P \cup Z)}(f))|_{Z \leftarrow 0}\right)|_{\{x_\ell \leftarrow \sigma_\ell \mid \ell \geq 1\}}$*

Proof. We first demonstrate a useful property of $\mathcal{L}$, and then show that it implies the properties stated in the lemma. Consider a term $T = c \cdot \prod_{\ell=1}^{n} x_\ell^{d_\ell}$ in the expansion of f. Each such term is produced by the sum of various products of terms from the sparse-substituted inputs:

$$T \equiv \sum_i c_i \prod_{\ell=1}^{n} x_\ell^{d_\ell} = \sum_i \prod_j T_{ij},$$

where each T_{ij} is a term from a sparse-substituted input. We can assume that for each i, the terms T_{ij} are all from different sparse-substituted inputs. Since f is structurally multilinear, for each i, the terms T_{ij} are variable disjoint, and hence each variable may occur in at most one factor T_{ij}.

Consider $\mathcal{L}_{x_\ell^d}(T)$. If $x_\ell^d \mid T$, but $x_\ell^{d+1} \nmid T$, then $\mathcal{L}_{x_\ell^d}(T) = y_{\ell,d} \cdot T/x_\ell^d$. Otherwise $\mathcal{L}_{x_\ell^d}(T) = T$. This is a 1-1 mapping on terms, and linearly extends to the sum of terms forming the expansion of a structurally-multilinear sparse-substituted formula f. Moreover, for any set of variable powers A, $\mathcal{L}_A$ maps the terms of a structurally-multilinear sparse-substituted formula in a 1-1 way.

We now prove the properties claimed by the lemma.

Part 1. $\mathcal{L}(f)$ is multilinear, because for each term and variable power in the expansion of f, the exact variable power x_ℓ^d is replaced by a $y_{\ell,d}$. $\mathcal{L}(f)$ is a multilinear sparse-substituted formula because the transformation is performed on each sparse-substituted input individually. $\mathcal{L}(f)$ is read-k because each $y_{\ell,d}$ occurs in no more sparse-substituted inputs of $\mathcal{L}(f)$ than x_ℓ does in f.

Part 2. We demonstrated that $\mathcal{L}$ induces a 1-1 correspondence between the terms of f and $\mathcal{L}(f)$. Moreover, nonzero terms are mapped to nonzero terms. Hence $f \equiv 0$ iff $\mathcal{L}(f) \equiv 0$.

Part 3. By definition the y variables in $\mathcal{L}_{X(P\cup Z)}(f)$ are in $P \cup Z$. The conclusion follows because partial derivatives and substitutions eliminate all dependence on the variables they act over.

Part 4. This property follows from two claims, which hold for any structurally-multilinear sparse-substituted formula g:

(i) $\mathcal{L}(g)|_{\left\{y_{\ell,j} \leftarrow \sigma_\ell^j \mid \ell,j\geq 1\right\}} \equiv g|_{\{x_\ell \leftarrow \sigma_\ell \mid \ell \geq 1\}}$, and

(ii) $\partial_P \mathcal{L}(g)|_{Z\leftarrow 0} \equiv \mathcal{L}(\partial_P \mathcal{L}_{X(P\cup Z)}(g)|_{Z\leftarrow 0})$.

Claim (i) follows immediately from the 1-1 mapping between terms of g and $\mathcal{L}(g)$ established above. To see claim (ii) we argue that for all constants c:

$$\mathcal{L}(g)|_{y_{\ell,d}\leftarrow c} \equiv \mathcal{L}(\mathcal{L}_{x_\ell^d}(g)|_{y_{\ell,d}\leftarrow c}). \tag{3.2}$$

This essentially says that substitutions for y variables can be moved ahead of most of the transformation done by $\mathcal{L}$. Consider a term T in the expansion of g. If $x_\ell^d \mid T$, but $x_\ell^{d+1} \nmid T$, then $\mathcal{L}_{x_\ell^d}(T) = y_{\ell,d} \cdot T/x_\ell^d$ and

$$\begin{aligned}\mathcal{L}(T)|_{y_{\ell,d}\leftarrow c} \equiv (y_{\ell,d} \cdot \mathcal{L}(\frac{T}{x_\ell^d}))|_{y_{\ell,d}\leftarrow c} \equiv c \cdot \mathcal{L}(\frac{T}{x_\ell^d}) \equiv \mathcal{L}(c \cdot \frac{T}{x_\ell^d}) \equiv \mathcal{L}((y_{\ell,d} \cdot \frac{T}{x_\ell^d})|_{y_{\ell,d}\leftarrow c}) \\ \equiv \mathcal{L}(\mathcal{L}_{x_\ell^d}(T)|_{y_{\ell,d}\leftarrow c}).\end{aligned}$$

Otherwise, $\mathcal{L}(T)$ does not depend on $y_{\ell,d}$, then $\mathcal{L}(T|_{y_{\ell,d}\leftarrow c}) = \mathcal{L}(T)$, and therefore T contributes equally to both sides of Equation (3.2). By linearity we have Equation (3.2). Claim (ii) follows by performing similar analysis for partial derivatives. This completes the proof of Part 4 and the lemma. ∎

3.7.5.2 Generalizing the Key Lemma

Recall that our goal is to prove a version of the Key Lemma that works with structurally-multilinear formulas. The statement of the generalization is almost identical to the original, except that $\bar{\sigma}$ must be the common nonzero of more formulas. The proof is via a reduction to Lemma 3.18, and is sketched in the following paragraph.

Let F be a structurally-multilinear formula. Suppose that $F(\bar{x} + \bar{\sigma}) \in \mathcal{T}_n$ for some assignment $\bar{\sigma}$. By a hybrid argument we show that for each variable x_ℓ there is a degree d_ℓ such that substituting the appropriate power of σ_ℓ into the variables $y_{\ell,j}$, for $j < d_\ell$, makes the multilinear sparse-substituted formula $\mathcal{L}(F)$ divisible by the linear polynomial $(y_{\ell,d_\ell} - \sigma_\ell^{d_\ell})$. Doing this to $\mathcal{L}(F)$ for each $\ell \in [n]$ produces a formula which is divisible by a

shifted monomial in the y_{ℓ,d_ℓ} variables. The only variables $y_{\ell,j}$ that remain have $j = d_\ell$, or $j > d_\ell$. The variables y_{ℓ,d_ℓ} will be the "x" variables when we apply the Key Lemma. We fix the variables $y_{\ell,j}$, for $j > d_\ell$, to a typical substitution, so that the relevant properties are preserved. The result is a multilinear sparse-substituted formula in the variables y_{ℓ,d_ℓ} which computes a shifted monomial. This allows us to reach a contradiction by applying Lemma 3.18.

The remaining question is: What are the conditions on $\bar{\sigma}$? $\bar{\sigma}$ must be a common nonzero of all the nonzero subformulas that may be considered by the Key Lemma when the above process is complete. However, we do not know *a priori* which choices our proof makes for the d_ℓ, and hence which variables remain when applying the Key Lemma. Therefore, we require that $\bar{\sigma}$ be a common nonzero with respect to all possible choices of the d_ℓ. In particular, we want $\bar{\sigma}$ to be the common nonzero of the nonzero $\partial_P(\mathcal{L}_{X(P\cup Z)}(f))|_{Z\leftarrow 0}$ where f is a subformula of F, and P and Z are sets of y variables. This way, independent of the choices the proof makes for the d_ℓ the conditions of the Key Lemma can be satisfied.

This intuition is formalized the following lemma.

Lemma 3.22 (Generalized Key Lemma). *Let $F = c + \sum_{i=1}^{m} F_i$, where c is a constant, and each F_i is a non-constant structurally-multilinear sparse-substituted read-k_i formula. If $\bar{\sigma}$ is a common nonzero of the nonzero formulas of the form $\partial_P(\mathcal{L}_{X(P\cup Z)}(f))|_{Z\leftarrow 0}$ where f is a subformula of the F_i's and $P, Z \subseteq Y \doteq \{y_{\ell,j} \mid \ell, j \geq 1\}$, $|P \cup Z| \leq b \doteq (k-m+1)\cdot 4k \cdot R(k+2)\cdot(\log(t)+1)$, then*

$$F(\bar{x} + \bar{\sigma}) \notin \mathcal{D}_n,$$

for $n \geq w \doteq (8k \cdot R(k+2))^{k-m+1}(\log(t)+1)$, where $k \doteq \sum_{i=1}^{m} k_i$, t denotes the maximum number of terms in each substituted polynomial, and R is the function given by Lemma 3.8.

Proof. Assume the contrary, without loss of generality, that $F(\bar{x} + \bar{\sigma}) \equiv Q \cdot M_n$ for some nonzero polynomial Q and $n \geq w$. Observe that for each $\ell \in [n]$, $F|_{x_\ell \leftarrow \sigma_\ell} \equiv 0$. Denote $\hat{F} = \sum_{i=1}^{m} \hat{F}_i \doteq \mathcal{L}(F)$. By Lemma 3.21, Part 2, and the definition of $\mathcal{L}$:

$$\begin{aligned} 0 &\equiv \mathcal{L}(F|_{x_\ell \leftarrow \sigma_\ell}) \\ &= \mathcal{L}(F)|_{\left\{y_{\ell,j} \leftarrow \sigma_\ell^j \mid j \geq 1\right\}} \\ &= \hat{F}|_{\left\{y_{\ell,j} \leftarrow \sigma_\ell^j \mid j \geq 1\right\}} \end{aligned}$$

As $\hat{F} \not\equiv 0$ and $\hat{F}|_{\left\{y_{\ell,j} \leftarrow \sigma_\ell^j \mid j \geq 1\right\}} \equiv 0$ there must exist $j' \geq 1$ such that

$$\hat{F}|_{\left\{y_{\ell,j} \leftarrow \sigma_\ell^j \mid j \in [j'-1]\right\}} \not\equiv 0$$

and has $(y_{\ell,j'} - \sigma_\ell^{j'})$ as a factor. By repeating this argument sequentially for every $\ell \in [n]$, and using the fact that substitutions on multilinear polynomials commute, we obtain a sequence $(d_1, \ldots, d_n) \in \mathbb{N}^n$ such that

$$\hat{F}' = \sum_{i=1}^{m} \hat{F}_i' \doteq \hat{F}|_{\left\{y_{\ell,j} \leftarrow \sigma_\ell^j \mid \ell \geq 1\,,\, j \in [d_\ell - 1]\right\}} \not\equiv 0.$$

Moreover, $\hat{F}'$ is a multilinear sparse-substituted m-sum of read-k_i formulas and

$$\hat{F}' \equiv Q' \cdot \prod_{\ell \in [n]} (y_{\ell,d_\ell} - \sigma_\ell^{d_\ell})$$

for some nonzero polynomial Q'. Partition $Y = \{y_{\ell,j} \mid \ell, j \geq 1\}$ into three sets depending on whether $j < d_\ell$, $j = d_\ell$, or $j > d_\ell$. Call these three sets $Y^<$, $Y^=$, and $Y^>$ respectively.

Consider a subformula $\hat{f}'$ of some $\hat{F}_i'$ and let f and $\hat{f}$, respectively, be the corresponding subformulas of F_i and $\hat{F}_i$. Let $P, Z \subseteq Y^=$ be such that $|P \cup Z| \leq b$ and $\partial_P \hat{f}'|_{Z \leftarrow 0} \not\equiv 0$. By

Lemma 3.21, Part 4, and the definition of $\bar{\sigma}$:

$$\left(\partial_P \hat{f}|_{Z\leftarrow 0}\right)|_{\left\{y_{\ell,j}\leftarrow \sigma_\ell^j \mid \ell,j\geq 1\right\}} = \left(\partial_P(\mathcal{L}_{X(P\cup Z)}(f))|_{Z\leftarrow 0}\right)|_{\{x_\ell\leftarrow\sigma_\ell \mid \ell\geq 1\}} \not\equiv 0.$$

The substitution on the LHS of the above equation can be partitioned corresponding to the sets $Y^<, Y^=$, and $Y^>$. We drop the substitutions associated with $Y^>$; this keeps the formula nonzero. Since $\hat{f}$ is multilinear, $P, Z \subseteq Y^=$, and $Y^=$ is disjoint from $Y^<$, the substitutions of variables from $Y^<$ commutes with partial derivatives on P and zero-substitutions on Z. This fact allows us to push the substitutions over $Y^<$ closer to $\hat{f}$ (to form $\hat{f}'$), and reach the following conclusion

$$\left(\partial_P \hat{f}'|_{Z\leftarrow 0}\right)|_{\left\{y_{\ell,d_\ell}\leftarrow \sigma_\ell^{d_\ell} \mid \ell\geq 1\right\}} \equiv \left(\partial_P \hat{f}|_{Z\leftarrow 0}\right)|_{\left\{y_{\ell,j}\leftarrow \sigma_\ell^{j} \mid \ell\geq 1\,,\, j\in[d_\ell]\right\}} \not\equiv 0.$$

This argument shows that substituting $\bar{\sigma}' \doteq (\sigma_\ell^{d_\ell})$ for $Y^=$ does not zero the formula $\partial_P \hat{f}'|_{Z\leftarrow 0}$. Moreover, this argument is generic with respect to the choice of $\hat{f}'$, P and Z, so the substitution $\bar{\sigma}'$ for $Y^=$ does not zero $\partial_P \hat{f}'|_{Z\leftarrow 0}$ for any subformula $\hat{f}'$ of $\hat{F}'$, and any choice of disjoint $P, Z \subset Y^=$ satisfying $|P \cup Z| \leq b$.

However, the resulting formulas are over $Y^> \cup Y^=$ not just $Y^=$. Observe that Q' may only depend on variables from $Y^>$ because $\hat{F}'$ is multilinear. Fix the variables of $Y^>$ so that $\bar{\sigma}'$ is a common nonzero of the formulas $\partial_P \hat{f}'|_{Z\leftarrow 0}$, and Q' is not zeroed (for this, a typical substitution suffices). Let $\hat{F}''$ be the result of applying this substitution to $\hat{F}'$. We have that

$$\hat{F}'' = a \cdot \prod_{\ell\in[n]} (y_{\ell,d_\ell} - \sigma'_\ell),$$

for some nonzero constant a, is a multilinear sparse-substituted formula over only the variables in $Y^=$. Set $\bar{x}' \doteq (y_{\ell,d_\ell})$ then $\hat{F}''(\bar{x}' + \bar{\sigma}')$ is a term of degree n. Furthermore, we

argued that for all subformulas $\hat{f}''$ of $\hat{F}''$ and disjoint $P, Z \subset Y^{=}$, with $|P \cup Z| \leq b$, we have that $\bar{\sigma}'$ does not zero $\partial_P \hat{f}''|_{Z \leftarrow 0}$. Collect the constant branches of $\hat{F}''$ into a single constant branch; the resulting formula satisfies all preconditions of Lemma 3.18, and a contradiction immediately follows. ■

Note, that if the given structurally-multilinear formula is in fact a multilinear formula, then the conditions of the lemma are equivalent to the conditions of Lemma 3.18 (up to a relabeling of the variables). Also note the proof of Lemma 3.22 only used sets P and Z that are disjoint, and that contain at most one y variable that corresponds to each x_ℓ, so we could have relaxed the statement of the lemma accordingly.

3.7.5.3 Blackbox Reduction

In this subsection we give a generalization of Lemma 3.20 to structurally-multilinear formulas. We first observe that Theorem 10 generalizes to structurally-multilinear formula.

Theorem 11. *Let $F = c + \sum_{i=1}^{m} F_i$, where c is a constant, and each F_i is a non-constant structurally-multilinear sparse-substituted read-k_i formula. If $\bar{\sigma}$ is a common nonzero of the nonzero formulas of the form $\partial_P(\mathcal{L}_{X(P \cup Z)}(f))|_{Z \leftarrow 0}$ where f is a subformula of the F_i's and $P, Z \subseteq \{y_{\ell,j} \mid \ell, j \geq 1\}$, $|P \cup Z| \leq b \doteq (k - m + 1) \cdot 4k \cdot R(k+2) \cdot (\log(t) + 1)$, then*

$$F \not\equiv 0 \Rightarrow F(G_w + \bar{\sigma}) \not\equiv 0,$$

for $w \geq (8k \cdot R(k+2))^{k-m+1}(\log(t)+1)$, where $k \doteq \sum_{i=1}^{m} k_i$, t denotes the maximum number of terms in each substituted polynomial, and R is the function given by Lemma 3.8.

Proof. Observe that the statement of this theorem is the same as Theorem 10 except that it takes on the conditions associated with Generalized Key Lemma (Lemma 3.22). To prove

this theorem follow the proof of Theorem 10, but use the stronger preconditions to apply Lemma 3.22 instead of Lemma 3.18. ∎

With Theorem 11 in hand, we can argue the following generalization of Lemma 3.20. The statement is identical to the original except that "multilinear sparse-substituted" is replaced by "structurally-multilinear sparse-substituted".

Lemma 3.23 ($\sum^m$-Read-k PIT $\leq$ Read-k PIT – Blackbox). *For an integer $k \geq 1$, let $\mathcal{G}$ be a generator for n-variate structurally-multilinear sparse-substituted read-k formulas. Then $\mathcal{G}+G_{w_{m,k}\cdot(\log(t)+1)}$ is a generator for n-variate structurally-multilinear sparse-substituted $\sum^m$-read-k formulas, where $w_{m,k} \doteq (8km \cdot R(km+2))^{(k-1)m+1}$, t denotes the maximum number of terms in each substituted polynomial, and R is the function given by Lemma 3.8.*

Proof. The proof is the same as in the original version except that Theorem 11 is applied instead of Theorem 10. This means that the class $\mathcal{F}$ of polynomials which $\bar{\sigma}$ is a common nonzero of must be larger to account for the stronger preconditions of the theorem. ∎

3.8 Identity Testing Read-k Formulas

Before moving on to prove our main theorems, we briefly stop to recall the overall approach. For clarity we only state the non-blackbox approach; the blackbox approach follows a similar pattern. We construct an identity test for structurally-multilinear read-k formulas using four tools.

Lemma 3.15 – a reduction from identity testing multilinear sparse-substituted read-$(k+1)$ formulas to identity testing multilinear sparse-substituted $\sum^2$-read-k formulas.

Lemma 3.19 – a reduction from identity testing multilinear sparse-substituted $\sum^2$-read-k formulas to identity testing multilinear sparse-substituted read-k formulas.

Lemma 3.24 – an identity test for multilinear sparse-substituted read-once formulas.

Lemma 3.21 (Parts 1 & 2) – a reduction from identity testing of structurally-multilinear read-k formulas to identity testing multilinear sparse-substituted read-k formulas.

Observe that combining the first two reductions reduces identity testing multilinear sparse-substituted read-$(k+1)$ formulas to identity testing multilinear sparse-substituted read-k formulas. Applying this observation recursively and combining it with Lemma 3.24 as the base case, establishes our main theorem – an identity test for multilinear sparse-substituted read-k for arbitrary k. We then plug in Lemma 3.21 to lift this result to structurally-multilinear formulas. Lemma 3.24 and its corresponding blackbox version are proved in the following subsections immediately before the corresponding main theorem. In the blackbox setting we deal directly with structurally-multilinear formulas. In the last subsection we develop a specialized blackbox identity test for structurally-multilinear sparse-substituted read-k formulas of constant depth.

3.8.1 Non-Blackbox Identity Test

We begin by describing a simple identity test for multilinear sparse-substituted read-once formula. Note that here *multilinear* is not a redundant qualifier because the sparse substitutions could make the read-once formula non-multilinear.

Lemma 3.24. *There is a deterministic algorithm for identity testing multilinear sparse-substituted read-once formulas that runs in time* $\mathrm{poly}(n, s, t)$*, where* s *denotes the size of the formula,* n *the number of variables, and* t *the maximum number of terms in each substituted polynomial.*

Proof. First, consider how to identity test sparse polynomials. Examine the list of terms and merge duplicate monomials. The sparse polynomial is nonzero iff any term remains

(with a nonzero coefficient). Notice that the same procedure can be used to determine whether a sparse polynomial is constant. This process takes time polynomial in the number of variables and the bound on the sparsity of the substitutions.

To identity test multilinear sparse-substituted read-once formulas, first apply the above procedure to each sparse substitution. If a sparse substitution is non-constant, replace it with a unique new variable; otherwise, replace it with the constant value. The resulting formula is read-once because each variable only occurs in one sparse input. This procedure does not affect the (non-)zeroness of the formula. Moreover, the procedure runs in time polynomial in the size of the formula and reduces the problem to testing read-once formulas. There can be no additive cancellation of variables in read-once formulas, therefore the only way for such a formula to be zero is if it is multiplied by the constant zero. Thus, (non-)zeroness can be determined by traversing the read-once formula from the bottom up, simplifying gates over constants and eliminating gates that have a multiplication by zero.■

Combining Lemmas 3.15, 3.19, and 3.24 in the way suggested above proves the following main result.

Theorem 12. *There exists a deterministic polynomial identity testing algorithm for multilinear sparse-substituted formulas that runs in time* $s^{O(1)} \cdot n^{k^{O(k)}(\log(t)+1)}$*, where* s *denotes the size of the formula,* n *the number of variables,* k *the maximum number of substitutions in which a variable appears, and* t *the maximum number of terms a substitution consists of.*

Proof. We proceed by induction on k. The base case is $k = 1$, which is handled by the identity test from Lemma 3.24. Consider the induction step for arbitrary $k + 1$. Assume there is an identity test for multilinear sparse-substituted read-k formulas that runs in time $T(k, n, s, t)$. Lemma 3.19 implies there is a deterministic algorithm that runs in time $k^2 n^b \cdot T(k, n, s, t) + n^w \cdot \text{poly}(k, n, s, t))$ that tests multilinear sparse-substituted $\sum^2$-read-k formulas. The lemma bounds $b = O(k^4 \log k \cdot (\log(t) + 1))$ and $w = k^{O(k)}(\log(t) + 1)$. Given

this identity test for multilinear sparse-substituted $\sum^2$-read-k formulas, Lemma 3.15 results in an identity test for multilinear sparse-substituted read-$(k+1)$ formulas that runs in time $T(k+1,n,s,t) = O(kn^2(k^2n^b \cdot T(k,n,s,t) + n^w \cdot \text{poly}(k,n,s,t)) + \text{poly}(k,n,s,t))$. Solving this recurrence results in the bound claimed. ∎

This gives an identity test for structurally-multilinear bounded-read formulas.

Proof (of Theorem 8 – Non-blackbox). Consider a structurally-multilinear read-k formula F. In time polynomial in the size of F compute $\mathcal{L}(F)$. By Lemma 3.21, Parts 1 and 2, $\mathcal{L}(F)$ is a multilinear sparse-substituted read-k formula, and $\mathcal{L}(F) \equiv 0$ iff $F \equiv 0$. Determine whether $\mathcal{L}(F)$ is zero by applying the algorithm from Theorem 12. ∎

This proves the non-blackbox part of Theorem 8, and yields the following corollary for constant read.

Corollary 3.3. *There exists a deterministic polynomial identity testing algorithm for structurally-multilinear sparse-substituted constant-read formulas that runs in time $s^{O(1)} \cdot (nd)^{O(\log t)}$, where s denotes the size of the formula, n the number of variables, t the maximum number of terms a substitution consists of, and d the maximum degree of individual variables in the substitutions.*

When t is constant the algorithm is runs in polynomial time. In particular, we obtain the following corollary.

Corollary 3.4. *There exists a deterministic polynomial-time algorithm for identity testing multilinear constant-read formulas.*

Using transformations different from $\mathcal{L}$ (Definition 3.7) it is possible to attain alternate (often incomparable) running-time parameterizations in the main theorem.

3.8.2 Blackbox Identity Test

We proceed analogously to the previous subsection. We first argue that the SV-generator G works for structurally-multilinear sparse-substituted read-once formulas – this extends the argument in [SV09], which worked for read-once formulas. Additionally, the argument is stated with respect to a depth parameter to make a later specialization to constant-depth more concise.

The idea is the following. We recurse on the structure of the structurally-multilinear sparse-substituted read-once formula F and argue that the SV-generator takes non-constant subformulas to non-constant subformulas. There are three generic cases, based on the top gate of F: (i) addition, (ii) multiplication, and (iii) a sparse-substituted input.

In case (i), the fact that F is read-once implies that addition branches are variable disjoint. This means that there is a variable whose partial derivative eliminates at least half of the formula and reduces the depth by one. Combining this fact with Lemma 3.5 completes the case. In case (ii), the fact that the SV-generator takes non-constant subformulas to non-constant subformulas immediately implies that if G hits the children of a multiplication gate it also hits the gate itself. In case (iii) we can immediately conclude using Lemma 3.6.

Lemma 3.25. *Let F be a nonzero structurally-multilinear sparse-substituted depth-D read-once formula. Then G_w hits F for $w \doteq \min\{\lceil \log |var(F)| \rceil, D\} + \lceil \log t \rceil + 1$, where t denotes the maximum number of terms in each substituted polynomial. Moreover, if F is non-constant then so is $F \circ G_w$.*

Proof. We proceed by structural induction on F. When F is constant, $F \circ G_w = F$ and the lemma trivially holds. When F is a non-constant sparse-substituted input with t terms, $F \circ G_w$ is non-constant for $w > \lceil \log t \rceil + 1$ by Lemma 3.6. In the induction step F is non-constant and not a sparse-substituted input. There are two induction cases.

Case 1: The top gate of F is an addition gate. Say $F = \alpha \cdot \sum_{i=1}^{m} F_i + \beta$, where the F_i are structurally-multilinear sparse-substituted depth-$(D-1)$ read-once formulas. Because F is in standard form, it has at least two non-constant branches F_1 and F_2. Then, because F is read-once: F_1 and F_2 are variable disjoint, without loss of generality $|\text{var}(F_1)| \leq \frac{|\text{var}(F)|}{2}$, and for any $x \in \text{var}(F_1)$ there exists $\gamma \in \bar{\mathbb{F}}$ such that $\partial_{x,\gamma} F = \partial_{x,\gamma}(\alpha \cdot \sum_{i=1}^{m} F_i + \beta) = \alpha \cdot \partial_{x,\gamma} F_1 \not\equiv 0$. Thus, $\partial_{x,\gamma} F$ has depth at most $D-1$ and depends on at most $\frac{|\text{var}(F)|}{2}$ variables. Observe that

$$\min\left\{\left\lceil \log \frac{|\text{var}(F)|}{2} \right\rceil, D-1\right\} + \lceil \log t \rceil + 1 = w - 1.$$

The induction hypothesis immediately gives that the $\partial_{x,\gamma} F \not\equiv 0$ is hit by G_{w-1}. Applying Lemma 3.5 implies that $F \circ (G_{w-1} + G_1)$ is non-constant. By the Proposition 3.3, Part 3, $G_{w-1} + G_1 = G_w$, completing this case.

Case 2: The top gate of F is a multiplication gate. Say $F = \alpha \cdot \prod_{i=1}^{m} F_i + \beta$, where the F_i are structurally-multilinear sparse-substituted depth-$(D-1)$ read-once formulas. The fact that F is in standard form implies that each F_i is non-constant. The induction hypothesis immediately implies that $G_{w'}$ hits each F_i, where $w' = \min\{\lceil \log |\text{var}(F)| \rceil, D-1\} + \lceil \log t \rceil + 1$. Further, each $F_i \circ G_{w'}$ is non-constant. Combining this with the fact that $w \geq w'$ implies that $\alpha \cdot (\prod_{i=1}^{m} F_i) \circ G_w + \beta$ is non-constant, completing this case. ∎

We formally conclude using Lemmas 3.16, 3.23, and 3.25 to prove the following main result.

Theorem 13. *For some function $w_k = k^{O(k)}$, the polynomial map $G_{w_k \cdot (\log(t)+1) + k \log n}$ is a hitting set generator for n-variate structurally-multilinear sparse-substituted read-k formulas, where t denotes the maximum number of terms a substitution consists of.*

Proof. We proceed by induction on k and argue that we can set w_k equal to the value $w_{2,k}$ from Lemma 3.23. The base case is immediate from Lemma 3.25. Consider the induction

step for arbitrary k. Assume that $\mathcal{G} \doteq G_{w_k \cdot (\log(t)+1)+k\log n}$ is a generator for structurally-multilinear sparse-substituted read-k formulas. Lemma 3.23 with $m = 2$ implies that $\mathcal{G} + G_{w_k \cdot (\log(t)+1)}$ is a generator for structurally-multilinear sparse-substituted $\sum^2$-read-k formulas. Apply Lemma 3.16 to $\mathcal{G}' \doteq \mathcal{G} + G_{w_k \cdot (\log(t)+1)}$. This gives that $G_{w_k \cdot (\log(t)+1)+k\log n} + G_{w_k \cdot (\log(t)+1)} + G_{\log n}$ is a generator for structurally-multilinear read-$(k+1)$ formulas. Apply the basic properties of G from Proposition 3.3, Part 3, to get that a total seed length of $2w_k \cdot (\log(t)+1) + (k+1)\log n$ suffices to hit structurally-multilinear read-$(k+1)$ formulas. As we can assume without loss of generality that $2w_k \leq w_{k+1}$, the theorem follows. ∎

A structurally-multilinear formula F on n variables, with individual degree d, has total degree at most dn. The SV-generator G_w with output length n has total degree at most n. Combining these facts and Proposition 3.2 with the previous theorem establishes the blackbox part of Theorem 8. In particular, it gives a quasi-polynomial-time blackbox algorithm for identity testing structurally-multilinear sparse-substituted constant-read formulas.

Corollary 3.5. *There exists a deterministic blackbox polynomial identity testing algorithm for structurally-multilinear sparse-substituted constant-read formulas that runs in time* $(dn)^{O(\log(n)+\log(t))}$ *and queries points from an extension field of size* $O(dn^2)$*, where* n *denotes the number of variables,* t *the maximum number of terms a substitution consists of, and* d *the maximum degree of individual variables in the substitutions.*

3.8.3 Special Case of Constant-Depth

We can improve the running time of our blackbox constant-read identity test by further restricting formulas to be constant-depth. We consider only the blackbox case because that is where we can get a substantial improvement. In the constant-depth setting we allow addition and multiplication gates that have arbitrary fanin. In order to specialize our previous argument to the constant depth case, we first give a version of the structurally-

multilinear Fragmentation Lemma (Lemma 3.13) parameterized with respect to the depth. We then carry through the different parameterization in Lemma 3.16 and Theorem 13.

Lemma 3.26 (Bounded-Depth Fragmentation Lemma). *Let $\emptyset \subsetneq V \subseteq [n]$, $k \geq 2$, and let F be a depth-D n-variable structurally-multilinear sparse-substituted read$_V$-k formula that depends on at least one variable in V. Let t denote the maximum number of terms in each substituted polynomial. There exists a variable $x \in V$ and $\alpha \in \bar{\mathbb{F}}$ such that $\partial_{x,\alpha} F$ is nonzero and is the product of*

1. *subformulas of F that have depth at most $D-1$, and*
2. *at most one structurally-multilinear sparse-substituted $\sum^2$-read$_V$-$(k-1)$ formula, which is the derivative with respect to x and α of some subformula of F.*

The proof of this lemma is quite similar to the original (Lemma 3.13), however, we make some different choices based on the depth.

Proof. Given the original Fragmentation Lemma, we only need to argue the second part. Assume without loss of generality that V only contains variables on which F depends, and that the children of multiplication gates are variable disjoint with respect to V.

If none of the variables in V occur k times in F, any choice of variable $x \in V$ does the job. So, let us assume that at least one variable in V occurs k times.

The algorithm recurses through the structure of F, maintaining the following invariant: The current gate being g visited, g, contains below it k occurrences of some variable in V. Setting g to be the output gate of F satisfies this invariant initially.

If g is a multiplication gate, recurse on a child of g that depends on a variable from V that occurs k times in g. Such a child must exist by the invariant and because F is multilinear.

If g is an addition gate, and at least one of its children, g_i, has a variable in V that occurs k times in g_i, recurse to g_i. Otherwise, select a variable $x \in V$ that occurs k times in g ending the recursion. In this case all of the children of g are structurally-multilinear sparse-substituted read$_V$-$(k-1)$ formulas. Since F depends on x, there is a $\alpha \in \bar{\mathbb{F}}$ such that $\partial_{x,\alpha} F$ is nonzero. Since g has at most k children that contain x, $\partial_{x,\alpha} g$ can be represented as a $\sum^k$-read$_V$-$(k-1)$ formula.

In the partial derivative $\partial_{x,\alpha} F$, all unvisited addition branches along the path from the output gate of F to the final g have been eliminated. Also, all unvisited multiplication branches along the path become factors of $\partial_{x,\alpha} F$ together with $\partial_{x,\alpha} g$. More formally, $\partial_{x,\alpha} F = (\partial_{x,\alpha} g) \prod_i F_i$, where the F_i are the unvisited multiplication branches. The F_i's are structurally-multilinear read$_V$-k formulas that have depth at most $D-1$, because they are the children of some multiplication gate in F. When the process stops at an addition gate, $\partial_{x,\alpha} g$ is a structurally-multilinear $\sum^k$-read$_V$-$(k-1)$ formula that may depend on many variables from V. ∎

Lemma 3.26 leads to the following variant of Lemma 3.16 in the bounded-depth setting.

Lemma 3.27. *For an integer $k \geq 1$, let $\mathcal{G}$ be a generator for n-variate structurally-multilinear sparse-substituted depth-D $\sum^{k+1}$-read-k formulas and let F be a nonzero n-variable structurally-multilinear sparse-substituted depth-D read-$(k+1)$ formula. Then $\mathcal{G} + G_D$ hits F.*

Proof. First observe that if F is read-k, we are immediately done because $F \circ \mathcal{G} \not\equiv 0$ and $\bar{0}$ is in the range of G (by the first item of Proposition 3.3).

The proof goes by induction on d. If $D = 0$, the lemma holds trivially as F is constant. If $D = 1$, F is a read-once formula, which is covered by the above observation. For the induction step, by the above observation we can assume that F is read-$(k+1)$ and not read-k. Therefore, F meets the conditions to apply the second part of the Fragmentation Lemma

for bounded depth formulas (Lemma 3.26). The lemma produces a variable $x \in \mathrm{var}(F)$ and $\alpha \in \bar{\mathbb{F}}$. The factors of $\partial_{x,\alpha} F$ all have depth at most $D-1$ and are structurally-multilinear read-$(k+1)$ formulas, except for at most one which might be a $\sum^{k+1}$-read-k formula. The induction hypothesis gives that the former factors of $\partial_{x,\alpha} F$ are all hit by $\mathcal{G} + G_{D-1}$. The latter factor (if it occurs) is hit by $\mathcal{G}$. Applying Lemma 3.5 gives that $\mathcal{G} + G_{D-1} + G_1$ hits F. Recalling Proposition 3.3, Part 3, implies that $\mathcal{G} + G_D$ hits F. ∎

We can use the previous lemma with Lemmas 3.23 and 3.25 to construct a hitting set generator specialized to bounded depth. The proof is almost identical to Theorem 13, except that fanin of the reduced instance increases to $k+1$ from 2. This weakens the parameterization of the seed length with respect to k.

Theorem 14. *For some function $w_k = k^{O(k^2)}$, the polynomial map $G_{w_k \cdot (\log(t)+1)+kD}$ is a hitting set generator for n-variate structurally-multilinear sparse-substituted depth-D read-k formulas, where t denotes the maximum number of terms a substitution consists of.*

Proof. We proceed by induction on k and argue that we can set w_k equal to the value $w_{k+1,k}$ from Lemma 3.23. The base case is immediate from Lemma 3.25. Consider the induction step for arbitrary k. Assume that $\mathcal{G} \doteq G_{w_k \cdot (\log(t)+1)+kD}$ is a generator for structurally-multilinear depth-D read-k formulas. Lemma 3.23 with $m = k+1$ implies that $\mathcal{G} + G_{w_k \cdot (\log(t)+1)}$ is a generator for structurally-multilinear depth-D $\sum^{k+1}$-read-k formulas. Apply Lemma 3.27 to $\mathcal{G}' \doteq \mathcal{G} + G_{w_k \cdot (\log(t)+1)}$. This gives that $G_{w_k \cdot (\log(t)+1)+kD} + G_{w_k \cdot (\log(t)+1)} + G_D$ is a generator for structurally-multilinear depth-D read-$(k+1)$ formulas. Apply the basic properties of G from Proposition 3.3, Part 3, to get that a total seed length of $2w_{k+1} \cdot (\log(t)+1) + (k+1)D$ suffices to hit structurally-multilinear depth-D read-$(k+1)$ formulas. As we can assume without loss of generality that $2w_k \leq w_{k+1}$, the theorem follows. ∎

Analogous to the unbounded depth setting, combining Theorem 14 with Proposition 3.2 establishes the following theorem.

Theorem 15 (Improvement for Bounded-Depth Formulas). *There exists a deterministic blackbox polynomial identity testing algorithm for structurally-multilinear sparse-substituted formulas with unbounded fanin that runs in time* $(dn)^{k^{O(k^2)}(\log(t)+1)+O(kD)}$ *and queries points from an extension field of size* $O(dn^2)$*, where* n *denotes the number of variables,* D *the depth of the formula,* k *the maximum number of substitutions in which a variable appears,* t *the maximum number of terms a substitution consists of, and* d *the maximum degree of individual variables in the substitutions.*

When the read of a formula is constant we obtain the following corollary.

Corollary 3.6. *There exists a deterministic blackbox polynomial identity testing algorithm for structurally-multilinear sparse-substituted constant-depth constant-read formulas that runs in time* $(dn)^{O(\log t)}$ *and queries points from an extension field of size* $O(dn^2)$*, where* n *denotes the number of variables,* t *the maximum number of terms a substitution consists of, and* d *the maximum degree of individual variables in the substitutions.*

The important difference between the above corollary and Corollary 3.5 is that the exponent no longer depends on n. Additionally, if the sparsity of substituted polynomials is constant the algorithm becomes polynomial-time. In particular, we obtain the following corollary.

Corollary 3.7. *There is a deterministic polynomial-time blackbox algorithm for identity testing multilinear constant-depth constant-read formulas.*

3.9 Further Research

The obvious big question remains: Are there polynomial-time deterministic identity tests for general arithmetic formulas? The more tenable open questions that are immediately raised by our work are: Can we get more efficient – polynomial-time – algorithms in the blackbox setting or in non-blackbox setting for sparse-substituted formulas? In fact, it is still open whether there is a polynomial-time blackbox test for read-once formulas. Multilinearity is used essentially in our arguments. Can this requirement be removed? Our work has spurred further improvements; a recent paper by Agrawal et al. provides partial answers to some of these questions for the special case of constant-depth: [ASSS11] give a polynomial-time blackbox test for constant-read constant-depth sparse-substituted formulas over fields of large characteristic. Their paper also gives more insight into the connections between the bounded-read and the bounded-top-fanin requirements of the past depth-three and -four results [SS11, SV11].

One direction where we have some partial results is in applying our techniques to arithmetic circuits of a more general type, namely, algebraic branching programs. Using our techniques, we can replicate the identity tests for read-once branching programs in [JQS10].

Another direction is *formula reconstruction*, that is, the problem of efficiently learning an arithmetic formula for a polynomial from examples given by blackbox. Shpilka and Volkovich do this efficiently for read-once formulas [SV08, SV09] using their identity tests for sums of read-once formulas. Can we leverage our identity tests and structural results for constant-read multilinear formulas to reconstruct formulas of the same type? Finally, it would be interesting to explore the hardness-randomness connection for constant-read formulas as has been done for other classes of formulas [KI04, DSY08].

4 Locality from Circuit Lower Bounds

The art of doing mathematics consists in finding that special case which contains all the germs of generality.

— David Hilbert

In this chapter we describe our results on the locality for first-order formulas. We begin with an overview of our techniques in Section 4.1. We discuss related work in Section 4.2. In Section 4.3 we present general background concerning neighborhoods and locality. In Section 4.4 we formally introduce our notion of Arb-invariance and describe its connection to AC^0. Section 4.5 contains our results for Gaifman locality, Section 4.6 our results for Hanf locality on strings, and Section 4.7 our application to regular languages. We conclude in Section 4.8 with some suggestions for further research.

4.1 Overview

Our proof of the *upper bound* on Gaifman locality in Theorem 3 exploits the tight connection between Arb-invariant FO formulas and the complexity class AC^0: Given an Arb-invariant FO formula φ that distinguishes two points of the universe whose neighborhoods up to distance r are of the same type, we construct a circuit on $2m = \Theta(r)$ bits that distinguishes inputs with exactly m ones from inputs with exactly $m + 1$ ones. In the special case where the neighborhoods of the two points are disjoint the circuit actually computes parity. The depth of the circuit is a constant depending on φ, and its size is polynomial in n. The known exponential circuit lower bounds [Hås86] then imply that r is bounded by a polylogarithmic function in n. This argument establishes the upper bound in Theorem 3 for the case of

formulas with a single free variable. In order to handle an arbitrary number k of free variables, we show how to reduce any case with $k > 1$ free variables to one with fewer variables in a way that is conceptually similar to (but technically different from) [GS00]. See Section 4.2 for a more detailed discussion of this related work.

As mentioned before, we do not know how to extend the upper bound of Theorem 3 to the stronger notion of Hanf locality in general, but we can establish it in Theorem 4 for the special case of strings. The reason the latter case is simpler is because on strings being Hanf local is equivalent to closure under swapping substrings whose endpoints have the same neighborhood type — a condition that has much of a Gaifman locality flavor. In fact, to prove that closure under such swaps holds for Arb-invariant FO($Succ$) formulas, we use a reduction to the upper bound for Gaifman locality from Theorem 3.

The *lower bounds* in Theorems 3 and 4 follow because arithmetic predicates like addition and multiplication allow one to define a bijection between the elements of a first-order definable set S of polylogarithmic size and an initial segment of the natural numbers [DLM07]. Thus, the binary representation of a single element of the entire domain can be used to represent a list of elements of S. By exploiting this, Arb-invariant FO can express, e.g., reachability between two nodes in S by a path of polylogarithmic length.

For the proof of Theorem 5 we use a characterization from [SS10b] stating that a regular language is definable in addition-invariant FO iff it is definable in FO($Succ, lm$) iff it is closed under two operations called "swap" and "transfer". By applying a pumping argument we obtain that Hanf locality and regularity imply closure under "swaps". Furthermore, a reduction from circuit lower bounds, similar to the one used for the upper bound proof of Theorem 3, along with a pumping argument shows that regular languages definable in Arb-invariant FO are closed under "transfers".

4.2 Related Work

We now give a brief overview of related work.

Invariant logics. The expressiveness of order-invariant FO was considered in various places, cf., e.g., [AHV95, Lib04, EF99, GS00, BS09]. Logics allowing invariant uses of predicates weaker than the linear order were considered in [Ros07, Ott00], concentrating on successor-invariant FO and epsilon-invariant FO, respectively. Logics allowing invariant uses of arbitrary numerical predicates were formally introduced in [Mak97], pointing out, in particular, that the graph properties definable in Arb-invariant FO are precisely the graph properties computable in AC^0. Similarly, [Mak98] showed that the graph properties definable in Arb-invariant least fixed-point logic coincide with the graph properties computable in P/poly. By results of [Imm87, Imm86, Var82] it is known that $(+, *)$-invariant FO (i.e., Arb-invariant FO where the formulas only use the numerical predicates $+$ and $\times$) and order-invariant least fixed-point logic precisely capture the graph properties computable in uniform AC^0 and in polynomial time, respectively.

Quite a number of articles in the circuit complexity literature and the finite model theory literature concentrated on graph properties (or queries) computable in AC^0 or definable in Arb-invariant FO (or variants thereof), without explicitly mentioning the notion of Arb-invariance. For example, [Raz85, And85, AB87] showed an exponential lower bound on the size of monotone circuits computing the k-clique problem on n-vertex graphs. Recently, [Ros08, Ros10] established a strong lower bound on the size of constant-depth circuits computing the k-clique problem, and applied this to show that the bounded variable hierarchy inside FO is strict on the class of finite ordered graphs and on the class of finite graphs enriched by arbitrary numerical predicates. [Ajt89] showed that the query selecting all pairs (x, y) of nodes in a graph that are connected by a path of length at most $f(n)$, where n is the size of the graph and f is an unbounded function, is not definable in Arb-invariant

FO. In [Ajt83] it was shown that the class of graphs having an even number of edges is not definable in the Arb-invariant version of the extension of FO called existential monadic second-order logic (EMSO). [FSV95, Sch96] proved that connectivity of graphs is not definable in EMSO with numerical predicates of moderate degree.

Locality. The notions of Hanf and Gaifman locality were introduced in [FSV95, HLN99], going back to results from [Han65, Gai82]. Showing that a logic is Hanf or Gaifman local provides insight into the limitations of its expressiveness and constitutes a high-level tool for proving that certain properties or queries cannot be expressed by formulas of this logic. Hanf and Gaifman locality results have been obtained for FO and for various extensions of FO (e.g., by counting quantifiers). For an overview on locality results and their applications in complexity theory we refer to [LN00]. Most locality results obtained in the literature deal with locality radii of constant size (cf., the example on FO mentioned at the beginning of the introduction, and the results mentioned in [HLN99, LN00, GS00, Lib04]). In their concluding sections, the articles [HLN99, GS00], however, proposed to also consider notions of locality where the radius of the neighborhoods grows with the size of the structures — this is what we do in the present paper. As pointed out in [HLN99, GS00], an analogue of our Theorem 3 for order-invariant first-order logic with counting quantifiers would lead to a separation of the complexity classes TC^0 and LOGSPACE.

The notion of locality in logic has a somewhat similar flavor to the notion of sensitivity in circuit complexity. The sensitivity of a Boolean function f at an input x is the number of bit positions i in x such that if we flip the ith bit in x, then the value of f changes. The average sensitivity of every function f in AC^0 over all inputs of length n is known to be polylogarithmically bounded in n [LMN93]. The latter result is closely related to the exponential lower bounds for parity on constant-depth circuits [Hås86]. Rather than going through sensitivity, our argument for proving Theorems 3 and 4 directly uses those circuit lower bounds to establish a polylogarithmic upper bound on the locality of Arb-invariant

FO.

Comparison with [GS00]. Our Theorem 3 can be viewed as an analogue of the main result of [GS00]. While their result states that *order*-invariant FO queries are Gaifman local with a *constant* locality radius r (depending on the query), our result states that *Arb*-invariant FO queries are Gaifman local with a locality radius $(\log n)^c$ where c is a constant (depending on the query) and n is the size of the underlying structure. Our proof of the upper bound in Theorem 3 has the same overall structure as the proof of [GS00]: It first considers queries of arity $k = 1$ for the case of disjoint neighborhoods, then for the case of overlapping neighborhoods, and afterwards it gives a reduction from queries of arbitrary arity $k > 1$ to queries of arity $k-1$. Our method for handling the case of overlapping neighborhoods uses techniques from [GS00]; however, our overall argument for treating queries of arity $k = 1$ gives a reduction to lower bounds in circuit complexity, while the argument of [GS00] relies on Ehrenfeucht-Fraïssé games. Our proof for the arity reduction from $k > 1$ to $k-1$ is conceptually similar to the proof of [GS00], but involves substantial technical differences. On the one hand, the notion of Arb-invariance allows us to give a non-uniform reduction (while the order-invariance of [GS00] requires uniformity). On the other hand, Arb-invariance requires us to construct a reduction that does not change the size of the universe of the structures considered (while the reduction of [GS00] changes the size of the universe and builds on the fact that this preserves order-invariance).

4.3 Background

In this section we briefly review relevant background material on neighborhoods, Gaifman locality, and Hanf locality. Recall that in Sections 2.3-2.4.1 we presented background on circuit complexity, finite model theory, and first-order logic.

Neighborhoods. To each structure M we associate an undirected graph $G(M)$, known as

the *Gaifman graph* of M, whose vertices are the elements of the domain of M and whose edges relate two elements of M whenever there exists a tuple in one of the relations of M in which both appear. For example, consider a relational schema τ consisting of one binary relation symbol E. Each τ-structure M is then a directed graph in the standard sense, and $G(M)$ coincides with M when ignoring the orientation. Given two elements u and v of a structure M, we denote as $dist^M(u, v)$ the distance between u and v in M, which is defined as their distance in the Gaifman graph $G(M)$. If $\bar{a}$ and $\bar{b}$ are tuples of elements of M, then $dist^M(\bar{a}, \bar{b})$ denotes the minimum distance between any pair of elements (one from $\bar{a}$ and one from $\bar{b}$).

For every $r \in \mathbb{N}$ and tuple $\bar{a} \in dom(M)^k$, the *r-ball around $\bar{a}$ in M* is the set

$$N_r^M(\bar{a}) \doteq \left\{ v \in dom(M) : dist^M(\bar{a}, v) \leq r \right\},$$

and the *r-neighborhood around $\bar{a}$ in M* is the structure

$$\mathcal{N}_r^M(\bar{a}) \doteq \left(M_{|N_r^M(\bar{a})}, \bar{a} \right).$$

That is, $\mathcal{N}_r^M(\bar{a})$ is the induced substructure of M on $N_r^M(\bar{a})$ with k distinguished elements $\bar{a}$. Two neighborhoods $\mathcal{N}_r^M(\bar{a})$ and $\mathcal{N}_r^{M'}(\bar{b})$ are isomorphic, if there is an isomorphism $\pi : M_{|N_r^M(\bar{a})} \cong M'_{|N_r^{M'}(\bar{b})}$ that maps $\bar{a}$ to $\bar{b}$.

Locality. Let $\phi(\bar{x})$ be a logical formula with k free variables. We consider two notions of locality of $\phi(\bar{x})$ (the precise definitions are basically taken from [Lib04]). We first define the notions with respect to *fixed* structures and then with respect to *all* structures.

Definition 4.1 (Gaifman Locality). *A formula $\phi(\bar{x})$ is* Gaifman r-local *with respect to a τ-structure M, if for all tuples $\bar{a}, \bar{b} \in dom(M)^k$ we have*

$$\mathcal{N}_r^M(\bar{a}) \cong \mathcal{N}_r^M(\bar{b}) \quad \Longrightarrow \quad M \models \phi(\bar{a}) \;\; \textit{iff} \;\; M \models \phi(\bar{b}). \tag{4.1}$$

For any two τ-structures M, M' and any tuples $\bar{a} \in dom(M)^k$ and $\bar{b} \in dom(M')^k$ we write $(M, \bar{a}) \equiv_r (M', \bar{b})$ if there is a bijection $h : dom(M) \to dom(M')$ such that for every element c in the domain of M, $\mathcal{N}_r^M(c\bar{a}) \cong \mathcal{N}_r^{M'}(h(c)\bar{b})$. Equivalently, the various *isomorphism types* (i.e., distinct local neighborhoods of radius r) occur with the same cardinality in M and M'.

Definition 4.2 (Hanf Locality). *A formula $\phi(\bar{x})$ is* Hanf r-local *with respect to a pair of τ-structures (M, M'), if for all tuples $\bar{a} \in dom(M)^k$ and $\bar{b} \in dom(M')^k$*

$$(M, \bar{a}) \equiv_r (M', \bar{b}) \quad \Longrightarrow \quad M \models \phi(\bar{a}) \;\; \textit{iff} \;\; M' \models \phi(\bar{b}). \tag{4.2}$$

For either notion of locality and every function $r : \mathbb{N} \to \mathbb{R}_{\geq 0}$, we call a formula $\phi(\bar{x})$ $r(n)$-*local* if there exists a constant n_ϕ such that $\phi(\bar{x})$ is $r(n)$-local with respect to all τ-structures of size $n \geq n_\phi$.

As for the relationship between the two notions of locality, there are two differences: (i) Hanf locality considers two structures that can be different, whereas Gaifman locality considers only one structure, and (ii) Hanf locality requires the existence of a global bijection, whereas Gaifman locality does not. Difference (i) makes Hanf locality a more powerful notion. In particular, Hanf locality is meaningful for sentences, whereas Gaifman locality for sentences trivially holds. When considering a *single* structure M, difference (ii) seems to make Hanf locality weaker than Gaifman locality but this is not the case (modulo a small loss in the distance parameter r). Intuitively, a global bijection can be constructed from an isomorphism between a pair of large-radius neighborhoods and the trivial global

isomorphism between two identical structures in such a way that the isomorphism types up to some smaller distance are preserved. One can formalize this argument to show that if a formula is Hanf r-local w.r.t. (M, M) then it is Gaifman $(3r+1)$-local w.r.t. M [HLN99].

4.4 Arb-Invariant First-Order Logic

In this section we introduce our notion of Arb-invariance and give a precise statement of the strong connection between Arb-invariant FO and the queries computable in AC^0.

Arb-invariance. We fix an infinite schema σ_{arb}, containing a binary symbol $<$ together with a symbol for each numerical predicate (the "arb" in σ_{arb} comes from allowing arbitrary numerical predicates). For instance, σ_{arb} contains a symbol $+$ for addition, $*$ for multiplication, and so on. Each numerical predicate is implicitly associated, for every $n \in \mathbb{N}$, with a specific interpretation as a relation of the appropriate arity over the domain $[n] \doteq \{1, 2, \ldots, n\}$. For instance, $+$ is associated with the classical relation of addition over $\mathbb{N}$ restricted to $[n]$. Conversely, for each such family of relations, σ_{arb} contains an associated predicate symbol.

Let M be a τ-structure and $n = |dom(M)|$. An *Arb-expansion* of M is a structure M' over the schema consisting of the disjoint union of τ and σ_{arb} such that $dom(M) = dom(M')$, M and M' agree on all relations in τ, and $<$ is interpreted as a linear order over $dom(M)$. This interpretation induces a bijection between $dom(M)$ and $[n]$, identifying each element of M' with its index relative to $<$. All the numerical predicates are then interpreted over $dom(M')$ via this bijection and their associated interpretation over $[n]$. For instance, $+$ is the ternary relation containing all tuples (a, b, c) of $dom(M')^3$ such that $i + j = k$, where a, b, and c are respectively the i^{th}, j^{th} and k^{th} elements of $dom(M')$ relative to $<$. Note that M' is completely determined by M and the choice of the linear order $<$ on $dom(M)$.

We denote by $\mathrm{FO}(\tau, \mathrm{Arb})$ the set of first-order formulas using the schema $\tau \cup \sigma_{\mathrm{arb}}$. A k-ary formula $\phi(\bar{x})$ of $\mathrm{FO}(\tau, \mathrm{Arb})$ is said to be *Arb-invariant with respect to a finite*

τ-structure M, if for any k-tuple $\bar{a}$ of elements of M, and any two Arb-expansions M' and M'' of M we have

$$M' \models \phi(\bar{a}) \iff M'' \models \phi(\bar{a}). \tag{4.3}$$

When $\phi(\bar{x})$ is Arb-invariant with respect to all finite structures M over a schema, we simply say that $\phi(\bar{x})$ is *Arb-invariant.* Note that this is a semantic property which is not decidable (cf., e.g., [Lib04]).

A k-ary Arb-invariant formula defines a k-ary query over τ-structures as follows. When $\phi(\bar{x})$ is an Arb-invariant formula of $\mathrm{FO}(\tau, \mathrm{Arb})$ on M, we write $M \models \phi(\bar{a})$ whenever there is an Arb-expansion M' of M such that $M' \models \phi(\bar{a})$. Hence we view Arb-invariant formulas as formulas over τ-structures, and so we consider the Gaifman graph of M' to contain edges derived only from the relations in τ (i.e., $G(M') = G(M)$). We denote by *Arb-invariant* $\mathrm{FO}(\tau)$ the set of Arb-invariant formulas of $\mathrm{FO}(\tau, \mathrm{Arb})$, or simply *Arb-invariant* FO if τ is clear from the context. When the formula uses only the predicate $<$ of σ_{arb}, we have the classical notion of order-invariant FO (cf., e.g., [GS00, Lib04]).

Arb-invariance and AC^0. There is a strong connection between AC^0, $\mathrm{FO}(\tau, \mathrm{Arb})$, and Arb-invariant FO. For ordered structures, AC^0 and $\mathrm{FO}(\tau, \mathrm{Arb})$ are equivalent, i.e., they can describe exactly the same sets of bit-strings [Imm87]. This means that for every circuit family $F = (C_m)_{m \in \mathbb{N}}$ of constant depth and polynomial size there is a $\mathrm{FO}(\tau, \mathrm{Arb})$-sentence ϕ_F (over a schema τ that uses a unary relation specifying the positions of the string that carry the letter 1) that is satisfied by exactly those bit-strings that are accepted by F. And vice versa, for every $\mathrm{FO}(\tau, \mathrm{Arb})$-sentence ϕ there exists a corresponding AC^0 circuit family F_ϕ.

For unordered τ-structures, a query is computable in AC^0 iff it is definable in Arb-invariant $\mathrm{FO}(\tau, \mathrm{Arb})$. Recall that, by definition, a k-ary query q on τ-structures is computable in AC^0 iff there is a circuit family $(C_m)_{m \in \mathbb{N}}$ of constant depth and polynomial size

such that for all τ-structures M, all $\bar{a} \in dom(M)^k$, and all $\Gamma \in \mathrm{Rep}(M, \bar{a})$: $C_{|\Gamma|}(\Gamma) = 1$ iff $\bar{a} \in q(M)$. We only need one direction of this equivalence, namely the one that is implied by the following lemma.

Lemma 4.1 (Implicit in [Imm87]). *For each k-ary $\mathrm{FO}(\tau, Arb)$ formula $\phi(\bar{x})$ with alternation depth d, there exists a family of depth-$(d+3)$ polynomial-size circuits $(C_m)_{m\in\mathbb{N}}$ such that for each τ-structure M, for each linear order $<$ on M and the Arb-expansion M' of M induced by $<$, for each tuple $\bar{a} \in dom(M)^k$, and for the string $\Gamma = enc_<(M, \bar{a})$,*

$$C_{|\Gamma|}(\Gamma) = 1 \quad \Longleftrightarrow \quad M' \models \phi(\bar{a}).$$

Note that for a circuit family $F = (C_m)_{m\in\mathbb{N}}$ to compute the query defined by the k-ary formula ϕ over τ-structures, it has to be the case that for all τ-structures M and all $\bar{a} \in dom(M)^k$, $C_{|\Gamma|}(\Gamma)$ is the same for every $\Gamma \in \mathrm{Rep}(M, \bar{a})$. The latter condition exactly corresponds to the formula ϕ in Lemma 4.1 being Arb-invariant.

4.5 Gaifman Locality

We now prove the main result of the chapter – the upper bound in Theorem 3. Recall, our theorem claims that every Arb-invariant FO formula is Gaifman $(\log n)^c$-local, for some constant c which depends only on the formula. In fact, we prove the following slightly stronger version.

Theorem 16. *For each $\mathrm{FO}(\tau, Arb)$ formula $\phi(\bar{x})$ with alternation depth d and any constant $c > d+2$, there exists a constant $n_{\phi,c}$ such that if $\phi(\bar{x})$ is Arb-invariant with respect to a τ-structure M with $n \doteq |M| \geq n_{\phi,c}$, then $\phi(\bar{x})$ is Gaifman $(\log n)^c$-local with respect to M.*

We now briefly sketch the overall proof of Theorem 16. Suppose we have two tuples, $\bar{a}$ and $\bar{b}$, on a τ-structure M, with domain size n, such that their r-neighborhoods, $\mathcal{N}_r^M(\bar{a})$ and $\mathcal{N}_r^M(\bar{b})$, are isomorphic (for some big enough r). Further suppose that there is an $\mathrm{FO}(\tau, \mathrm{Arb})$ formula $\phi(\bar{x})$ which is able to distinguish between $\bar{a}$ and $\bar{b}$ on M while being Arb-invariant with respect to M. Using the link between Arb-invariant $\mathrm{FO}(\tau, \mathrm{Arb})$ formulas and AC^0 circuits from Lemma 4.1, we can view the formula $\phi(\bar{x})$ as a small constant-depth circuit C.

Using the hypothesis that $\phi(\bar{x})$ is Arb-invariant and distinguishes between $\bar{a}$ and $\bar{b}$ on M, we can construct from the circuit C and structure M another circuit $\tilde{C}$ that for a $(2m)$-length binary string w distinguishes between the cases where w contains m occurrences of 1 and $m+1$ occurrences, for some m depending on r. This is the *key step* in our argument. If this happens for large enough m, we get a small circuit computing the promise problem described in Lemma 2.1. We can argue that $\tilde{C}$ has size polynomial in n and depth a constant d' depending only on the alternation depth of $\phi(\bar{x})$. Therefore, if $m > b(\log n)^{d'-1}$ for a large enough constant b, the circuit $\tilde{C}$ we construct violates Lemma 2.1, hence $\phi(\bar{x})$ cannot distinguish between tuples which have isomorphic r-neighborhoods. Our construction is such that m is linearly related to r and therefore $\phi(\bar{x})$ is Gaifman $(\log n)^c$-local for any constant $c > d' - 1$ and sufficiently large n.

4.5.1 Upper Bound for Unary Formulas

In this subsection we consider only unary FO formulas $\phi(x)$. For didactic reasons we first assume that the r-neighborhoods of the elements a and b are disjoint. We argue that we can perform the key step in this setting, and consider the general unary case afterward.

For clarity we describe the intuition with respect to structures that are graphs. Let M be a graph $G = (V, E)$ and take two vertices $a, b \in V$ such that $\pi : \mathcal{N}_r^G(a) \cong \mathcal{N}_r^G(b)$.

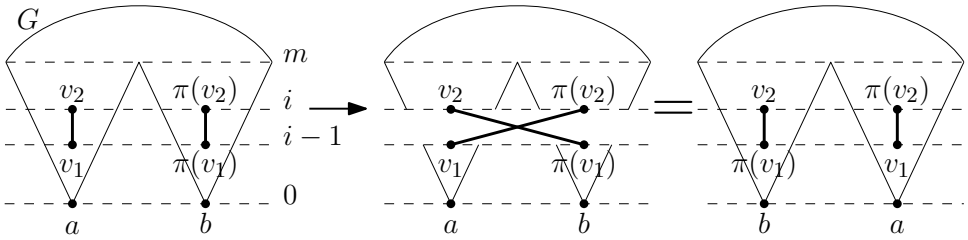


Figure 4.1: Diagram for swapping the neighborhoods of a and b of radius i, conditioned on $w_i = 1$.

Suppose, for the sake of contradiction, that there is a unary FO formula $\phi(x)$ which is Arb-invariant with respect to G and such that $G \models \phi(a) \wedge \neg\phi(b)$. Applying Lemma 4.1 to ϕ gives us a circuit C which, for any vertex $c \in V$, outputs the same value for all strings in $\mathrm{Rep}(G, c)$, and distinguishes $\mathrm{Rep}(G, a)$ from $\mathrm{Rep}(G, b)$.

4.5.1.1 Disjoint Neighborhoods

Let us assume that $\mathcal{N}_r^G(a) \cap \mathcal{N}_r^G(b) = \emptyset$. In this setting it turns out we can pick $m = r$. The neighborhood isomorphism, $\pi : \mathcal{N}_r^G(a) \cong \mathcal{N}_r^G(b)$, implies that the balls of radius $i < r$ around a and b are isomorphic and disjoint in G. Consider the following procedure, depicted in Figure 4.1. For some $i \in [m]$, cut all the edges linking nodes at distance $i - 1$ from a or b to nodes at distance i. Now, swap the positions of the $(i-1)$-neighborhoods around a and b and reconnect the edges in a way that respects the isomorphism π. The resulting graph is isomorphic to G, but the relative positions of a and b have swapped.

Using this intuition we construct a new graph G_w from G, a, and b that depends on a string of m Boolean variables $w \doteq w_1 w_2 \cdots w_m$. We construct G_w so that for each variable w_i, we swap the relative positions of the $(i-1)$-radius balls around a and b iff w_i is 1. The number of such swaps is $|w|_1$. The m-neighborhood isomorphism between a and b implies that $G_w \cong G$. When $|w|_1$ is even, $(G_w, a) \cong (G, a)$, and when $|w|_1$ is odd, $(G_w, a) \cong (G, b)$.

Using the above construction of G_w we derive a circuit $\tilde{C}$ from C that computes parity on

m bits. The circuit $\tilde{C}$ first computes a representation $\Gamma_w \in \mathrm{Rep}(G_w, a)$, and then simulates C on input Γ_w. The above distinguishing property then implies that $\tilde{C}$ accept m-bit strings with even parity. To construct Γ_w we start with a fixed string in $\mathrm{Rep}(G, a)$ and transform it into an element of $\mathrm{Rep}(G_w, a)$ by modifying the edges to switch between the shells in the manner suggested above. Observe that the presence of each edge in G_w depends on at most a single bit of w. This property implies that Γ_w consists of constants, and variables in w or their negations. This means that $\tilde{C}$ is no larger or deeper than C.

We formalize this intuition for general structures and obtain the following lemma.

Lemma 4.2. *Let $m \in \mathbb{N}$. Let M be a τ-structure. Let $a, b \in dom(M)$ such that $dist^M(a,b) > 2m$ and $\mathcal{N}_m^M(a) \cong \mathcal{N}_m^M(b)$. Let C be a circuit that accepts all strings in $\mathrm{Rep}(M,a)$, and rejects all strings in $\mathrm{Rep}(M,b)$. There is a circuit $\tilde{C}$ with the same size and depth as C that computes parity on m bits.*

Proof. For every $r \in \mathbb{N}$ and $a \in dom(M)$, the *r-shell around a in M* is the set

$$S_r^M(a) \doteq \left\{ v \in dom(M) : dist^M(a,v) = r \right\}.$$

Let π be an isomorphism from $\mathcal{N}_m^M(a)$ to $\mathcal{N}_m^M(b)$. Extend π to take $\mathcal{N}_m^M(b)$ back to $\mathcal{N}_m^M(a)$ (that is, extend the domain of the map to the elements of $N_m^M(b)$ and act as π^{-1} for those elements). This is well-defined because $dist^M(a,b) > 2m$ and the m-neighborhoods are disjoint. Note that, in particular, $\pi(a) = b$, and for all $i \in [m]$, $\pi\big(S_i^M(a)\big) = S_i^M(b)$. Let $S_i \doteq S_i^M(a) \cup S_i^M(b)$ for $i \leq m$. Note, that $S_0 = \{a, b\}$.

Let $w \doteq w_1 w_2 \cdots w_m$ be a string of m Boolean variables. We design a structure M_w that has the following property:

$$\text{If } |w|_1 \text{ is even, then } (M_w, a) \cong (M, a).$$
$$\text{If } |w|_1 \text{ is odd, then } (M_w, a) \cong (M, b).$$

When $|w|_1$ is even, $C(M_w, a)$ accepts, because $(M_w, a) \cong (M, a)$. Similarly, when $|w|_1$ is odd $C(M_w, a)$ rejects because $(M_w, a) \cong (M, b)$. Thus, the above property is sufficient to claim that $C(M_w, a)$ accepts iff the parity of w is even. We show how to construct M_w.

Consider a tuple $\bar{v} = (v_1, ..., v_k)$ that belongs to a relation R^M, for some symbol $R \in \tau$ of arity k, that intersects the shells S_{i-1} and S_i for some $i \in [m]$. (Note that other tuples are wholly contained in single shells, because the pairwise distances between the elements of $\bar{v}$ are at most one.) For clarity we reorder the components of $\bar{v}$ so that $\bar{v} \doteq (\bar{v}_1, \bar{v}_2)$ where $\bar{v}_1 \subseteq S_{i-1}$ and $\bar{v}_2 \subseteq S_i$. Each tuple $\bar{v}$ of this type in R^M induces a set of two potential tuples in R^{M_w}: $\bar{v}$ and $(\bar{v}_1, \pi(\bar{v}_2))$. If $w_i = 0$ we copy the tuple $\bar{v}$ from R^M into R^{M_w}; if $w_i = 1$ we add the tuple $(\bar{v}_1, \pi(\bar{v}_2))$ to R^{M_w}.

Observe that, by construction, for any $i \in [m]$, when $w_i = 1$, each tuple intersecting $S^M_{i-1}(a)$ and $S^M_i(a)$ is replaced with a tuple intersecting $S^M_{i-1}(a)$ and $\pi(S^M_i(a)) = S^M_i(b)$, and each tuple intersecting $S^M_{i-1}(b)$ and $S^M_i(b)$ is replaced with a tuple intersecting $S^M_{i-1}(b)$ and $\pi(S^M_i(b)) = S^M_i(a)$. Note this is general because the m-neighborhoods around a and b are disjoint (hence, "cross tuples" are not present in M). Further, for every i where $w_i = 1$, the construction interchanges the roles of the elements in $N^M_{i-1}(a)$ with their images under π in $N^M_{i-1}(b)$. This implies the relative positions of the elements a and b themselves are swapped once for each bit of w that is one. This argument also implies that $M_w \cong M$. Therefore, when the parity of w is odd $(M_w, a) \cong (M, b)$ because a and b swap positions an odd number of times. When the parity of w is even $(M_w, a) \cong (M, a)$ by the same token. This is the property claimed. We conclude the proof by constructing a Boolean circuit $\tilde{C}$, using M_w and C, which computes parity.

Fix an arbitrary string $\Gamma \in \mathrm{Rep}(M, a)$. We derive a new input string Γ_w, with $|\Gamma_w| = |\Gamma|$, from w and Γ. We want Γ_w to be a binary representation of the structure M_w paired with the element a. With this in mind, we copy the encoding of the distinguished element a from Γ into Γ_w. It remains to determine the encoding of M_w in Γ_w.

For each $i \in [m]$, each relation $R \in \tau$, and each tuple $\bar{v} \in R^M$ crossing between shells S_{i-1} and S_i, we encode the corresponding tuple in Γ_w in the following way: Set the bit of Γ_w corresponding to the tuple $\bar{v}$ and relation R to $\neg w_i$, and the bit of Γ_w corresponding to the tuple $(\bar{v}_1, \pi(\bar{v}_2))$ and relation R to w_i. That is, we modify the encoding so that $\bar{v} \in R^{M_w}$ when $w_i = 0$ and $(\bar{v}_1, \pi(\bar{v}_2)) \in R^{M_w}$ when $w_i = 1$. For all other bits in Γ specifying relations, we copy them verbatim from Γ into Γ_w. Observe that the bits of Γ_w are drawn from $\{0, 1, w_i, \neg w_i\}$. This completes the construction of $\Gamma_w \in \mathrm{Rep}(M_w, a)$.

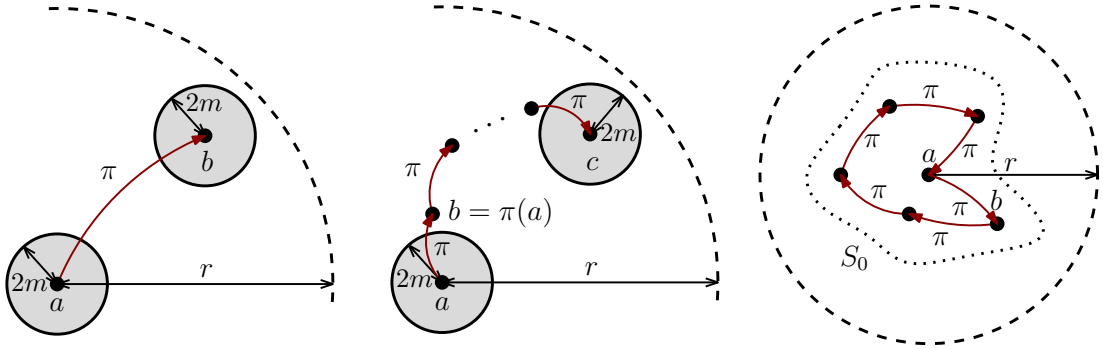
Finally, define the circuit $\tilde{C}(w) \doteq C(\Gamma_w)$. Observe that $\tilde{C}$ is an m-input circuit that has size and depth no more than C because each component of Γ_w is either a constant or a literal of w. ∎

4.5.1.2 General Neighborhoods

We now develop the transformation corresponding to Lemma 4.2 for the general unary case, where the r-neighborhoods around a and b may overlap. As before, we describe the intuition in terms of structures that are graphs.

Consider the iterated application of the isomorphism π to a. We distinguish between two cases. The first case occurs when this iteration travels far from a. That is, for some $t \in \mathbb{N}$, $\pi^t(a)$ is a point c that is far from a. Suppose r is large enough that the isomorphism π implies that a large neighborhood around c is isomorphic to the neighborhood around a. By the triangle inequality, since a is far from c, either (i) b is far from a, or else (ii) c is far from a and b (see Figure 4.2(i),(ii)). We claim that in each case there is a pair of vertices that are distinguished by C, and whose neighborhoods are isomorphic and disjoint. In case (i), a and b are such a pair; in case (ii), C must distinguish either a and c, or b and c, so a and c, or b and c form such a pair. For this pair of vertices, we are in the disjoint case and Lemma 4.2 can be applied to produce a small circuit that computes parity.

The other case occurs when the iterated application of π to a stays close to a (and b).



(i) $\pi(a) = b$ is far from a (Case 1 of Lemma 3.3)

(ii) $\pi^t(a) = c$ is far from a (Case 1 of Lemma 3.3)

(iii) orbit of a stays close to a (Case 2 of Lemma 3.3)

Figure 4.2: Diagram for the general unary case. (r is the radius of the domain of π.)

Let S_0 be the orbit of a under π (i.e., $S_0 \doteq \{\pi^z(a) | z \in \mathbb{N}\}$) (see Figure 4.2(iii)), and let S_i be the vertices at distance i from S_0, for $i \in [2m]$. Because $\pi(S_0) = S_0$ and π is a partial isomorphism on G, the shells S_i are closed under π.

We now play a game similar to the disjoint case. Consider the following procedure, depicted in Figure 4.3. For some $i \in [2m]$ cut all edges between the shells S_{i-1} and S_i. "Rotate" the radius $i-1$ ball around S_0 by π relative to S_i, and reconnect the edges. Because the shells are closed under π, the resulting graph is isomorphic to G. Further, the positions of a and b have shifted relative to an application of π.

As before, we encode this behavior into a modified graph G_w depending on a string of $2m$ Boolean variables $w \doteq w_1 w_2 \cdots w_{2m}$. When $w_i = 0$, we preserve the edges between the shells S_{i-1} and S_i. When $w_i = 1$ we rotate the edges by π. That is, an edge $(v_1, v_2) \in (S_{i-1} \times S_i) \cap E$ becomes the edge $(v_1, \pi(v_2))$ in G_w. The neighborhood isomorphism between a and b implies that $G \cong G_w$. We can argue that

$$(G_w, a) \cong (G, \pi^{|w|_1}(a)). \tag{4.4}$$

We define the circuit $\tilde{C}$ to simulate C on an input $\Gamma_w \in \mathrm{Rep}(G_w, a)$. The above distinguish-

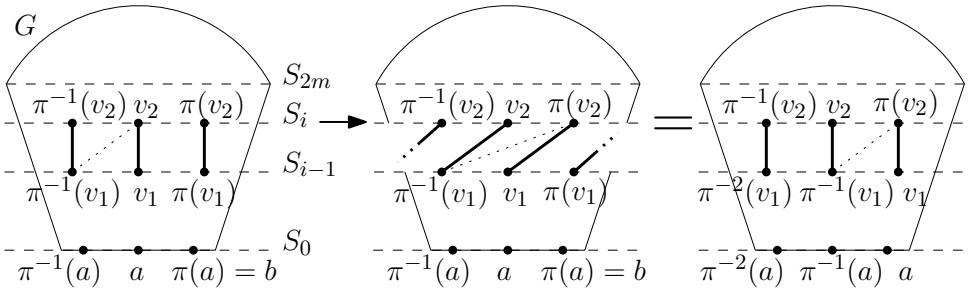


Figure 4.3: Diagram for rotating the shell of radius i around S_0 when $w_i = 1$.

ing property implies that $\tilde{C}$ distinguishes between $|w|_1 \equiv 0 \bmod |S_0|$ and $|w|_1 \equiv 1 \bmod |S_0|$. (Note, this is non-trivial because $|S_0| \geq 2$ since a and b are distinct and in S_0.) This is not quite the promise problem defined in Lemma 2.1. For this reason we modify the construction to shift a by m applications of π^{-1} in Γ_w. This means that $\Gamma_w \in \mathrm{Rep}(G_w, \pi^{-m}(a))$ and $\tilde{C}$ can distinguish between $|w|_1 \equiv m \bmod |S_0|$ and $|w|_1 \equiv m+1 \bmod |S_0|$. This is ruled out by Lemma 2.1, completing the argument.

For general structures, the idea is formalized in the following lemma, where we achieve $r = 10m$.

Lemma 4.3. *Let $m \in \mathbb{N}$. Let M be a τ-structure. Let $a, b \in dom(M)$ such that $\mathcal{N}^M_{10m}(a) \cong \mathcal{N}^M_{10m}(b)$. Let C be a circuit that accepts all strings in $\mathrm{Rep}(M, a)$ and rejects all strings in $\mathrm{Rep}(M, b)$, and for each $c \in dom(M)$, C has the same output for each string in $\mathrm{Rep}(M, c)$. There is a circuit $\tilde{C}$ with the same size and depth as C that distinguishes $|w|_1 = m$ and $|w|_1 = m+1$ for $w \in \{0,1\}^{2m}$.*

Proof. Let π be an isomorphism between $\mathcal{N}^M_{10m}(a)$ and $\mathcal{N}^M_{10m}(b)$. There are two cases:

Case 1. *The iterated isomorphism takes a far from a.*

More specifically, there exists $t \in \mathbb{N}$ such that

$$dist^M(a, \pi^t(a)) > 8m. \tag{4.5}$$

Let t be the minimal value such that (4.5) holds. Let $c \doteq \pi^t(a)$. Hence $dist^M(a,c) > 8m$. Since t is minimal, $dist^M(a,\pi^j(a)) \leq 8m$ for all $j < t$. Because the isomorphism π preserves neighborhoods contained in $N^M_{10m}(a)$, it follows that for all $j < t$, $N^M_{2m}(\pi^j(a)) \subseteq N^M_{8m+2m}(a)$, and π induces an isomorphism from $\mathcal{N}^M_{2m}(\pi^j(a))$ to $\mathcal{N}^M_{2m}(\pi^{j+1}(a))$. This implies that $\mathcal{N}^M_{2m}(a) \cong \mathcal{N}^M_{2m}(b) \cong \mathcal{N}^M_{2m}(c)$.

As $dist^M(a,c) > 8m$, the triangle inequality implies that either $dist^M(a,b) > 4m$ or $dist^M(b,c) > 4m$. In the former case, we complete by applying Lemma 4.2 and observing that parity on $2m$ bits distinguishes between inputs with m ones and inputs with $m+1$ ones. In the latter case, depending on whether C accepts $\mathrm{Rep}(M,c)$ or not, we can proceed either with the pair b and c or the pair a and c. In either case, from the above we see that this pair of points have isomorphic $2m$-neighborhoods and are more than $4m$ apart. Therefore Lemma 4.2 again suffices to reach the required conclusion.

Case 2. *The iterated isomorphism keeps a close to a.*

More specifically, for all $t \in \mathbb{N}$, $dist^M(a,\pi^t(a)) \leq 8m$.

Let $S_0 \subseteq dom(M)$ be the orbit of a under π. Note that $\pi(S_0) = S_0$ and $b \in S_0$. We define S_i as the set of elements of M at distance i from S_0. Because π is an isomorphism from $\mathcal{N}^M_{10m}(a)$ to $\mathcal{N}^M_{10m}(b)$, each S_i is also closed under π for $i \leq 2m$.

Let $w \doteq w_1 w_2 \cdots w_{2m}$ be a string of $2m$ Boolean variables. We proceed similarly to the proof of Lemma 4.2 when constructing a structure M_w and representation Γ_w. We construct M_w and a distinguished element a' so that the following property holds:

$$\text{If } |w|_1 \equiv m \bmod |S_0|, \text{ then } (M_w, a') \cong (M, a).$$
$$\text{If } |w|_1 \equiv m+1 \bmod |S_0|, \text{ then } (M_w, a') \cong (M, b).$$

When $|w|_1 = m$, $C(M_w, a')$ accepts, because $(M_w, a') \cong (M, a)$. Similarly, when $|w|_1 = m+1$, $C(M_w, a')$ rejects, because $(M_w, a') \cong (M, b)$. This property is sufficient to

claim that $C(M_w, a')$ distinguishes between $|w|_1 = m$ and $|w|_1 = m+1$, because $|S_0| > 1$ since a and b are distinct and in S_0. We now show how to construct M_w and a'.

Let the structure M_i denote the result of performing the construction from Lemma 4.2 only for the tuples intersecting shells S_j and S_{j+1}, for all $j < i$. Note $M_0 = M$ and $(M_0, v) = (M, v)$ for all $v \in S_0$. When $w_i = 0$, we have $M_{i-1} = M_i$, hence $(M_{i-1}, v) = (M_i, v)$ for any $v \in S_0$, because the construction makes no modifications to the structure between shell S_{i-1} and S_i in this case. When $w_i = 1$, we are rotating the neighborhood below shell S_i by π^{-1} relative to S_i. Since the shells are closed under the action of π, we can conclude that $M_i \cong M_{i-1}$ for $i \in [2m]$. This also implies that when $w_i = 1$, $(M_i, v) \cong (M_{i-1}, \pi(v))$. It follows that for any $v \in S_0$ and $i \in [2m]$: $(M_i, v) \cong (M_{i-1}, \pi^{w_i}(v))$. By applying this fact $2m$ times we reach the conclusion that for all $v \in S_0$,

$$(M_w, v) \doteq (M_{2m}, v) \cong (M_0, \pi^{|w|_1}(v)) = (M, \pi^{|w|_1}(v)).$$

The length of the orbit of a with respect to π is $|S_0|$. Define $a' \doteq \pi^{-m}(a)$. Since $a' \in S_0$, it follows that when $|w|_1 \equiv 0 \bmod |S_0|$, $(M_w, a') \cong (M, a')$ and when $|w|_1 \equiv 1 \bmod |S_0|$, $(M_w, a') \cong (M, \pi(a'))$. Observe this implies that when $|w|_1 \equiv m \bmod |S_0|$, $(M_w, a') \cong (M, \pi^m(\pi^{-m}(a))) = (M, a)$ and when $|w|_1 \equiv m+1 \bmod |S_0|$, $(M_w, a') \cong (M, \pi^{m+1}(\pi^{-m}(a))) = (M, b)$. This is the property claimed. It remains to construct the circuit $\tilde{C}$.

We construct the string Γ_w in the same way as in the proof of Lemma 4.2 with respect to M, w, π, and the shells $\{S_i\}_{i \leq 2m}$ defined above. Note that in the case where the construction would assign both w_i and $\neg w_i$ to a bit of Γ_w corresponding to some tuple in a relation of M_w we instead set the corresponding bit of Γ_w to 1. The string Γ_w represents the pairing of the structure M_w with the element a. Form Γ'_w from Γ_w by replacing the encoding of distinguished element a with an encoding of a'. Note that $\Gamma'_w \in \mathrm{Rep}(M_w, a')$. Setting

$\tilde{C}(w) \doteq C(\Gamma'_w)$ completes the proof. ∎

Notice that the idea behind the proof of this lemma is quite similar to the disjoint case. When the neighborhoods are disjoint, the above construction gives $S_0 = \{a, b\}$. In this case the "rotation" by π becomes a swap. Further, since $|S_0| = 2$, the promise problem we solve is distinguishing between $|w|_1 \equiv m \bmod 2$, and $|w|_1 \equiv m + 1 \bmod 2$ – this is exactly parity! Thus, in the case of disjoint neighborhoods, the construction in the proof of Lemma 4.3 reduces to the one from Lemma 4.2.

With Lemma 4.3 in hand, we are ready to finish the proof of Theorem 16 in the unary case.

Proof (of Theorem 16 for the case $k = 1$). Assume that $\phi(x)$ is a unary FO(τ, Arb) formula with alternation depth d that is Arb-invariant with respect to a τ-structure M with $n \doteq |M|$. Since $\phi(x)$ is FO(τ, Arb), it is computable by a family of circuits in AC^0 (cf. Lemma 4.1). That is, there are a constant e and a circuit C with depth $d + 3$ and size n^e such that, C computes $\phi(x)$ on size n τ-structures. Since $\phi(x)$ is Arb-invariant with respect to M, for each fixed $a \in dom(M)$, C has the same output for all strings in $\mathrm{Rep}(M, a)$.

Now, for the sake of contradiction, suppose $\phi(x)$ is *not* Gaifman $(\log n)^c$-local with respect to M, for some constant $c > d + 2$.This implies that $\phi(x)$ distinguishes between two elements $a, b \in dom(M)$ having isomorphic $(\log n)^c$-neighborhoods.

Let $m \doteq \lfloor \frac{(\log n)^c}{10} \rfloor$. Therefore $\mathcal{N}^M_{10m}(a) \cong \mathcal{N}^M_{10m}(b)$. The circuit C then satisfies the assumptions of Lemma 4.3. From the lemma, we obtain a circuit $\tilde{C}$ of depth $d + 3$ and size n^e that distinguishes between $|w|_1 = m$ and $|w|_1 = m + 1$ for $w \in \{0, 1\}^{2m}$.

From Lemma 2.1 we obtain that $n^e > 2^{\alpha m^{1/(d+3-1)}}$, which is equivalent to $e \log n > \alpha m^{1/(d+2)}$. The latter condition is violated if we set $m = (\log n)^c$ whenever c is a constant larger than $d + 2$ and n is sufficiently large (depending on ϕ and c). This yields the required contradiction, completing the proof. ∎

4.5.2 Reducing the Arity

To argue Theorem 16 in the case of formulas with an arbitrary number of free variables, we prove the following reduction. Given a k-ary FO(Arb) formula ϕ that is Arb-invariant with respect to the structure M and distinguishes two k-tuples $\bar{a}$ and $\bar{b}$ that have isomorphic r-neighborhoods, we produce, for some $k' < k$, a k'-ary FO(Arb) formula ϕ' that is Arb-invariant with respect to an extended structure M' and distinguishes between two k'-tuples $\bar{a}'$ and $\bar{b}'$ that have isomorphic r'-neighborhoods. Furthermore, r' is only slightly smaller than r.

Repeated application of this idea transforms a distinguishing k-ary formula into a distinguishing unary formula with slightly weaker parameters. For large enough initial radius r this is sufficient to contradict the Gaifman locality of unary formulas.

We first give an intuitive description of the reduction argument. Let $\phi(\bar{x})$ be a k-ary formula as above, and let $\pi : \mathcal{N}_r^M(\bar{a}) \cong \mathcal{N}_r^M(\bar{b})$ denote a neighborhood isomorphism. The main idea is to transfer some of the information present in the initial k-tuples $\bar{a}$ and $\bar{b}$ into a new marking relation R such that we can recover $\bar{a}$ and $\bar{b}$ from $k' < k$ of their components $\bar{a}'$ and $\bar{b}'$ as extensions of $\bar{a}'$ and $\bar{b}'$ that satisfy R. In that case, the formula

$$\phi'(\bar{y}) = (\exists \bar{z})\, R \wedge \phi(\bar{y}, \bar{z}) \tag{4.6}$$

has arity $k' < k$, and distinguishes the tuples $\bar{a}'$ and $\bar{b}'$ over the extension M' of M with R, where R is a relation of arity at most k evaluated over some of the variables in $\bar{y}$ and $\bar{z}$. The formula ϕ' is Arb-invariant over M' since ϕ is Arb-invariant over M, the domain of M' is the same as of M, and R does not use the Arb relations. Moreover, provided the marking relation R is invariant under π, π also induces a neighborhood isomorphism $\mathcal{N}_{r'}^{M'}(\bar{a}') \cong \mathcal{N}_{r'}^{M'}(\bar{b}')$ in the new structure, albeit possibly for a smaller radius r', e.g., due to the effect of the introduction of R on the Gaifman graph.

We start by considering three situations in which it is relatively simple to obtain a π-invariant marking relation R, and then see how to handle the remaining case. Throughout, we assume without loss of generality that $\bar{a}$ is accepted by ϕ and $\bar{b}$ is rejected by ϕ, and we use the notation $\bar{a} \doteq (a_1, a_2, \ldots, a_k)$ and $\bar{b} \doteq (b_1, b_2, \ldots, b_k)$.

Case 1. *The tuples $\bar{a}$ and $\bar{b}$ have a component in common.*

Say $a_k = b_k$. Then the fact that ϕ distinguishes $\bar{a}$ and $\bar{b}$ is independent of the last component of the tuples. To exploit this redundancy we mark the element a_k and derive a $(k-1)$-ary formula $\phi'(\bar{y})$ as in (4.6), where R checks whether $\bar{z}$ equals a_k. Since $a_k = b_k$ and π has to map a_k to b_k, a_k is a fixed point of π, which guarantees that the marking relation is invariant under π. In this case the original isomorphism π remains a neighborhood isomorphism with the same radius $r' = r$.

Case 2. *The elements in the orbit of $\bar{a}$ stay well within the isomorphism neighborhood of $\bar{a}$.*

The *orbit* of $\bar{a}$ is the set of tuples $\pi^t(\bar{a})$ for all $t \in \mathbb{N}$. We now use the relation R to mark all tuples in the orbit of $\bar{a}$. The relation R is π-invariant because the entire orbit stays within the domain of the neighborhood isomorphism π.

This marking allows us to reduce the arity as follows. First, note that we can apply Case 1 whenever there is a pair of marked tuples that is distinguished by ϕ and has a component in common. Therefore, we can assume that ϕ does not distinguish any marked tuples that share a component.

This observation allows us to recover the tuples $\bar{a}$ and $\bar{b}$ (or equivalent ones) from their first components only. Let $\bar{a}' \doteq (a_1)$ and $\bar{b}' \doteq (b_1)$. Since ϕ accepts $\bar{a}$ and $\bar{a}$ is marked, existentially guessing the remaining components consistent with $\bar{a}$ and R shows that ϕ' defined by (4.6) accepts $\bar{a}'$. On the other hand, since ϕ rejects $\bar{b}$, ϕ rejects all marked tuples that share the first component with $\bar{b}$. Therefore, no marked tuples with b_1 as

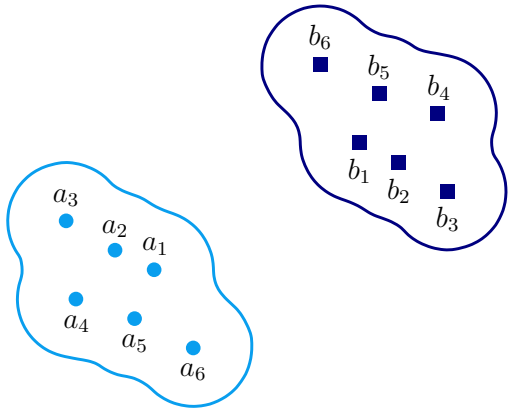


Figure 4.4: Diagram for Case 3.

first component are accepted by ϕ, and hence ϕ' rejects $\bar{b}'$. This establishes the required distinguishing property of ϕ'.

We already mentioned that the marking relation R is π-invariant. The fact that the entire orbit of $\bar{a}$ stays *well within* the isomorphism neighborhood guarantees that the new isomorphism radius r' is not too much smaller than the original radius r.

Case 3. *All components of $\bar{a}$ are close to each other.*

Because of Case 2, we only need to consider the situation where the iterates of π take some component of $\bar{a}$ far from $\bar{a}$. Without loss of generality we can assume that $\bar{b}$ has a component that is far from $\bar{a}$. Also, since π preserves distances, we know that all components of $\bar{b}$ are close to each other. This allows us to choose a relatively large $r' \leq r$ such that the r'-neighborhoods of $\bar{a}$ and $\bar{b}$ do not intersect. So, the situation is as sketched in Figure 4.4.

In this case, simply marking the tuples $\bar{a}$ and $\bar{b}$ yields a relation R that is invariant under π on $N_{r'}^M(\bar{a})$ for the relatively large radius r'. The π-invariance follows from the fact that $\bar{a}$ and $\bar{b} = \pi(\bar{a})$ are far apart, as the range of π on $N_{r'}^M(\bar{a})$ falls entirely outside of $N_{r'}^M(\bar{a})$, so no tuple other than $\bar{a}$ needs to be marked in $N_{r'}^M(\bar{a})$ in order for the marking to be π-invariant.

With this marking, knowledge of one component of $\bar{a}$ and $\bar{b}$ suffices to recover the full

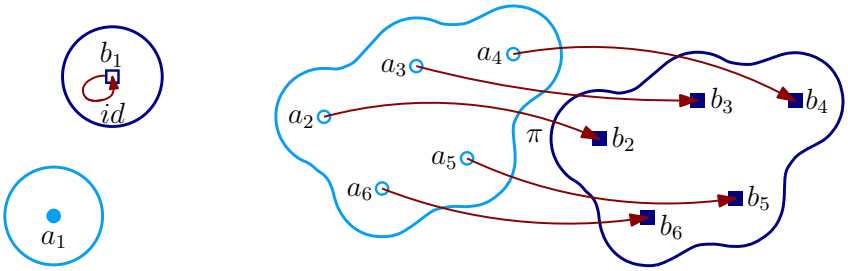


Figure 4.5: Diagram for the hybrid isomorphism from Case 4 that maps $\bar{h} = (b_1, a_2, \ldots, a_k)$ to $\bar{b} = (b_1, b_2, \ldots, b_k)$.

tuples, so we can reduce the arity to $k' = 1$ following (4.6).

Case 4. *Hybrid case.*

In the remaining case we can assume without loss of generality that some component of $\bar{b}$, say b_1, is far from $\bar{a}$, and that a_1 is far from some other component of $\bar{a}$. Due to the isomorphism π, the latter is equivalent to b_1 being far from some other component of $\bar{b}$.

In this case, we do not know how to apply the π-invariant marking strategy to the given tuples $\bar{a}$ and $\bar{b}$. However, we can construct a "hybrid" tuple $\bar{h}$ that has some components in common with $\bar{a}$ and some with $\bar{b}$ such that Case 1 applies to either $\bar{a}$ and $\bar{h}$, or to $\bar{h}$ and $\bar{b}$.

For simplicity, let us first consider the situation where b_1 is far from *all* other components of $\bar{b}$. Recall that b_1 is also far from $\bar{a}$. These two facts imply that for a large radius the neighborhood around $\bar{b}$ is isomorphic to the neighborhood around the tuple $\bar{h} \doteq (b_1, a_2, \ldots, a_k)$. To see this, consider the map which acts as the identity map on the elements near b_1 and acts as π on the elements near $a_2, \ldots, a_k$. Figure 4.5 illustrates the construction. The distance between b_1, and both $\bar{a}$ and the rest of $\bar{b}$ ensures that no tuples that are in a relation of M straddle the neighborhood of b_1 as well as the neighborhood of some other component. Therefore, on such a tuple our map either acts like the identity on all components, or like π on all components. Since both the identity and π preserve the relations of M, so does our map.

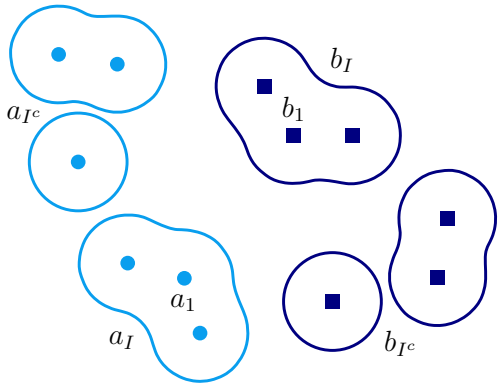


Figure 4.6: Diagram for Case 4 when b_1 is close to some component of $\bar{b}$.

By transitivity, we also have a neighborhood isomorphism between $\bar{a}$ and $\bar{h}$. We also know that ϕ distinguishes $\bar{h}$ from one of $\bar{a}$ and $\bar{b}$ because those tuples are themselves distinguished by ϕ. Thus, there is some pair of tuples which have large isomorphic neighborhoods, are distinguished by ϕ, and share components (since $\bar{h}$ is a hybrid of $\bar{a}$ and $\bar{b}$). We conclude by applying Case 1 to reduce the arity of the formula, while only slightly decreasing the radius of the neighborhood isomorphism.

Finally, consider the case where b_1 is close to some components of $\bar{b}$ and far from others. We iteratively group the components of $\bar{b}$ closest to b_1 to form the set $\bar{b}_I$, until all remaining components are far from $\bar{b}_I$. So, by construction the elements of $\bar{b}_I$ are far from the other components of $\bar{b}$ and far from $\bar{a}$, since b_1 is far from $\bar{a}$. Viewing the component $\bar{b}_I$ as a single element allows us to apply the argument from the previous paragraph to reduce the instance, albeit with some further loss in the isomorphism radius. This loss is caused by the distortion in distance from grouping elements in this way. See Figure 4.6 for a diagram of this case.

These ideas are formalized in the proof of the following lemma.

Lemma 4.4. *Let $k, d, r \in \mathbb{N}$ and τ be a schema. Let M be a τ-structure with tuples $\bar{a}, \bar{b} \in dom(M)^k$. Let $\phi(\bar{x})$ be a k-ary $\mathrm{FO}(\tau, Arb)$ formula with alternation depth $d > 0$ which is Arb-invariant with respect to M. Suppose:*

1. *$M \models \phi(\bar{a}) \wedge \neg\phi(\bar{b})$, and*

2. *$\pi : \mathcal{N}_r^M(\bar{a}) \cong \mathcal{N}_r^M(\bar{b})$.*

There is a $k' < k$, a schema $\tau' \supseteq \tau$, a τ'-structure M' with tuples $\bar{a}', \bar{b}' \in dom(M')^{k'}$ and a k'-ary $\mathrm{FO}(\tau', Arb)$ formula $\phi'(\bar{y})$ with alternation depth d which is Arb-invariant with respect to M' such that:

1′. *$M' \models \phi(\bar{a}') \wedge \neg\phi(\bar{b}')$, and*

2′. *$\pi' : \mathcal{N}_{r'}^{M'}(\bar{a}') \cong \mathcal{N}_{r'}^{M'}(\bar{b}')$,*

where

$$r' = \frac{r}{9k}. \tag{4.7}$$

Proof. Since $d > 0$, we can assume without loss of generality that the first quantifier of ϕ is existential, otherwise, we can work with the formula $\neg\phi$ instead and swap the labels of $\bar{a}$ and $\bar{b}$.

Let $\bar{a} \doteq (a_1, a_2, \cdots, a_k)$ and $\bar{b} \doteq (b_1, b_2, \cdots, b_k)$. There are three main cases. In each of the cases, the resulting formula $\phi'(\bar{y})$ is of the form

$$\phi'(\bar{y}) = (\exists \bar{z} \in dom(M')^{k-k'})\ R \wedge \phi(\bar{y}, \bar{z}),$$

where R is a new relation on some subset of the variables $\bar{y}$ and $\bar{z}$ added to the structure M to form M' that does not depend on the order or the arbitrary numerical predicates. It follows that since ϕ is Arb-invariant with respect to M, ϕ' is Arb-invariant with respect

to M'. The form of ϕ' also implies that ϕ' has alternation depth d since ϕ begins with an existential quantifier. We use two distance parameters ℓ and s, in addition to r', which we establish conditions on in the course of the proof. We optimize their value at the end. It remains to show that properties $1'$ and $2'$ hold for radius r' in all cases.

Case 1. *There exists $i \in [k]$ such that $a_i = b_i$.*

Assume without loss of generality that $i = k$. Expand the structure M to M' by adding a new unary predicate $R \notin \tau$ where R is satisfied only by the element a_k. Construct a new formula:

$$\phi'(y_1, y_2, \ldots, y_{k-1}) \doteq (\exists z_k) \;\; R(z_k) \wedge \phi(y_1, y_2, \ldots, y_{k-1}, z_k).$$

Let $\bar{a}' \doteq (a_1, a_2, \ldots, a_{k-1})$ and $\bar{b}' \doteq (b_1, b_2, \ldots, b_{k-1})$. Property $1'$ holds because of Property 1. To see that Property $2'$ holds, observe that the isomorphism π is a bijection between $\cup_{j<k} N_r^M(a_j)$ and $\cup_{j<k} N_r^M(b_j)$. For all relations in τ, π is an isomorphism between these two sets. Further π preserves R on these sets because π maps a_k to itself. From this, it follows that $\pi : \mathcal{N}_{r'}^{M'}(\bar{a}') \cong \mathcal{N}_{r'}^{M'}(\bar{b}')$ and Property $2'$ holds for any radius $r' \leq r$.

In summary, the isomorphism radius of this case satisfies $r' \leq r$, the isomorphism π is not modified, the structure gains one new relation, and the arity of the formula is reduced by one.

Case 2. *For all $t \in \mathbb{N}$ and $i \in [k]$, $dist^M(\bar{a}, \pi^t(a_i)) \leq 2\ell$.*

For all $t \in \mathbb{N}$, $\pi^t(\bar{a})$ is a tuple in $N_{2\ell}^M(\bar{a})$. It follows that $N_{r'}^M(\pi^t(\bar{a})) \subseteq N_{2\ell+r'}^M(\bar{a})$. If

$$r \geq 2\ell + r', \tag{4.8}$$

these r'-neighborhoods of $\pi^t(\bar{a})$ are contained in the domain of π and we have a chain of r'-neighborhood isomorphisms resulting in $\mathcal{N}_{r'}^M(\bar{a}) \cong \mathcal{N}_{r'}^M(\pi^t(\bar{a}))$.

Suppose that there is $t \in \mathbb{N}$ and $i \in [k]$ such that $b_i = \pi^t(b_i)$, and $M \models \phi(\pi^t(\bar{b}))$. In this case, we finish via the argument in Case 1, because ϕ distinguishes the tuples $\bar{b}$ and $\pi^t(\bar{b})$, these tuples share a component, and $\mathcal{N}_{r'}^M(\bar{b}) \cong \mathcal{N}_{r'}^M(\pi^t(\bar{b}))$. Thus, assume otherwise, i.e., for all $t \in \mathbb{N}$ and $i \in [k]$,

$$b_i = \pi^t(b_i) \Rightarrow M \models \neg\phi(\pi^t(\bar{b})). \tag{4.9}$$

We expand the structure M to M' by adding a new k-ary relation $R \notin \tau$ containing the tuples $\cup_{t\in\mathbb{N}}\{\pi^t(\bar{b})\}$. Define a new formula:

$$\phi'(y_1) \doteq (\exists z_2, z_3, \ldots, z_k) \;\; R(y_1, z_2, \ldots, z_k) \wedge \phi(y_1, z_2, \ldots, z_k).$$

Let $\bar{a}' \doteq (a_1)$ and $\bar{b}' \doteq (b_1)$. We now establish Property 1′. First, observe that $M' \models \phi'(\bar{a}')$ via the witness $(a_2, a_3, \ldots, a_k)$. We now argue that $M' \models \neg\phi'(\bar{b}')$. Suppose the contrary, that $M' \models \phi'(\bar{b}')$, then there exists $(c_2, c_3, \ldots, c_k) \in dom(M)^{k-1}$ and $t \in \mathbb{N}$ such that $M \models \phi(b_1, c_2, c_3, \ldots, c_k)$ and $\pi^t(\bar{b}) = (b_1, c_2, c_3, \ldots, c_k)$. This contradicts (4.9). Therefore $M' \models \phi'(\bar{a}') \wedge \neg\phi'(\bar{b}')$, hence Property 1′ holds.

We now establish Property 2′. Observe that $R \subseteq (N_{2\ell}^M(\bar{a}))^k$. This implies that for all $t \in \mathbb{N}$, $N_{r'}^{M'}(\pi^t(\bar{a}')) \subseteq N_{2\ell+r'}^M(\bar{a})$ and further that $N_{r'}^{M'}(\pi^t(\bar{a}'))$ is within the domain of π. Hence when π acts on $N_{r'}^{M'}(\pi^t(\bar{a}'))$ all relations in τ are preserved. The mapping π also preserves R on $N_{r'}^{M'}(\pi^t(\bar{a}'))$ because R is exactly the orbit of $\bar{a}$ under π. From this, it follows that for all $t \in \mathbb{N}$, $\mathcal{N}_{r'}^{M'}(\bar{a}') \cong \mathcal{N}_{r'}^{M'}(\pi^t(\bar{a}'))$. In particular, $\mathcal{N}_{r'}^{M'}(\bar{a}') \cong \mathcal{N}_{r'}^{M'}(\bar{b}')$ and Property 2′ holds with $r' \leq r - 2\ell$ (see (4.9)).

In summary, the isomorphism radius is reduced to $r' \leq r - 2\ell$, the structure gains a new relation, and the arity is reduced to one.

Cases 3 & 4 *There exists* $t \in \mathbb{N}$ *and* $i \in [k]$ *such* $dist^M(\bar{a}, \pi^t(a_i)) > 2\ell$.

Select t minimal and assume without loss of generality that $i = 1$. Let $\bar{c} \doteq \pi^t(\bar{a})$. Thus, $dist^M(\bar{a}, c_1) > 2\ell$. We argue that we can assume with loss of generality that we have a pair of tuples $\bar{a}^*$ and $\bar{b}^*$ such that for a large distance s (to be determined later):

(i) $\phi \models \phi(\bar{a}^*) \wedge \neg\phi(\bar{b}^*)$,

(ii) $\mathcal{N}_s^M(\bar{a}^*) \cong \mathcal{N}_s^M(\bar{b}^*)$, and

(iii) $dist^M(\bar{a}^*, b_1^*) > \ell$.

Suppose $dist^M(\bar{b}, c_1) \leq \ell$. Since $dist^M(\bar{a}, c_1) > 2\ell$ it follows that $dist^M(\bar{a}, b_j) > \ell$, for some $j \in [k]$. Therefore $\bar{a}$ and $\bar{b}$ satisfy condition (iii) with coordinate j permuted to 1. Conditions (i), and (ii) with

$$s \leq r \tag{4.10}$$

follow by properties 1 and 2 in the hypothesis of the lemma.

Otherwise, $dist^M(\bar{b}, c_1) > \ell$. Because t is selected minimally, for all $j < t$, $N_s^M(\pi^j(\bar{a})) \subseteq N_{2\ell+s}^M(\bar{a})$. If

$$r \geq 2\ell + s, \tag{4.11}$$

these s-neighborhoods of $\pi^j(\bar{a})$ are contained in the domain of π and we have a chain of s-neighborhood isomorphisms resulting in $\mathcal{N}_s^M(\bar{a}) \cong \mathcal{N}_s^M(\bar{c})$. Since ϕ distinguishes $\bar{a}$ and $\bar{b}$, ϕ must be able to distinguish between either $\bar{a}$ and $\bar{c}$, or $\bar{b}$ and $\bar{c}$. Therefore, for some pair, all three conditions (i), (ii), and (iii) are met.

Thus, we have a pair of tuples $\bar{a}^*$ and $\bar{b}^*$ that satisfy conditions (i), (ii), and (iii) with $s \leq r - 2\ell$. Let π relabel the isomorphism between the s-neighborhoods for this new pair. There are two subcases.

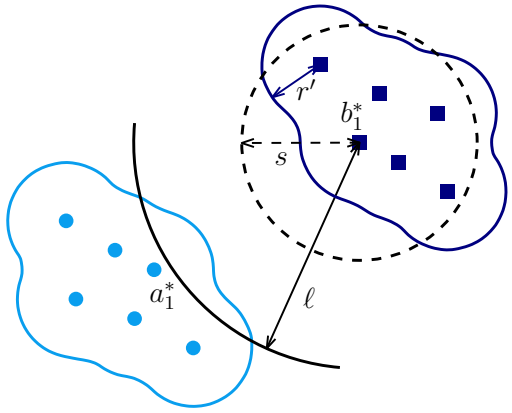


Figure 4.7: Diagram for Case 3.

Case 3. *For all* $j \in [k]$, $dist^M(b_1^*, b_j^*) \leq s$.

See Figure 4.7 for a diagram of this case. Because of property (ii) and since isomorphisms preserve distance, for all $j \in [k]$, $dist^M(a_1^*, a_j^*) \leq s$.

Expand the structure M to form M' by introducing a new k-ary relation $R \notin \tau$ containing only the tuples $\bar{a}^*$ and $\bar{b}^*$. Let $\bar{a}' \doteq (a_1^*)$ and $\bar{b}' \doteq (b_1^*)$. Construct a new formula:

$$\phi'(y_1) \doteq (\exists z_2, z_3, \ldots, z_k) \;\; R(y_1, z_2, \ldots, z_k) \wedge \phi(y_1, z_2, \ldots, z_k).$$

By Property (iii), a_1^* and b_1^* are distinct. This means that $\bar{a}'$ and $\bar{b}'$ correspond with the distinct elements of R, and, with Property (i), we determine that $M' \models \phi'(\bar{a}') \wedge \neg\phi'(\bar{b}')$, hence Property $1'$ holds.

To establish Property $2'$, consider radius r', with

$$r' \leq s. \tag{4.12}$$

The tuple $\bar{a}^* \in R$ is fully contained in $N_{r'}^{M'}(\bar{a}')$, because, due to the addition of R when going from M to M', the distance between any two distinct points in $\bar{a}^*$ is reduced to

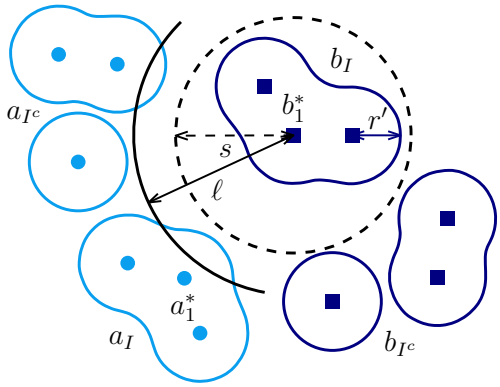


Figure 4.8: Diagram for Case 4.

1. Thus, if the sets $N_{r'}^{M'}(\bar{a}')$ and $N_{r'}^{M'}(\bar{b}')$ are disjoint, there is no intersection between $\bar{a}^*$ and $N_{r'}^{M'}(\bar{b}')$ (similarly for the tuple $\bar{b}^*$ and $N_{r'}^{M'}(\bar{a}')$). The fact that $R = \{\bar{a}^*, \bar{b}^*\}$, Property (ii) holds, and (4.12) imply that π preserves R on the domain $\mathcal{N}_{r'}^{M'}(\bar{a}')$. Hence, $\pi : \mathcal{N}_{r'}^{M'}(\bar{a}') \cong \mathcal{N}_{r'}^{M'}(\bar{b}')$. Thus Property 2′ holds for r'. It remains to establish a sufficient condition for such disjointness.

First, observe that $dist^M(\bar{a}^*, \bar{b}^*) > \ell - s$, because all elements of $\bar{b}^*$ are within s of b_1^* and $dist^M(\bar{a}^*, b_1^*) > \ell$. This implies that $dist^{M'}(\bar{a}^*, \bar{b}^*) > \ell - s$, because the tuples in R cannot contribute an edge in a shortest path between $\bar{a}^*$ and $\bar{b}^*$. Further, since $\bar{a}'$ and $\bar{b}'$ are elements in $\bar{a}^*$ and $\bar{b}^*$, $dist^{M'}(\bar{a}', \bar{b}') > \ell - s$. Therefore, if we select

$$r' \leq \frac{\ell - s}{2}, \tag{4.13}$$

the r'-neighborhoods of $\bar{a}'$ and $\bar{b}'$ are disjoint in M'.

In summary, the structure gets one new relation, the isomorphism radius reduces to $r' \leq \min(\frac{\ell - s}{2}, s)$, and the arity reduces to one.

Case 4. *There exists $j \in [k]$, such that $dist^M(b_1^*, b_j^*) > s$.*

See Figure 4.8 for a diagram of this case. In this case we can construct a hybrid tuple $\bar{h}$

from $\bar{a}^*$ and $\bar{b}^*$ such that ϕ distinguishes $\bar{h}$ from one of $\bar{a}^*$ or $\bar{b}^*$ and the r'-neighborhoods of all three tuples are isomorphic, where the term "hybrid" means that $\bar{h}$ has components from both $\bar{a}^*$ and $\bar{b}^*$. As this pair of tuples shares some common components we can apply Case 1 to conclude.

For an index set $I \subseteq [k]$, let the tuple $\bar{b}_I$ consist of only the components of $\bar{b}^*$ with indices in I. Start with $I \doteq \{1\}$. While there is an $i \in [k] \backslash I$ such that $dist^M(b_i^*, \bar{b}_I) \leq 2r' + 1$, add i to I. If

$$(k-1)(2r'+1) \leq s, \tag{4.14}$$

$\bar{b}_I$ cannot contain every component of $\bar{b}^*$ (by the hypothesis of this case). Let I^c be the complement of I (i.e., $I^c \doteq [k] \backslash I$) and define $\bar{h} \doteq (\bar{b}_I, \bar{a}_{I^c})$. We argue that we can construct an isomorphism

$$\rho : \mathcal{N}_{r'}^M(\bar{b}^*) = \mathcal{N}_{r'}^M(\bar{b}_I, \bar{b}_{I^c}) \cong \mathcal{N}_{r'}^M(\bar{b}_I, \bar{a}_{I^c}) = \mathcal{N}_{r'}^M(\bar{h}).$$

This implies that

$$\rho \circ \pi : \mathcal{N}_{r'}^M(\bar{a}^*) \cong \mathcal{N}_{r'}^M(\bar{h}).$$

Together this implies that the r'-neighborhoods of $\bar{a}^*$, $\bar{b}^*$ and $\bar{h}$ are isomorphic. Since one of $M \models \phi(\bar{h})$ or $M \models \neg\phi(\bar{h})$ holds we can conclude by applying Case 1 with $\bar{h}$ and one of $\bar{a}^*$ and $\bar{b}^*$. This establishes properties $1'$ and $2'$ for radius r'.

It remains to argue the isomorphism ρ exists. If $N_{r'}^M(\bar{b}_I)$ is at least distance two from both $N_{r'}^M(\bar{b}_{I^c})$ and $N_{r'}^M(\bar{a}_{I^c})$, it suffices to define ρ to be identity map on the domain $N_{r'}^M(\bar{b}_I)$ and act as π^{-1} on the domain $N_{r'}^M(\bar{b}_{I^c})$, since no tuple in a relation can straddle both parts of the domain. By construction $dist^M(\bar{b}_I, \bar{b}_{I^c}) > 2r' + 1$. This implies that $dist^M(N_{r'}^M(\bar{b}_I), N_{r'}^M(\bar{b}_{I^c})) > 1$. It remains to argue $dist^M(\bar{b}_I, \bar{a}_{I^c}) > 2r' + 1$. Suppose otherwise, $dist^M(\bar{b}_I, \bar{a}_{I^c}) \leq 2r' + 1$. Then $dist^M(b_1^*, \bar{a}^*) \leq (k-2)(2r'+1) + dist^M(\bar{b}_I, \bar{a}_{I^c}) \leq$

$(k-1)(2r'+1)$. This contradicts Property (iii) as long as we choose,

$$\ell \geq (k-1)(2r'+1). \tag{4.15}$$

Therefore ρ exists and the case is concluded.

In summary, during Case 4 the isomorphism π and the structure are modified, and the isomorphism radius is reduced to r' satisfying (4.10), (4.11), (4.14), and (4.15).

This completes the case analysis.

Choosing $\ell = k(2r'+1)$ and $s = (k-1)(2r'+1)$ we see that the conditions (4.8), (4.10), (4.11), (4.12), (4.13), (4.14), and (4.15) are satisfied for $r \geq (3k-1)(2r'+1)$. Since the lemma holds trivially when $r' < 1$, selecting $r = 9kr' = (3k)(3r') \geq (3k-1)(2r'+1)$ suffices. This is (4.7) in the statement of the lemma which completes the proof. ∎

4.5.3 Upper Bound for General Formulas

In this section we prove the general case of Theorem 16, which implies the upper bound in Theorem 3. The critical cases are the ones with positive alternation depth. In those cases, the idea is to iteratively apply Lemma 4.4 to reduce to the unary version of Theorem 16.

Suppose that a k-ary formula ϕ with alternation depth $d > 0$ is Arb-invariant and not $(\log n)^c$-local with respect to a structure M, where c is some positive constant. This means that there exists a violation in M to the $(\log n)^c$-locality of ϕ. Iteratively applying Lemma 4.4 yields a violation of the $(\gamma_k \cdot (\log n)^c)$-locality of some unary formula ϕ' of alternation depth d on some structure M' with $|M'| = |M| = n$, where γ_k only depends on k. For c' a constant such that $d+2 < c' < c$, and n sufficiently large such that $(\gamma_k \cdot (\log n)^c) \geq (\log n)^{c'}$, we obtain a contradiction with the unary version of Theorem 16 as long as $n \geq n_{\phi',c'}$. Thus, if we can upper bound the values $n_{\phi',c'}$ that can arise in the reduction from ϕ, we are done.

The upper bound follows because the number of different formulas ϕ' that can arise from ϕ is bounded. This is because in each case of the proof of Lemma 4.4, the resulting formula ϕ' consists of (i) an existential quantification over the marking relation to construct a tuple, and (ii) an evaluation of ϕ on the quantified tuple. The number of such formulas depends only on how the free variables are situated. This means that in the end the reduction produces only a bounded number of unary formulas, depending on ϕ.

We now formalize this argument.

Proof (of Theorem 16). We first remark that the case $d = 0$ in Theorem 16 trivially holds. To see this, consider a k-ary quantifier-free formula $\phi(\bar{x})$ of $\mathrm{FO}(\tau, \mathrm{Arb})$ that is Arb-invariant on a τ-structure M. We show that $\phi(\bar{x})$ is 0-local with respect to M. This implies that $\phi(\bar{x})$ is $(\log n)^c$-local with respect to M for any constant c. Let $n \doteq |M|$. Assume that $M \models \phi(\bar{a})$ and consider a tuple $\bar{b}$ such that $\mathcal{N}_0^M(\bar{a}) \cong \mathcal{N}_0^M(\bar{b})$. Consider any Arb-expansion M' of M. By Arb-invariance we have $M' \models \phi(\bar{a})$. Recall that the linear order of M' induces a bijection h' between $dom(M)$ and $[n]$. Let h'' be any bijection between $dom(M)$ and $[n]$ such that $h'(\bar{a}) = h''(\bar{b})$. This bijection induces a new linear order on M and a new Arb-expansion M'' of M. We claim that $M'' \models \phi(\bar{b})$. By Arb-invariance this implies $M \models \phi(\bar{b})$ as desired. From $\mathcal{N}_0^M(\bar{a}) \cong \mathcal{N}_0^M(\bar{b})$ we get that each atom of ϕ involving a relation in τ is true on M (and therefore on M' and M'') for $\bar{a}$ iff it is true for $\bar{b}$. From $h'(\bar{a}) = h''(\bar{b})$ we get that each atom of ϕ involving a numerical predicate is true for $\bar{a}$ on M' iff it is true for $\bar{b}$ on M''. As $\phi(\bar{x})$ is quantifier free we conclude that $M' \models \phi(\bar{a})$ iff $M'' \models \phi(\bar{b})$, which finishes the quantifier-free case.

Consider now the case of alternation depth $d > 0$. Suppose that $\phi(\bar{x})$ is a k-ary alternation-depth-d formula of $\mathrm{FO}(\tau, \mathrm{Arb})$ that is Arb-invariant with respect to a τ-structure M. Further suppose that $\phi(\bar{x})$ is not $(\log n)^c$-Gaifman local with respect to M, where $n \doteq |M| \geq n_{\phi,c}$ ($n_{\phi,c}$ will be determined later). The non-locality of ϕ is witnessed by two

tuples $\bar{a}$ and $\bar{b}$ on M. To ϕ and the witness $(M, \bar{a}, \bar{b})$ we can apply Lemma 4.4 at most $k-1$ times to produce a unary formula ϕ' with alternation depth d which is Arb-invariant and non-local with respect to a structure M'. This non-locality is witnessed by the elements a' and b' distinguished by ϕ', and an isomorphism between the $\lfloor \frac{(\log n)^c}{(9k)^{k-1}} \rfloor$-neighborhoods of a' and b'. Let c' be any constant such that $d+2 < c' < c$.

If we select $n_{\phi,c}$ satisfying $\lfloor \frac{(\log n_{\phi,c})^c}{(9k)^{k-1}} \rfloor \geq (\log n_{\phi,c})^{c'}$ and $n_{\phi,c} \geq n_{\phi',c'}$ we have a structure M', with $|M'| = n \geq n_{\phi',c'}$, where ϕ' is Arb-invariant on M', but not $(\log n)^{c'}$-local with respect to M'. This contradicts the unary version of this theorem. Therefore, it suffices to pick $n_{\phi,c}$ to be the maximum of $2^{(9k)^{\frac{k-1}{c-c'}}}$ and $n_{\phi',c'}$ for each ϕ' that may result from the iterated applications of Lemma 4.4. We now argue that the number of such formulas ϕ' is bounded by a constant.

Claim 4.1. *For a given formula ϕ, the number of different formulas ϕ' that can be produced by Lemma 4.4 for different choices of M, $\bar{a}$, and $\bar{b}$ is upper bounded by a function of k only.*

Proof. Consider each case of the proof of Lemma 4.4 and the formula produced. In all cases except Case 1, the formula is of the form

$$\phi'(\bar{y}) = (\exists \bar{z} \in dom(M')^{k-k'})\ R(\bar{y}, \bar{z}) \wedge \phi(\bar{y}, \bar{z}).$$

This induces at most $k-1$ different formulas because the range of k' is $1 \leq k' \leq k-1$. Note that Case 1 is slightly different in that R is a unary predicate, not a k-ary relation. So, taken together we have at most k basic formula types. However, we often (implicitly) relabeled the free variables for convenience of notation. These variations may induce distinct formulas as well. This relabeling can increase the number of distinct formulas by a factor of at most $k!$ (one for each ordering of the variables). We conclude that there are at most $k \cdot k!$ different formulas that the proof may produce. Note that bound is independent of M,

$\bar{a}$, $\bar{b}$, the Arb relations and even $|M|$. ∎

Since the number of possible ϕ' is bounded by a constant (depending on ϕ), $n_{\phi,c}$ can be selected to be such a constant as well. This concludes the proof. ∎

4.5.4 Lower Bound

For $c = 1$, the lower bound of Theorem 3 is implicit in [DLM07, Corollary 2]. For generalizing the result to arbitrary $c \geq 1$, the proof idea is as follows: We consider graphs represented as τ_E-structures, where τ_E is the schema consisting of a binary relation symbol E. We construct an Arb-invariant formula $\phi_c(y)$ which, when evaluated in a graph G, expresses that (i) G has less than $(\log n)^{c+1}$ non-isolated nodes (where n denotes the total number of nodes of G), and (ii) y is reachable from a node that lies on a triangle. To note that ϕ_c is *not* Gaifman $(\log n)^c$-local, consider, for a sufficiently large n, the graph G that consists of the disjoint union of

- a triangle, connected to a path of length $(\log n)^c + 1$,
- a path of length $(\log n)^c + 1$, and
- enough isolated nodes such that the total number of nodes of G is exactly n.

Let b and b' be the last nodes on the two paths present in G. Obviously, their $(\log n)^c$-neighborhoods are isomorphic. But b is reachable from a node that lies on a triangle, and b' is not. Since n is sufficiently large, the total number of non-isolated nodes is less than $(\log n)^{c+1}$. Thus, $G \models \phi_c(b)$ and $G \not\models \phi_c(b')$. In summary, ϕ_c is not Gaifman $(\log n)^c$-local.

For the construction of the formula ϕ_c, we use the following lemma. This lemma will also be used later on, in Section 4.6, for the proof of the lower bound of Theorem 4.

Lemma 4.5. *Let τ_{ES} be the schema consisting of a binary relation symbol E and a unary relation symbol S. For every integer $d \geq 1$ there is an Arb-invariant* $\mathrm{FO}(\tau_{ES}, Arb)$*-formula $reach_d(x, y)$ such that the following is true for all finite τ_{ES}-structures M, all elements a, b in S^M, and $n := |dom(M)|$:*

$$M \models reach_d(a,b) \iff |S^M| < (\log n)^d \text{ and there is a path from } a \text{ to } b \text{ in the induced subgraph of } G := (dom(M), E^M) \text{ on } S^M.$$

Proof. For the proof, we use the following technical result of [DLM07].

Lemma 4.6 (Corollary 1 in [DLM07]). *Let τ_S be the schema consisting of a unary relation symbol S. For every integer $d \geq 1$ there is a* $\mathrm{FO}(\tau_S, Arb)$*-formula $bij_d(x, y)$ such that the following is true for all τ_S-structures M, all Arb-expansions M' of M, all elements a, b in $dom(M)$, and $n := |dom(M)|$:*

$$M' \models bij_d(a,b) \iff |S^M| < (\log n)^d \text{ and } a \text{ is the } i\text{th largest element w.r.t. } <^{M'} \text{ in } S^M, \text{ where } i \text{ is the index of } b \text{ in } dom(M') \text{ w.r.t. } <^{M'}.$$

Note that the formula $bij_d(x, y)$ of Lemma 4.6 constitutes a bijection from the set S^M to an initial set of elements of $dom(M')$ (initial, with respect to the linear order present in the structure M'), and thus to the natural numbers $1, 2, \ldots, |S^M|$. This enables us to represent elements of S^M by natural numbers of size $\leq |S^M| < (\log n)^d$. Using its binary representation, we encode each such number by a binary string of length (exactly) $d \log\log n$.

Hence a sequence of elements of S^M of size bounded by $\ell(n) := \frac{\log n}{d \log\log n}$ can be represented using $\log n$ bits, i.e., a natural number $< n$ or, equivalently, an element of M. Given an element of M, using the appropriate numerical predicates present in the Arb-expansion M'

of M, we can extract from this element any of its blocks of length $d \log\log n$ and, using $bij_d(x, y)$, retrieve the corresponding element of S^M.

We can use this to construct an FO(τ_{ES}, Arb)-formula $\varrho(x, y)$ which, when evaluated in M', expresses that x and y are elements in S^M such that there is a path of length at most $\ell(n)$ in S^M from x to y: The formula $\varrho(x, y)$ simply guesses the path by existentially quantifying over the element of M representing its sequence and then checks that there is indeed an edge between any two consecutive nodes of this path. From the discussion above, this can be expressed in FO(τ_{ES}, Arb).

In summary, $\varrho(x, y)$ is an FO(τ_{ES}, Arb) formula such that the following is true for all finite τ_{ES}-structures M, all Arb-expansions M' of M, all elements a, b in S^M, and $n := |dom(M)|$:

$$M' \models \varrho(a, b) \iff |S^M| < (\log n)^d \text{ and there is a directed path from } a \text{ to } b \text{ of length} \leq \ell(n) \text{ in the induced subgraph of } G := (dom(M), E^M) \text{ on } S^M.$$

Thus, obviously, $\varrho(x, y)$ is Arb-invariant.

We iterate $\varrho(x, y)$ for a suitable number of times in order to obtain a formula for reachability $reach_d(x, y)$ by paths of length up to $(\log n)^d$: Let $\psi_1(x, y) := \varrho(x, y)$, and for $i \geq 2$, let $\psi_i(x, y)$ be the formula obtained from $\varrho(x, y)$ by replacing every atom of the form $E(z, z')$ by the formula $\psi_{i-1}(z, z')$. It is straightforward to see that $\psi_i(x, y)$ states that there is a path of length at most $\ell(n)^i$ in S^M from x to y.

For $i := d+1$, there exists an n_0 such that for all $n > n_0$ we have $\ell(n)^i \geq (\log n)^d > |S^M|$. Therefore, we can choose $reach_d(x, y)$ to be the formula stating that either $|dom(M)| > n_0$ and $\psi_i(x, y)$ holds, or, for some $\ell \in \{1, \ldots, n_0\}$, we have $|dom(M)| = \ell$ and $|S^M| \leq (\log \ell)^d$, and x and y are nodes in S^M such that y is reachable from x by a path of length $\leq \ell$ that uses only nodes in S^M (note that for each fixed ℓ this can be expressed in FO(τ_{ES})). This

concludes the proof of Lemma 4.5. ∎

Remark 4.1. *Note that the formula $bij_d(x, y)$ of Lemma 4.6 only relies on the numerical predicates $+$ and $\times$. Furthermore, the rest of the above proof can be accomplished by using just the linear order $<$ and the predicates for addition and multiplication, cf., e.g., the textbook [Imm99]. This means that the lower bound of Theorem 3 actually holds for uniform* AC^0.

We are now ready for the proof of the lower bound of Theorem 3.

Proof (of Theorem 3 – Lower Bound). Let τ_E be the schema consisting of a binary relation symbol E, let $d := c + 1$, and let $reach_d(x, y)$ be the Arb-invariant formula provided by Lemma 4.5. Let $\varrho(x, y)$ be the formula obtained from $reach_d(x, y)$ by replacing every atomic formula of the form $S(z)$ by a formula stating that z is a non-isolated node (i.e., by the formula $\exists z'\big(E(z, z') \vee E(z', z)\big)$). Clearly, when evaluated in a τ_E-structure M, the formula $\varrho(x, y)$ states that there are less than $(\log n)^d$ non-isolated nodes in M, x and y are non-isolated, and there is a path from x to y.

Let $\phi_c(y)$ be the formula

$$\exists x \exists x_1 \exists x_2 \big(E(x, x_1) \wedge E(x_1, x_2) \wedge E(x_2, x) \wedge \varrho(x, y)\big).$$

Obviously, ϕ_c is Arb-invariant, since ϱ is Arb-invariant. Furthermore, when evaluated in a τ_E-structure M, the formula ϕ_c expresses that (i) there are less than $(\log n)^d$ non-isolated nodes (where n denotes the size of $dom(M)$), and (ii) y is reachable from a node that lies on a triangle.

By the reasoning given at the beginning of Section 4.5.4, ϕ_c is not Gaifman $(\log n)^c$-local. This completes the proof of the lower bound in Theorem 3. ∎

4.6 Hanf Locality for String Structures

In Section 4.5 we showed that Arb-invariant FO formulas are Gaifman $(\log n)^{O(1)}$-local. We are not able to prove that Arb-invariant FO formulas are also Hanf $(\log n)^{O(1)}$-local in general but are able to do so in the special case when the structures represent strings.

Fix a finite alphabet A and consider structures over the schema τ_s containing one unary predicate per element of A and one binary predicate E. Let $\mathcal{S}$ be the class of τ_s-structures M that interpret E as a successor relation and where each element of M belongs to exactly one of the unary predicates in τ_s. Each structure in $\mathcal{S}$ represents a string in the obvious way and we blur the distinction between a string w and its actual representation as a structure. We then consider $\mathrm{FO}(\tau_s \cup \sigma_{\mathrm{arb}})$ formulas that are Arb-invariant over all structures in $\mathcal{S}$ and denote the corresponding set of formulas by Arb-invariant FO(*Succ*). We say that a language $L \subseteq A^*$ is definable in Arb-invariant FO(*Succ*) if there is a sentence of Arb-invariant FO(*Succ*) whose set of models in $\mathcal{S}$ is exactly L.

The goal of this section is to prove Theorem 4. The lower bound part will be proved in Section 4.6.4. For the upper bound part we actually show the following result:

Theorem 17. *Arb-invariant* FO(*Succ*) *formulas with alternation depth d are Hanf $(\log n)^c$-local for any constant $c > d + 2$.*

The crux of Theorem 17 is the case where the formula is a sentence. For that reason, we only consider sentences in Sections 4.6.1 and 4.6.2. We return to the general case in Section 4.6.3.

The proof of Theorem 17 for sentences consists of two parts. In Section 4.6.1 we introduce a closure property of languages allowing to swap substrings inside a string without affecting membership in the language as long as the neighborhoods around the endpoints of the substrings look similar. We then show that a language being closed under swaps is equivalent to the language being Hanf local, where the size of the boundary neighborhoods

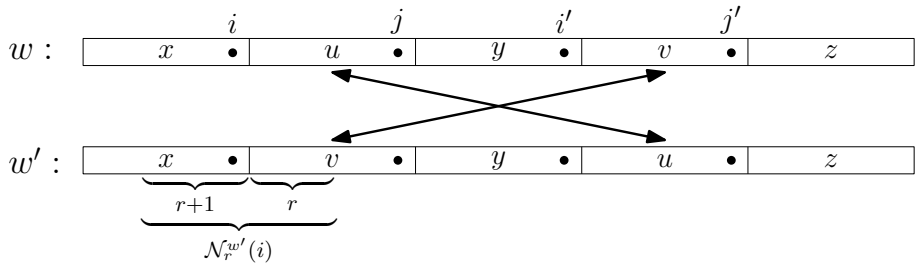


Figure 4.9: r-swapping $w = xuyvz$ to $w' = xvyuz$.

is essentially the isomorphism radius. In Section 4.6.2 we show that languages definable in Arb-invariant FO(*Succ*) are closed under this swap operation for boundary neighborhoods of radius $(\log n)^{O(1)}$. We conclude in Section 4.6.3 by combining the two previous results and derive Theorem 17.

4.6.1 Connection with Closure under Swaps for Sentences

In this section we introduce the key notion of a swap. It is an operation that exchanges two substrings inside a string as long as the neighborhoods around the endpoints of the substrings look similar. Our notion of a swap is somewhat related to a similar notion that was introduced in [TW85] for regular languages (see also [BP89, BS10]).

Let $w \in A^*$, $i, j \in \mathbb{N}$, define $w[i, j]$ to be the substring of w starting at position i of w and ending at position j. Let $n = |w|$ and $r > 0$, then the *r-suffix* of w is $w[n - r + 1, n]$ and the *r-prefix* of w is $w[1, r]$. Notice that if i is the last position of u in the string $w = uv$ then $\mathcal{N}_r^w(i)$ is the concatenation of the $(r + 1)$-suffix of u with the r-prefix of v.

Let $r \in \mathbb{N}$ and $w \in A^*$. A string $w' \in A^*$ is obtained from w by a *r-swap operation* if $w = xuyvz$, $\mathcal{N}_r^w(i) \cong \mathcal{N}_r^w(i')$ and $\mathcal{N}_r^w(j) \cong \mathcal{N}_r^w(j')$ where i, j, i', and j' are, respectively, the positions in w immediately before the substrings u, y, v, and z, and $w' = xvyuz$. See Figure 4.9 for a diagram.

Let $r : \mathbb{N} \to \mathbb{R}_{\geq 0}$. A language L is said to be *closed under r(n)-swaps* if there exists a

$n_0 \in \mathbb{N}$ such that for all strings $w, w' \in A^*$, with $|w| = n > n_0$, if w' is obtained from w by a $r(n)$-swap operation then we have:

$$w \in L \qquad \text{iff} \qquad w' \in L.$$

Informally, a language is closed under swaps if the language is unable to distinguish the relative order of substrings whose local neighborhoods look the same.

There are tight connections between closure under swaps and Hanf locality. The first one is that an r-swap operation does not change the $\equiv_r$-class of a string. This follows from the observation that an r-swap preserves r-neighborhoods. The second one concerns the opposite direction: If two strings are in a same $\equiv_r$-class then there is a sequence of $(r-1)$-swap operations transforming one into the other. Intuitively this is shown as follows. Assuming that $w \equiv_r w'$, we transform w' into w using $(r-1)$-swaps to embed larger and larger prefixes of w within the transformed string. Here an embedding is a partial function on string positions preserving r-neighborhoods and the string order (i.e., the relative ordering of the positions in the prefix of w is preserved by the embedding). Eventually, the entirety of w embeds into the transformed string, and because $|w| = |w'|$ this implies that the transformed string coincides with w. Thus, we have transformed w' into w via a sequence of $(r-1)$-swaps.

We now sketch the transformation procedure. See Figure 4.10 for a diagram of this construction. In the i^{th} step of the procedure, we consider the i-prefix of w that we assume embeds into the string $w_i \equiv_r w$ via the embedding h_i. Our goal is to extend the embedding to the $(i+1)$-prefix of w while preserving the $\equiv_r$-class. Let $j \doteq h_i(i)$. Because $w \equiv_r w_i$, there is a position j' in w_i, outside the image of h_i, that has the same r-neighborhood as $i+1$. If $j' > j$, mapping $i+1$ to j' preserves the ordering of w and extends the embedding h_i to the $(i+1)$-prefix of w. Otherwise, we have $j' < j$. Let i_1 be the maximal position

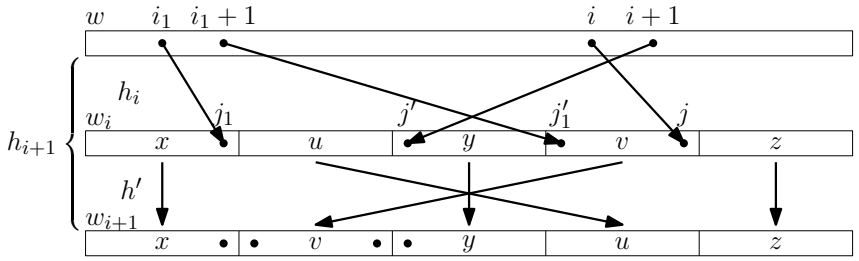


Figure 4.10: Constructing h_{i+1} from h_i and w_i.

in the i-prefix of w such that $h_i(i_1) < j'$. By maximality of i_1, we have the following relative positions within w_i: $h_i(i_1) < j' < h_i(i_1 + 1) \leq j$. The key observation is that because i and $j = h_i(i)$ have the same r-neighborhood then $j' - 1$ and j have the same $(r-1)$-neighborhood. As the same consequence can be derived for $h_i(i_1) + 1$ and $h_i(i_1 + 1)$, we can $(r-1)$-swap the substrings of w_i with these endpoints (i.e., substrings u and v in Figure 4.10). We then observe that the $(i+1)$-prefix of w embeds into the resulting string w_{i+1} and that $w_{i+1} \equiv_r w_i$, so w_{i+1} remains in the same $\equiv_r$-class as w. Initializing $w_1 = w'$ and $h_1 : w \equiv_r w'$ establishes the conditions required to start the procedure.

Lemma 4.7. *Let $r \in \mathbb{N}$, and $w, w' \in A^*$.*

1. *If w' is obtained from w by a r-swap operation then $w \equiv_r w'$.*
2. *If $w \equiv_r w'$ then there is a finite sequence of $(r-1)$-swap operations transforming w into w'.*

Proof. Fix $r \in \mathbb{N}$.

Part 1.

Assume $w = xuyvz$, $w' = xvyuz$, and for i, j, i' and j' which are, respectively, the positions immediately before u, y, v, and z in w, we have $\mathcal{N}_r^w(i) \cong \mathcal{N}_r^w(i')$ and $\mathcal{N}_r^w(j) \cong \mathcal{N}_r^w(j')$. Let h be a bijection from w to w' that sends each block x, u, y, v, z to its corresponding block in w'. In other words, h sends the first letter of x in w to the first letter of x in w'

and so on; h acts on the other substrings u, y, v, and z in a similar fashion. We show that h preserves r-neighborhoods. It is enough to show this for the harder cases, i.e., the boundary cases. By symmetry we only need to consider the boundaries of y.

Recall that j is the position immediately before y, i.e., the last position of u. Let $k \doteq h(j)$ be the last position of u in w'. We want to show $\mathcal{N}_r^{w'}(k) \cong \mathcal{N}_r^w(j)$. First consider the right part of the respective neighborhoods. In w this is the r-prefix of yvz while in w' it is the r-prefix of z. By hypothesis, $\mathcal{N}_r^w(j) \cong \mathcal{N}_r^w(j')$, and the r-prefix of z is the same as the r-prefix of yvz, so we are done.

Now consider the left part of the respective neighborhoods. In w it is xu while in w' it is $xvyu$. We need to show that they have the same $(r+1)$-suffix. From $\mathcal{N}_r^w(i) \cong \mathcal{N}_r^w(i')$ we know that x and xuy have the same $(r+1)$-suffix; this implies that xv and $xuyv$ have the same $(r+1)$-suffix. From $\mathcal{N}_r^w(j) \cong \mathcal{N}_r^w(j')$ we get that $xuyv$ and xu have the same $(r+1)$-suffix. By combining these two facts we see that xu and xv have the same $(r+1)$-suffix. Therefore xuy and xvy have the same $(r+1)$-suffix. Hence x and xvy have the same $(r+1)$-suffix, and thus xu and $xvyu$ do as well, as desired.

The other boundary of y, position i', is treated similarly.

Part 2.

Let w and w' be two strings of A^* such that $w \equiv_r w'$ and $|w| = n$. Let h be a bijection witnessing $w \equiv_r w'$.

We construct by induction a sequence of strings $w_1, \cdots, w_n$ such that (i) $w_1 = w'$, (ii) for $i \leq n$, $w \equiv_r w_i$ via a bijection h_i verifying $\forall j < j' \leq i, \;\; h_i(j) < h_i(j')$, and (iii) w_{i+1} is either w_i or is obtained from w_i via a $(r-1)$-swap operation. Note that (ii) implies that h_n is the identity and therefore $w_n = w$. Properties (i) and (iii) imply that $w = w_n$ is obtained from $w' = w_1$ by a finite sequence of $(r-1)$-swap operations, proving the result.

The base case is immediate by setting $w_1 \doteq w'$ and $h_1 \doteq h$.

Suppose we have constructed w_i and h_i satisfying the inductive properties (i), (ii), and (iii) up to i. Let $j \doteq h_i(i)$ and $j' \doteq h_i(i{+}1)$. If $j' > j$, (ii) is already satisfied and we are done. Assume now that $j' < j$. Let $i_1 < i$ be the position in w such that $h_i(i_1) < j' < h_i(i_1 + 1)$, and let $j_1 \doteq h_i(i_1)$ and $j_1' \doteq h_i(i_1 + 1)$. Note an index i_1 such that $h_i(i_1) < j'$ exists because h_i preserves r-neighborhood-types, and therefore must map the element of w with index 1 to the element of w_i with index 1 (i.e., $1 = h_i(1) < j'$). See Figure 4.10 for a diagram of the construction.

Now, notice that because h_i preserves r-neighborhood-types and j_1, j_1' are the images of consecutive positions in w, $\mathcal{N}^{w_i}_{(r-1)}(j_1) \cong \mathcal{N}^{w_i}_{(r-1)}(j_1' - 1)$. For the same reason, $\mathcal{N}^{w_i}_{(r-1)}(j' - 1) \cong \mathcal{N}^{w_i}_{(r-1)}(j)$.

Hence our string w_i can be decomposed as $xuyvz$ where (in the case where $j' = j_1 + 1$, u is the empty string):

$$
\begin{aligned}
x &\doteq w_i[1, j_1], \\
u &\doteq w_i[j_1 + 1, j' - 1], \\
y &\doteq w_i[j', j_1' - 1], \\
v &\doteq w_i[j_1', j], \\
z &\doteq w_i[j + 1, n]
\end{aligned}
$$

and the conditions for a $(r-1)$-swap hold. Note that this swap induces a permutation h' on the positions of w_i.

We set $w_{i+1} \doteq xvyuz$ and condition (iii) holds. We now set $h_{i+1} \doteq h' \circ h_i$, the composition of h_i and h'. We have the following claim.

Claim 4.2. $h_{i+1} : w \equiv_r w_{i+1}$

Proof. We show that $h' : w_i \equiv_r w_{i+1}$. The claim then follows because $h_i : w \equiv_r w_i$. We argue that the r-neighborhoods of the substrings x, u, y, v, and z are identical in both w_i and w_{i+1}, hence $w_i \equiv_r w_{i+1}$.

We start by deriving a few identities from our hypothesis. For a string s, we use the notation $P(s)$ to denote the r-prefix of s and $S(s)$ to denote the r-suffix of s. Consider $S(x)$. Because $\mathcal{N}_r^w(i_1) \cong \mathcal{N}_r^{w_i}(j_1)$, $S(x) = S(w[1, i_1])$. Moreover, as $\mathcal{N}_r^w(i_1 + 1) \cong \mathcal{N}_r^{w_i}(j_1')$, we have $S(w[1, i_1]) = S(xuy)$. Hence $S(x) = S(xuy)$. Similarly the known neighborhood isomorphisms give us the following facts, where the text in the square brackets indicates which neighborhoods the identities address, e.g., "left nbh of v" means that the identity shows that the strings of length r preceding v in w and w' are the same.

$$S(x) = S(xuy) \qquad \text{[left nbh of } v\text{]} \tag{4.16}$$

$$S(xu) = S(xuyv) \tag{4.17}$$

$$P(z) = P(yvz) \qquad \text{[right nbh of } u\text{]} \tag{4.18}$$

$$P(vz) = P(uyvz) \tag{4.19}$$

We can derive a number of implications using (4.16)-(4.19).

$$(4.16) \wedge (4.17) \Rightarrow S(xv) = S(xuyv) = S(xu) \qquad \text{[left nbh of } y\text{]} \tag{4.20}$$

$$\Rightarrow S(xuy) = S(xvy) \Rightarrow_{(4.16)} S(x) = S(xvy) \qquad \text{[left nbh of } u\text{]} \tag{4.21}$$

$$\Rightarrow S(xu) = S(xvyu) \Rightarrow_{(4.17)} S(xuyv) = S(xvyu) \qquad \text{[left nbh of } z\text{]}$$

$$(4.18) \wedge (4.19) \Rightarrow P(uz) = P(uyvz) = P(vz) \qquad \text{[right nbh of } y\text{]} \tag{4.22}$$

$$\Rightarrow P(yvz) = P(yuz) \Rightarrow_{(4.18)} P(z) = P(yuz) \qquad \text{[right nbh of } v\text{]}$$

$$\Rightarrow P(vz) = P(vyuz) \Rightarrow_{(4.19)} P(uyvz) = P(vyuz) \qquad \text{[right nbh of } x\text{]} \tag{4.23}$$

With these facts in hand we can argue that the r-neighborhoods of each substring x, u, y, v, z are identical in w_i and w_{i+1}. As before we only prove it for the boundary cases.

Consider first x and its last position j_1. Notice that $h'(j_1) = j_1$. In order to show that $\mathcal{N}_r^{w_i}(j_1) \cong \mathcal{N}_r^{w_{i+1}}(j_1)$ it remains to show that the r-prefix of $uyvz$ is the same as the r-prefix of $vyuz$. This is (4.23).

Now consider u. If u is empty, h' trivially preserves the r-neighborhoods of the elements in u. Otherwise, assume that u is not empty and consider its first position $j_1 + 1$ and let $k \doteq h'(j_1 + 1)$. In order to show that $\mathcal{N}_r^{w_i}(j_1 + 1) \cong \mathcal{N}_r^{w_{i+1}}(k)$ we need to show that $S(x) = S(xvy)$ and that the $(r+1)$-prefix of $uyvz$ is the same as the $(r+1)$-prefix of uz. The former is (4.21) and the latter is immediate from (4.18) as u is not empty. Consider now the last position $j' - 1$ of u and let $k' \doteq h'(j' - 1)$. In order to show that $\mathcal{N}_r^{w_i}(j' - 1) \cong \mathcal{N}_r^{w_{i+1}}(k')$ we need to show that the $(r + 1)$-suffix of xu is the same as the $(r + 1)$-suffix of $xvyu$ and that $P(yvz) = P(z)$. The latter is (4.18) while the former is immediate from (4.21) as u is not empty.

Finally consider the first position j' of y. Let $k \doteq h'(j')$. In order to show that $\mathcal{N}_r^{w_i}(j') \cong \mathcal{N}_r^{w_{i+1}}(k)$ we need to show that $S(xu) = S(xv)$ and that the $(r + 1)$-prefix of yvz is the same as the $(r + 1)$-prefix of yuz. The former is (4.20) while the latter is immediate from (4.22) as y is not empty.

The other cases are treated similarly by symmetry. (Claim 4.2)■

Observe that for all $k \leq i_1$, $h_i(k)$ maps into x, and for all $k \in [i_1 + 1, i]$, $h_i(k)$ maps into v. Since $h_i(k)$ is monotone for $k \leq i$ and $h_{i+1}(i + 1)$ maps onto j', $h_{i+1}(k)$ is monotone for $k \leq i + 1$. Combining this fact with the claim implies that we have extended property (ii) to $i + 1$. With all properties satisfied the induction step is complete. ■

4.6.2 Closure under Swaps

We now show that a language definable by a sentence of Arb-invariant FO(*Succ*) is closed under $(\log n)^c$-swaps for some constant c.

Lemma 4.8. *If L is a language definable by an Arb-invariant* FO(*Succ*) *sentence with alternation depth d then L is closed under $(\log n)^c$-swaps for any constant $c > d + 2$.*

To prove this lemma we observe that it suffices to consider swaps where the various neighborhoods are disjoint. This is because (i) when the isomorphic neighborhoods involved have substantial overlap, the swap operation has no effect and trivially preserves membership to L, and (ii) when the isomorphic neighborhoods have small overlap, restricting their radius slightly yields isomorphic neighborhoods that are disjoint. In Section 4.6.2.1 we show that Arb-invariant FO(*Succ*) is closed under disjoint swaps, and in Section 4.6.2.2 we formalize (i) and (ii) by showing that closure under disjoint swaps implies closure under general swaps modulo a small constant factor increase in the isomorphism radius. We combine these two steps to prove Lemma 4.8.

4.6.2.1 Disjoint Swaps

We weaken the condition for closure under r-swaps slightly by only considering neighborhoods $\mathcal{N}_r^w(i)$, $\mathcal{N}_r^w(i')$, $\mathcal{N}_r^w(j)$, and $\mathcal{N}_r^w(j')$ which are pairwise disjoint. We call this closure under *disjoint* r-swaps. We argue that languages definable in Arb-invariant FO(*Succ*) are closed under disjoint $(\log n)^c$-swaps, for some constant c depending on the alternation-depth of the language. One way to prove this fact is by mimicking our proof of Gaifman locality for the special case of string structures. Alternatively, we can use our Gaifman locality result as a blackbox. We follow the latter approach. The idea is that given a pair of strings w and w' of length n which witness the violation of the closure-under-$(\log n)^c$-swaps property for a sentence ϕ, we can derive: (i) a τ-structure M of size n with two tuples that have

isomorphic neighborhoods up to distance $\Omega((\log n)^c)$, and (ii) an FO(τ, Arb) formula ψ distinguishing these tuples that is Arb-invariant with respect to M. We instantly conclude by applying our Gaifman locality theorem (Theorem 16) to produce a contradiction.

Proposition 4.1. *If L is a language definable by an Arb-invariant* FO(*Succ*) *sentence with alternation depth d then L is closed under* disjoint $(\log n)^c$*-swaps for any constant $c > d+2$.*

Proof. Let ϕ be an Arb-invariant FO(*Succ*) sentence with alternation depth d defining L. Suppose that L is not closed under disjoint r-swaps, where $r \doteq (\log n)^c$ and c is a large enough constant depending only on ϕ that will become apparent during the proof. Then there exists an infinite class of equal-length string pairs $\mathcal{W}$, such that for every pair $\langle w, w' \rangle \in \mathcal{W}$, the conditions for disjoint r-swaps are satisfied for the pair, but $w \in L$ and $w' \notin L$.

Consider one such pair $\langle w, w' \rangle \in \mathcal{W}$ and let $n \doteq |w| = |w'|$. Let u, v, x, y, z be as in the definition of r-swaps, with $w = xuyvz$ and $w' = xvyuz$. Let i, i', j, j' denote the positions in w supplied by the definition of r-swaps. Using the assumed disjointness property, the neighborhoods $\mathcal{N}_r^w(i)$, $\mathcal{N}_r^w(i')$, $\mathcal{N}_r^w(j)$, and $\mathcal{N}_r^w(j')$ are all disjoint, and i, i', j, and j' are distinct positions in w.

Recall that w and w' can be seen as labeled graphs where the edge relation is called E. Consider the schema containing three extra binary relations E_1, E_2, and E_3. We construct a structure M from w over this extended schema by slightly modifying E. This construction is diagrammed in Figure 4.11. The purpose of the relations E_1, E_2, and E_3 is to mark the boundary vertices of u and v so that the boundaries can easily be recovered, and at the same time ensure that the neighborhoods around u and v in M appear identical. Note that the E-edges leaving i, j, i', and j' in w are eliminated in M, and all other E-edges of w are unchanged. M has the following property.

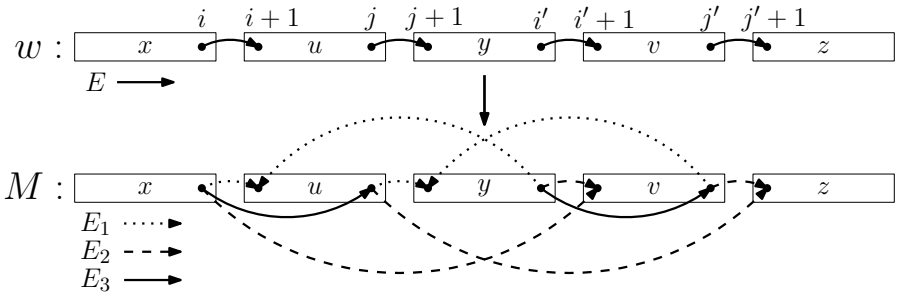


Figure 4.11: Constructing the structure M from the string w.

Claim 4.3. $\mathcal{N}_{r-1}^{M}(i,j,i',j') \cong \mathcal{N}_{r-1}^{M}(i',j',i,j)$.

Proof. To show this, we describe a witnessing isomorphism π. Let π act as the identity on $(r-1)$-prefixes of u, y, v, and z. Let π take the r-suffix of x to the r-suffix of y and *vice versa*. Let π take the r-suffix of u to the r-suffix of v and *vice versa*. This mapping is well-defined because the r-neighborhoods around i, j, i', and j' are all disjoint. It is now routine to verify that π is an isomorphism as claimed using the facts that $\mathcal{N}_r^w(i) \cong \mathcal{N}_r^w(i')$, $\mathcal{N}_r^w(j) \cong \mathcal{N}_r^w(j')$, and that these neighborhoods are disjoint. ∎

We can use the Arb-invariant FO(*Succ*) sentence ϕ to construct a formula ψ with four free variables x_1, y_1, x_2, y_2 that does the following: When evaluated in M, $\psi(i,j,i',j')$ simulates ϕ on w while $\psi(i',j',i,j)$ simulates ϕ on w', and for all other tuples ψ rejects. The formula ψ is constructed from ϕ as follows. In ϕ, replace all atoms $E(x,y)$ with

$$\theta(x,y,x_1,y_1,x_2,y_2) \doteq E(x,y) \vee ((x = x_1 \vee x = y_1) \wedge E_1(x,y)) \\ \vee ((x = x_2 \vee x = y_2) \wedge E_2(x,y)).$$

In order to ensure that on M ψ rejects when the tuple is not (i,j,i',j') or (i',j',i,j),

we explicitly test for these inputs using the E_3-edge and reject if not found.

$$\begin{aligned}\psi(x_1, y_1, x_2, y_2) \doteq\ & E_3(x_1, y_1) \wedge E_3(x_2, y_2) \wedge x_1 \neq x_2 \\ & \wedge \phi_{E(x,y) \leftarrow \theta(x,y,x_1,y_1,x_2,y_2)}(x_1, y_1, x_2, y_2).\end{aligned}$$

Here the notation indicates that we are replacing all occurrences of the relation $E(x, y)$ in ϕ by the relation $\theta(x, y, x_1, y_1, x_2, y_2)$. ψ is Arb-invariant with respect to M. To see this, observe that when the input tuple is not (i, j, i', j') or (i', j', i, j), the formula always rejects. In the other case ϕ is effectively evaluated on either w or w', which are strings, hence the action of ϕ is Arb-invariant by hypothesis.

Observe that ψ is a 4-ary formula that is defined only with respect to ϕ, and has the same alternation depth as ϕ. Applying Theorem 16 to ψ we see that for any constant $c' > d + 2$ and for $n \geq n_{\psi,c'}$, ψ is Gaifman $(\log n)^{c'}$-local for structures for which it is Arb-invariant. Now, the infinite class of r-swap closure violations $\mathcal{W}$ with respect to ϕ induces an infinite class of structures $\mathcal{M}$ with Gaifman $(r-1)$-locality violations with respect to the formula ψ, where each violation is of the form $\langle M, \bar{a} \doteq (i, j, i', j'), \bar{b} \doteq (i', j', i, j)\rangle$, and ψ is Arb-invariant with respect to the structures in $\mathcal{M}$. If we pick $c' < c$, this infinite class of violations allows us to select an input length n which is at least $n_{\psi,c'}$ and large enough to make $r - 1 = (\log n)^c - 1 \geq (\log n)^{c'}$. This violates Theorem 16, and completes the proof. ■

4.6.2.2 From Disjoint Swaps to General Swaps

We now argue that languages which are closed under disjoint swaps are also closed under swaps. Consider a language L which is closed under disjoint $r'(n)$-swaps and a pair of sufficiently long strings $w = xuyvz$ and $w' = xvyuz$ which satisfy the isomorphism conditions of an $r(n)$-swap. It suffices to argue that L does not distinguish w and w'.

We observe that when the neighborhoods of the substrings overlap a large amount the neighborhood isomorphisms induce periodic behavior within the $r(n)$-neighborhoods, so much so that the substrings uyv and vyu become identical. This implies that $w = w'$, and hence $w \in L$ iff $w' \in L$. This takes care of the case of large neighborhood overlap. We then focus on the case where the $r(n)$-neighborhoods of the substrings only overlap a small amount and show that there is freedom to select slightly smaller neighborhoods (of radius $r'(n)$) that are pairwise disjoint, though still induce the same effective swapping. This allows us to apply the closure of L under disjoint $r'(n)$-swaps to conclude that L does not distinguish w and w'. We now formalize this approach.

Proposition 4.2. *Let L be a language and r be a function $\mathbb{N} \to \mathbb{R}_{\geq 0}$. If L is closed under disjoint $\frac{r(n)}{14}$-swaps then L is closed under $r(n)$-swaps.*

Proof. Suppose that L is closed under disjoint $r'(n)$-swaps, where $r'(n) = r(n)/c$ for some constant c to be determined later. Let n_0 be the associated constant. Fix a length $n \geq n_0$, and a pair of length n strings $w \doteq xuyvz$ and $w' \doteq xvyuz$ whose substrings satisfy the isomorphism conditions for a $r(n)$-swap. To prove the lemma it suffices to argue that $w \in L$ iff $w' \in L$. We drop the parameter to r' and r in what follows.

The isomorphism conditions of r-swaps imply it suffices to consider $|x| \geq r$. Otherwise, to satisfy the conditions it must be that $|u| = |y| = |v| = 0$, and hence $w = xz = w'$, and we trivially conclude $w \in L$ iff $w' \in L$. Analogously, $|z| \geq r$.

We proceed by case analysis on the sizes of u, y, and v relative to r'. If the lengths of u, y, and v are all short relative to r', then their respective neighborhoods overlap and u, y, and v are present in each neighborhood. The neighborhood isomorphisms drag these three substrings around implying that $uyv = vyu$, and hence that $w = w'$. This is Case 0.

In the three remaining cases we use the closure of L under disjoint r'-swaps to conclude that L does not distinguish w and w'. In each of these cases, we determine a set of disjoint

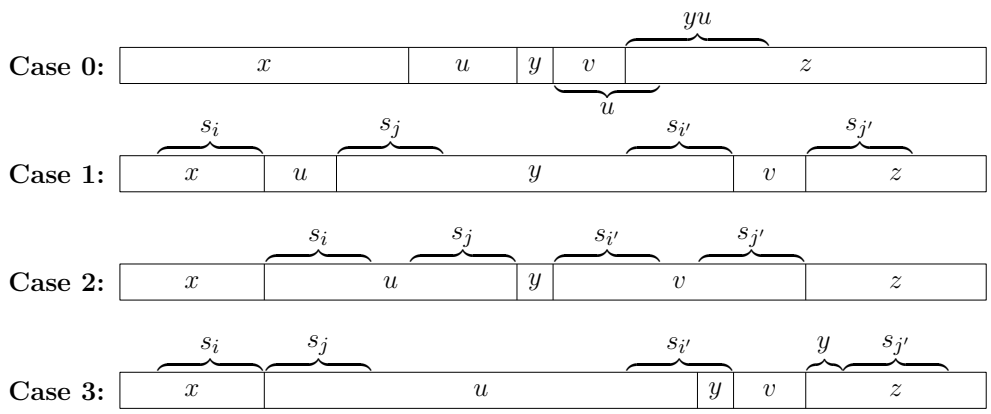


Figure 4.12: The cases in the proof of Lemma 4.8.

substrings $s_i, s_j, s_{i'}, s_{j'}$ occurring in this order in w, each of length $2r'$. We show that $s_i = s_{i'}$ and $s_j = s_{j'}$. Let i, j, i', j' be the respective middle positions of these substrings, then the r'-neighborhoods of i and i', and of j and j' are isomorphic. We argue that the result of swapping the substrings $w[i+1, j]$ and $w[i'+1, j']$ in w yields w', and thus w and w' are separated by a disjoint r'-swap. Since L is closed under disjoint r'-swaps and $n \geq n_0$, $w \in L$ iff $w' \in L$, concluding the proof in each case. See Figure 4.12 for a diagram of the four cases.

Case 0. $|u|, |y|, |v| < 4r'$.

Let i, i', j, and j' be the positions in w preceding the substrings u, y, v, and z, respectively. We claim that if $r \geq 12r'$ then $uyv = vyu$. This implies that $w = w'$ and hence $w \in L$ iff $w' \in L$. To see the claim, notice first that because $r \geq 4r'$ and $|u| < 4r'$, the right part of the r-neighborhood around i starts with u. Therefore, as $\mathcal{N}_r^w(i) \cong \mathcal{N}_r^w(i')$, u is a prefix of vz. A similar reasoning around j and j', using the fact that $r \geq 8r'$, shows that z and yvz have the same prefix of length $8r'$. As u is a prefix of vz and $|yu| < 8r'$, this implies that yu is a prefix of z. Therefore vyu is a prefix of vz. Now, because $r \geq 12r'$ and $\mathcal{N}_r^w(i) \cong \mathcal{N}_r^w(i')$, $uyvz$ and vz have the same prefix of length $12r'$. Because $|vyu| < 12r'$, this implies that vyu is a prefix of $uyvz$ and therefore $vyu = uyv$.

We now consider the cases with less overlap between neighborhoods.

Case 1. $|y| \geq 4r'$.

Let s_i be the $2r'$-suffix of x and $s_{i'}$ be the $2r'$-suffix of y. Let s_j be the $2r'$-prefix of y and $s_{j'}$ be the $2r'$-prefix of z. Since $2r' \leq |y|$, s_j and $s_{i'}$ do not overlap. Since $|x|, |z| \geq r > 2r'$, s_i and $s_{j'}$ are fully realized. The given neighborhood isomorphisms imply that $s_i = s_{i'}$ and $s_j = s_{j'}$. Inspection shows that swapping $w[i+1, j]$ with $w[i'+1, j']$ produces w', completing the case.

Case 2. $|u|, |v| \geq 4r'$.

Let s_i be the $2r'$-prefix of u and $s_{i'}$ be the $2r'$-prefix of v. Let s_j be the $2r'$-suffix of u and $s_{j'}$ be the $2r'$-suffix of v. Since $|u|, |v| \geq 4r'$ these substrings $s_i, s_{i'}, s_j, s_{j'}$ are pairwise disjoint. Further, using the known neighborhood isomorphisms, $s_i = s_{i'}$ and $s_j = s_{j'}$. Inspection shows that swapping $w[i+1, j]$ with $w[i'+1, j']$ produces w', completing the case.

Case 3. $|v|, |y| < 4r', |u| \geq 4r'$ (analogously, $|u|, |y| < 4r', |v| \geq 4r'$).

Let s_i be the $2r'$-suffix of x and $s_{i'}$ be the $2r'$-suffix of uy. Using the given neighborhood isomorphisms and the fact $|u| \geq 4r'$, s_i and $s_{i'}$ are disjoint and equal. Let s_j be the $2r'$-prefix of u. Since $|u| \geq 4r'$, s_j and $s_{i'}$ are disjoint. Let $s_{j'} \doteq z[|y|, |y| + 2r']$. Since $|z| \geq r$ and $|y| < 4r'$, $s_{j'}$ is fully realized if $6r' < r$. We need to argue that $s_j = s_{j'}$.

Observe that the last element of u and the first element of z are within $8r'$ of each other. Without loss of generality $|yv| > 0$, because otherwise $w = xuz = w'$. The fact that r-neighborhoods around the point at the end of u and the point immediately before z are isomorphic implies that the neighborhood following v contains a sequence of many repetitions of the string yv. This string yv repeats up to distance at least $r - 8r'$ into z. Therefore, if $14r' \leq r$, the string $s_{j'}$ starts with v followed by repetitions of yv for the entire length of $s_{j'}$, and is the same as $2r'$-prefix of vz. Hence the neighborhood isomorphism between the last point of x and the point preceding v implies that $s_j = s_{j'}$.

Write $z = yz'$, then $w = xuyvyz'$. Swapping the substring vy with the empty string between x and u produces $xvyuyz' = xvyuz = w'$. This completes the case.

Choosing $c = 14$ suffices to satisfy the assumptions made in each case and completes the proof. ∎

Combining Proposition 4.1 and Proposition 4.2 yields a proof of Lemma 4.8.

Proof (of Lemma 4.8). Let L be a language definable by an Arb-invariant FO(*Succ*) sentence with alternation depth d. For any constant $c' > d + 2$ we have, by Proposition 4.1, that L is closed under disjoint $(\log n)^{c'}$-swaps. By Proposition 4.2, L is closed under $14(\log n)^{c'}$-swaps. If we pick $c' < c$, then $14(\log n)^{c'} < (\log n)^c$ for n sufficiently large. It follows that L is closed under $(\log n)^c$-swaps. ∎

4.6.3 Upper Bound

We are now ready to prove Theorem 17. It is essentially a combination of Lemma 4.8 and Lemma 4.7 with a reduction from general formulas to sentences.

Proof (of Theorem 17). We first prove the case of sentences. Let L be a language definable by an Arb-invariant FO(*Succ*) sentence with alternation depth d. Let $c > c' > d + 2$. By Lemma 4.8 L is closed under $(\log n)^{c'}$-swaps. Consider now a pair of strings w, w' such that $|w| = |w'| = n$ for a sufficiently large n. Assume that $w \equiv_{(\log n)^c} w'$. By Lemma 4.7 there is a sequence of $((\log n)^c - 1)$-swaps that turns w into w'. For large enough n, $(\log n)^c - 1 \geq (\log n)^{c'}$, and therefore L is closed under such swaps. For this sufficiently large input length none of these swaps affect membership in L, hence $w \in L$ iff $w' \in L$ and the theorem is proved for the case of sentences.

For the general case, we can mark the free variables by new unary predicates, one per free variable. To the initial formula we can associate a sentence quantifying existentially

over these elements and then evaluating the initial formula. For any r, the initial query is Hanf r-local if its associated sentence is also Hanf r-local. Also, if the initial query is Arb-invariant then so is its associated sentence. We may assume that the quantifier depth of the initial formula is at least one; otherwise, the formula is trivially 0-local (see the proof of Theorem 16). Observe that a formula is Hanf r-local and Arb-invariant iff the negation of the formula has the same property. This allows us to further assume without loss of generality that the initial formula begins with an existential quantifier (otherwise, we take the negation), and hence the resulting sentence has the same alternation depth as the initial formula. Theorem 17 follows from case of sentences we have just proved. ∎

4.6.4 Lower Bound

We use a similar idea as in the proof of the lower bound of Theorem 3 to show the Hanf locality lower bound for Arb-invariant FO(*Succ*) claimed in Theorem 4.

Proof (of Theorem 4 - Lower Bound). Fix a constant $c > 0$ and consider the alphabet $A := \{a, b, e, f\}$. We will construct an Arb-invariant sentence ϕ_c that is satisfied by exactly those strings w of the form $(a|b|e)^*f^*$, where (i) the total number of a's, b's, and e's in w is less than $(\log n)^{c+1}$ (where n denotes the length of w), and (ii) the first occurrence of a in w is somewhere to the left of the first occurrence of a b in w.

To note that ϕ_c is *not* Hanf $(\log n)^c$-local, consider, for a sufficiently large n, the string w of length n that, for $m := 2(\log n)^c + 1$, is of the form $e^m a e^m b e^m f^*$. Furthermore, let w' be the string obtained from w by swapping the letters a and b. Note that the strings w and w' are chosen in such a way that

$$w \equiv_{(\log n)^c} w'.$$

Furthermore, w and w' are both of the form $(a|b|e)^*f^*$ and satisfy the following: Since n is sufficiently large, the total number of a's, b's, and e's in w as well as in w' is less

than $(\log n)^{c+1}$. In w, the first occurrence of the letter a is somewhere to the left of the first occurrence of the letter b; however, in w' this is not the case. In summary, we thus obtain that $w \models \phi_c$ and $w' \not\models \phi_c$. Hence, the strings w and w' witness that ϕ_c is not Hanf $(\log n)^c$-local.

To conclude the proof it remains to show how to construct the formula ϕ_c. For this, we use the Arb-invariant formula $reach_d(x, y)$ from Lemma 4.5 for $d := c + 1$. Let $\varrho(x, y)$ be the formula obtained from $reach_d(x, y)$ by replacing every occurrence of an atomic formula of the form $S(z)$ by a formula stating that position z carries one of the letters a, b, or e. Choosing ϕ_c to be the sentence stating that there exist positions x and y that carry the letters a and b, respectively, such that $\varrho(x, y)$ is satisfied, we obtain an Arb-invariant sentence that, when evaluated in a string w of the form $(a|b|e)^* f^*$, states that (i) the total number of a's, b's, and e's in w is less than $(\log n)^{c+1}$ (where n denotes the length of w), and (ii) the first occurrence of a in w is somewhere to the left of the first occurrence of a b in w. This concludes the proof for the lower bound part of Theorem 4. ■

We point out an alternate route for proving the lower bound in Theorem 4, namely by establishing the lower bound in Theorem 3 for strings. The former follows from the latter because if a formula is Hanf r-local w.r.t. (M, M) then it is Gaifman $(3r + 1)$-local w.r.t. M (recall Definitions 4.1 & 4.2) [HLN99]. This alternate route yields an Arb-invariant *formula*, rather than *sentence*, that is not Hanf $(\log n)^c$-local for a given constant c, but the formula can be transformed into a sentence using the translation given in the proof of Theorem 17. We did not follow this route because establishing the lower bound in Theorem 3 for strings would require the schema to have both a unary and a binary predicate, whereas our proof of the lower bound in Theorem 3 only needs the minimal requirement of a binary predicate.

4.7 Implications for Regular Languages

In this section we show that the locality results we proved in the previous sections have nice consequences for regular languages. It is shown in [BS09] that each order-invariant sentence of FO($Succ$) has an equivalent FO(τ_s) formula over strings. In other words, over strings, a linear order used in an order-invariant way does not bring any new expressive power. This is no longer the case when arithmetic is allowed. For instance, in the presence of addition the logic can express parity of the length of a string. To see this, consider the following Arb-invariant sentence which expresses that the string has even length.

$$\exists x, y \quad y = x + x \ \wedge \ \neg(\exists z \ E(y, z))$$

If addition is the only numerical predicate allowed, then it is shown in [SS10b] that addition-invariant FO($Succ$) definable regular languages are exactly those expressible in FO(τ_s, lm), where lm is the family of predicates testing the length of a string modulo some fixed number. We now show that adding any other numerical predicate does not allow new regular languages to be defined, i.e., we prove Theorem 5.

In order to do so, we make use of an equivalent characterization of definability of regular languages in FO(τ_s, lm) in terms of closure under certain operations. We first introduce those operations, which are themselves based on the notion of idempotence for a regular language.

Let L be a regular language, a string e of A^* is said to be *idempotent* (for L, but we will omit L when it is understood from the context) if it is not the empty string and for all $u, v \in A^*, uev \in L$ iff $ueev \in L$. Let $\omega \in \mathbb{N}$ be the smallest positive integer such that for all $u \in A^+$, u^ω is idempotent. Note that ω is well-defined.

A regular language L is *closed under swaps* if for all $x, u, y, v, z, e, f \in A^*$ such that e, f

are idempotent we have:

$$xeufyevfz \in L \quad \text{iff} \quad xevfyeufz \in L. \tag{4.24}$$

A regular language L is *closed under transfers* if for all $x, u, y, v, z \in A^*$ such that $|u| = |v|$ we have:

$$xu^\omega uyv^\omega z \in L \quad \text{iff} \quad xu^\omega yvv^\omega z \in L. \tag{4.25}$$

The following result was shown in [SS10b].

Lemma 4.9 ([SS10b]). *Let L be a regular language. Then L is definable in* $\mathrm{FO}(\tau_s, lm)$ *iff L is closed under transfers and under swaps.*

Lemma 4.9 allows us to prove Theorem 5 by arguing that the regular languages definable in Arb-invariant FO(*Succ*) are closed under transfers and swaps. In Section 4.7.1 we consider a generalization of the notion of closure under transfers for an arbitrary language L, namely closure under r-transfers, where $r : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is a function. We prove that Arb-invariant FO(*Succ*) is closed under $(\log n)^{O(1)}$-transfers. In Section 4.7.2 we use a pumping argument to show that for a regular language L, closure under r-transfers for any function r implies closure under transfers. Using a similar pumping argument, we show that for regular languages, Hanf locality implies closure under swaps. The upper bound in Theorem 4 then concludes the proof of Theorem 5.

4.7.1 Closure under Transfers

Let $r \in \mathbb{N}$ and $w \in A^*$. A string $w' \in A^*$ is obtained from w by an *r-transfer operation* if $w = xu^r uyv^r z$ and $w' = xu^r yv^r vz$ for some $x, u, y, v, z \in A^*$ with $|u| = |v| \neq 0$.

Let $r : \mathbb{N} \to \mathbb{R}_{\geq 0}$. A language L is said to be *closed under $r(n)$-transfers* if there exists a $n_0 \in \mathbb{N}$ such that for all strings $w, w' \in A^*$, with $|w| = n > n_0$, if w' is obtained from w by

a $r(n)$-transfer operation then we have:

$$w \in L \qquad \text{iff} \qquad w' \in L.$$

We begin by proving a lemma similar to Lemma 4.2, but specialized to the task at hand. We show that an Arb-invariant FO(*Succ*) sentence that can distinguish strings separated by a transfer can be used to solve the hard promise problem from Lemma 2.1.

Lemma 4.10. *Let $m \in \mathbb{N}$. Let x, u, y, v, z be strings over A with $|u| = |v|$. Let $w \doteq xu^{3m+1}yv^{3m+1}z$ and $w' \doteq xu^{3m}yv^{3m+2}z$. Suppose C is a circuit that accepts all strings in* $\mathrm{Rep}(w)$ *and rejects all strings in* $\mathrm{Rep}(w')$*, then there is a circuit $\tilde{C}$ with the same size and depth as C that distinguishes $|b|_1 = m$ and $|b|_1 = m + 1$ for $b \in \{0,1\}^{2m}$.*

Proof. Let $b \doteq b_1 b_2 \ldots b_{2m}$ be a string of $2m$ Boolean variables. We design a string w_b that is easy to compute from b and has the following property:

$$\text{If } |b|_1 = m, \text{ then } w_b \cong w.$$
$$\text{If } |b|_1 = m+1, \text{ then } w_b \cong w'.$$

Once we have such a string w_b, we argue as follows. Consider any binary encoding $\Gamma_b \in \mathrm{Rep}(w_b)$. When $|b|_1 = m$, $C(\Gamma_b)$ accepts, because $w_b \cong w$. Similarly, when $|b|_1 = m+1$, $C(\Gamma_b)$ rejects, because $w_b \cong w'$. Thus, $\tilde{C}(b) \doteq C(\Gamma_b)$ yields the required circuit provided Γ_b is sufficiently easy to compute from b. We construct the string w_b and a representation $\Gamma_b \in \mathrm{Rep}(w_b)$ as follows. Let

$$w_b = xu^{4m+1-|b|_1}yv^{2m+1+|b|_1}z. \tag{4.26}$$

We construct w_b as in Figure 4.13.

Add the strings x, y and z to w_b (this includes the vertices, internal edges and labels). Construct $2m + 1$ copies of the strings u and v and add them to w_b. Call these copies

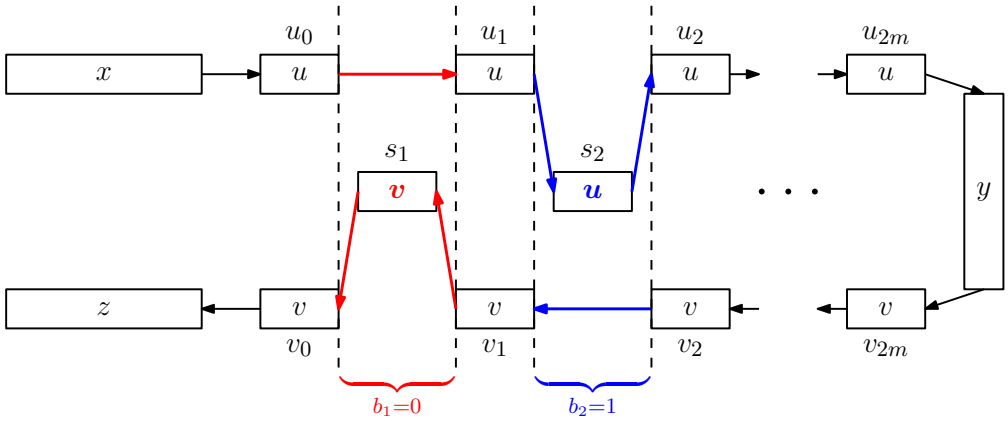


Figure 4.13: Constructing the string w_b from b. (Note that the strings v and z are drawn right to left.)

$u_0, u_1, ..., u_{2m}$, and $v_0, v_1, ..., v_{2m}$, respectively. Add edges from x to u_0, u_{2m} to y, y to v_{2m}, and v_0 to z, i.e., connect the vertex at the end of the former string to the vertex at the beginning of the latter. Construct $2m$ strings of length $|u| = |v|$ with no labels and add them to w_b. Call these strings $s_1, ..., s_{2m}$. Observe that w_b contains exactly $|w| = |w'|$ vertices thus far, though w_b itself is not yet a string because not all edges or labels have been set.

For each $i \in [2m]$:

1. If $b_i = 0$, connect v_i to s_i, s_i to v_{i-1}, and u_{i-1} to u_i, and label s_i as v.

2. If $b_i = 1$, connect u_{i-1} to s_i, s_i to u_i and v_i to v_{i-1}, and label s_i as u.

Note that the w_b constructed is a string of length $|w| = |w'|$ and can be written as in (4.26). Since the presence of each edge and the value of each label depends on at most one bit of b, the string w_b can be encoded in binary $\Gamma_b \in \mathrm{Rep}(w_b)$ so that each bit of the encoding is either a constant or a literal from b (see the proof of Lemma 4.2 for more details). This allows us to define the circuit $\tilde{C}(b) \doteq C(\Gamma_b)$ which has depth and size no larger than C, and completes the proof. ∎

With this lemma we can prove that Arb-invariant FO(*Succ*) is closed under $(\log n)^{O(1)}$-transfers following the same approach as in the proof of Theorem 3.

Lemma 4.11. *If L is a language definable in Arb-invariant* FO(*Succ*) *then there is a $c \in \mathbb{N}$ such that L is closed under $(\log n)^c$-transfers.*

Proof. Immediate from Lemma 4.10 and Lemma 2.1 via the same type of manipulation that proves Theorem 3. ∎

4.7.2 Definability under Arb-Invariance

We now use Lemma 4.11 to show that a regular language is definable in Arb-invariant FO(*Succ*) iff it is definable in FO(*Succ*, *lm*). The "only if" direction is straightforward as any predicate of the form *lm* can be expressed in Arb-invariant FO(*Succ*) (actually only addition is needed). For the "if" direction we show that a pumping argument combined with Hanf $(\log n)^{O(1)}$-locality and $(\log n)^{O(1)}$-transfers implies closure under swaps and transfers, and we can conclude using Lemma 4.9.

Proof (of Theorem 5). As L is definable in Arb-invariant FO(*Succ*), by Theorem 17 there is a constant $c \in \mathbb{N}$ such that L is Hanf $(\log n)^c$-local. Hence there is a constant n_0 such that for all strings w, w' with $|w| = |w'| > n_0$, $w \equiv_{(\log n)^c} w'$ implies $w \in L$ iff $w' \in L$.

Consider $x, u, y, v, z, e,$ and f as in the hypothesis for closure under swaps. As e and f are idempotents we have for all $i \in \mathbb{N}$, $xe^iuf^iye^ivf^iz \in L$ iff $xeufyevfz \in L$. Take i large enough so that $i \cdot |e|, i \cdot |f| \geq (\log |xe^{2i}uf^{2i}ye^{2i}vf^{2i}z|)^c \geq (\log n_0)^c$. Notice that $xe^{2i}uf^{2i}ye^{2i}vf^{2i}z \equiv_{(\log n)^c} xe^{2i}vf^{2i}ye^{2i}uf^{2i}z$ via the bijection sending respectively xe^i, e^iyf^i, e^iuf^i, e^ivf^i and f^iz to their corresponding substring in the other string. Hence by Theorem 17 we have $xe^{2i}uf^{2i}ye^{2i}vf^{2i}z \in L$ iff $xe^{2i}vf^{2i}ye^{2i}uf^{2i}z \in L$. But again, as e and f are idempotents, this implies $xeufyevfz \in L$ iff $xeufyevfz \in L$. Therefore L is closed under swaps.

The pumping argument for closure under transfers is identical. By Lemma 4.11 there are constants c and n_0 such that L is closed under $(\log n)^c$-transfers for strings of length bigger than n_0.

Consider x, u, y, v, z as in the hypothesis for closure under transfers. As u^ω and v^ω are idempotents we have for all $i \in \mathbb{N}$, $xu^{\omega \cdot i}uyv^{\omega \cdot i}z \in L$ iff $xu^\omega uyv^\omega z \in L$. Take i large enough so that $i \cdot \omega \geq (\log |xu^{\omega \cdot i}uyv^{\omega \cdot i}z|)^c \geq (\log n_0)^c$. Hence we can apply closure under $(\log n)^c$-transfers and by Lemma 4.11 we have $xu^{\omega \cdot i}uyv^{\omega \cdot i}z \in L$ iff $xu^{\omega \cdot i}yvv^{\omega \cdot i}z \in L$. But again, as u^ω and v^ω are idempotents, this implies $xu^\omega uyv^\omega z \in L$ iff $xu^\omega yvv^\omega z \in L$. Therefore L is closed under transfers.

This concludes the proof of Theorem 5. ∎

4.8 Further Research

We end with a few suggestions for further research.

We have established the precise level of locality of Arb-invariant FO formulas for the Gaifman notion of locality. As pointed out in [Gro09]: "It would be interesting to see a small complexity class like uniform AC^0 [...] captured by a logic." This remains an open problem – recall that although Arb-invariant FO does capture AC^0, it does not have an effective syntax. Note that *over regular languages*, we do have an effective syntax as we have shown that Arb-invariant FO(*Succ*) has exactly the same expressive power as $FO(\tau_s, lm)$.

Another open question is whether our upper bound for Gaifman locality also holds for Hanf locality on arbitrary structures. We have established this bound for string structures only. We believe that a similar argument should also work for tree structures.

5 On Sets that Realize Lines

Geometry is the science of correct reasoning on incorrect figures.

— George Polya

In this chapter we give tight bounds on the size of set systems that realize, through pair-wise intersection, all lines in $\mathrm{GF}(p)^2$. First, in Section 5.1 we give a formal description of the motivation and definitions. Then, in Section 5.2 we describe a set family $\mathcal{F}$ of size $O(p^{3/2})$ that is sufficient to realize all lines, and in Section 5.3 we conclude by showing that a family of this size is optimal.

5.1 Background

Recall that [DvM10] shows for vertex cover instances on n vertices that *no* deterministic polynomial-time mapping reduction exists to instances of size $O(n^c)$ (for any language L), for any $c < 2$, unless the polynomial hierarchy collapses. Our result, which we present later in this chapter, applies to an earlier weaker version of their theorem that rules out reduced instances only for $c < \frac{4}{3}$ [DvM09].

To give the formal motivation behind our result will not describe the full proof of [DvM10], but instead explain the aspects of it related to the problem we consider here. To this end we introduce the parameterized problem, k-CLIQUE, the problem of deciding whether a given graph G on n vertices has a subset S of the vertices of size k, such that every pair of vertices in S are connected by an edge.

Consider t graphs $G_1, G_2, \ldots, G_t$ each on s vertices. We say a graph G is a *packing* of the t graphs if for all $m \leq s$: G has a clique of size $m + 2$ iff at least one of the original t graphs is a clique of size m. Dell and Van Melkebeek implicitly prove the following lemma.

Lemma 5.1 ([DvM10, Implicit in proof of Theorem 2]). *Let be t be a polynomially bounded function of s. Suppose*

1. *there is a way to pack t clique instances each on s vertices into a single clique instance G with $|G|$ vertices, and*
2. *there is a constant c and a polynomial-time mapping reduction that takes clique instances on n vertices to instances of bit-length $O(n^c)$ of some language L.*

If $|G|^c = O(t \log t)$, then NP $\subseteq$ coNP/ poly *and the polynomial hierarchy collapses.*

Informally, the lemma says that if there is an efficient way of packing clique instances into a single instance, it rules out map reducing clique instances to instances of small bit-length unless the polynomial hierarchy collapses. Observe that the smaller the size of the packed instance G is as a function of t the larger the range of constants c that are ruled out unless the polynomial hierarchy collapses. Thus, our goal becomes showing how to pack clique instances into a graph that has as few vertices as possible.

Let $\mathbb{F} \doteq \mathrm{GF}(p)$ be a finite field. We now describe a possible construction of the packed graph G. Consider an s-partite graph P with p vertices per partition class. Note that P has size $s \cdot p$ and we can view the vertices of P as $(x, y) \in \{0, 1, \ldots, s-1\} \times \mathbb{F} \subseteq \mathbb{F}^2$ for this we need that $p \geq s$.

Definition 5.1 (Lines, $\mathcal{L}$). *Let $L_{a,b} \doteq \{(x, a + bx) \mid x \in \mathbb{F}\}$ be a* line *in $\mathbb{F}^2$. For ease of notation we often refer to the line $L_{a,b}$, for a tuple $\ell \doteq (a, b)$, as L_ℓ or simply L. Let $\mathcal{L} \doteq \{L_{a,b} \mid a, b \in \mathbb{F}\}$.*

The lines $\mathcal{L}$ naturally correspond to distinct subsets $P \cap L \subseteq P$ of size s. Moreover, $|P \cap L \cap L'| \leq 1$ for distinct lines L and L'. Uniquely assign each graph G_i to some line $L_i \in \mathcal{L}$; for this to be possible the number of lines, $p^2 = |\mathbb{F}|^2$, must be a least t, the number

of graphs. Then, for each G_i, connect the s vertices in $P \cap L_i$ as they are in G_i, i.e., so that the induced subgraph on $P \cap L_i$ is isomorphic to G_i.

Definition 5.2 (Realization). *Let $2^{\mathbb{F}^2}$ be the powerset of $\mathbb{F}^2$.*

- *A set $L \subseteq \mathbb{F}^2$ is* realized *by a pair of sets $S, Q \subseteq \mathbb{F}^2$ if $S \cap Q = L$.*
- *A family of sets $\mathcal{F} \subseteq 2^{\mathbb{F}^2}$,* realizes *a set $L \subseteq \mathbb{F}^2$ if there exists $S, Q \in \mathcal{F}$ that realize L.*
- *A family of sets $\mathcal{F} \subseteq 2^{\mathbb{F}^2}$,* realizes *a family of sets $\mathcal{L} \subseteq 2^{\mathbb{F}^2}$ if for all $L \in \mathcal{L}$, $\mathcal{F}$ realizes L.*

Let $\mathcal{F}$ be a set family that realizes $\mathcal{L}$. We form G by adding new vertices to P. Add the vertices u_s and v_s for each set S in $\mathcal{F}$, and connect u_S to all the vertices in $P \cap S$, and do the same for v_s. Now, connect the vertices u_Q and v_S with an edge iff the set $Q \cap S$ is a line in $\mathcal{L}$. It follows that the resulting graph G has a clique of size $s+2$ iff at least one of the G_i's is a clique of size s. The graph G has $s \cdot p + 2 \cdot |\mathcal{F}|$ vertices.

Suppose we have a set family $\mathcal{F}$ of size $p^{3/2}$. Lemma 5.1 implies that map reducing G to an instance of size $|G|^c$ for any $c < 4/3$ can be ruled out unless the hierarchy collapses. To see this, pick $p \sim \sqrt{t}$ and t to be a sufficiently large polynomial in s, then $|G|^c \leq (s\sqrt{t} + 2t^{\frac{3}{4}})^c \leq (3t^{\frac{3}{4}})^c$. Therefore, $|G|^c = O(t \log t)$ for any $c < 4/3$.

The result of this chapter shows that for this type of set system $c < \frac{4}{3}$ is optimal.

5.2 Upper Bound

In this section we assume that p is prime; this is not an inherent limitation of our proof, rather it is for ease of notation.

First, we consider the following simplification of the problem to the horizontal lines $L_a \doteq \{(x, a) \mid x \in \mathbb{F}\}$, for $a \in \mathbb{F}$.

Lemma 5.2. *There is a set family $\mathcal{F}$ of size $O(\sqrt{p})$ such that for each $a \in \mathbb{F}$, $\mathcal{F}$ realizes L_a.*

Proof. Let $r \doteq \lceil \sqrt{p} \rceil$. For $x \in \mathbb{F}$ and integers $0 \leq i, j < r$, let

$$Q_{i,x} \doteq \{(x, i + 0 \cdot r), (x, i + 1 \cdot r), \ldots, (x, i + (r-1) \cdot r)\}, \text{ and}$$
$$S_{j,x} \doteq \{(x, 0 + j \cdot r), (x, 1 + j \cdot r), \ldots, (x, (r-1) + j \cdot r)\},$$

where addition is over the integers and when the second element of a tuple is greater or equal to p we drop that tuple. Further define $Q_i \doteq \cup_{x \in \mathbb{F}} Q_{i,x}$ and $S_j \doteq \cup_{x \in \mathbb{F}} S_{j,x}$. For each $a \in \mathbb{F}$, since $r^2 > |\mathbb{F}|$ we can write $a = i + j \cdot r$ with $i, j \in \{0, 1, \ldots, r-1\}$. Then

$$L_a = Q_i \cap S_j = \{(x, \overbrace{i + j \cdot r}^{a}) \mid x \in \mathbb{F}\}.$$

Thus $\mathcal{F} \doteq \{Q_i\}_i \cup \{S_j\}_j$ realizes all horizontal lines. We conclude by noticing $|\mathcal{F}| = 2r = O(\sqrt{p})$. ∎

Observe that this bound is tight, since at least $\sqrt{p}$ sets are required to represent p distinct sets via pairs of sets. From now on let

$$\mathcal{L} \doteq \{L_{a,b} \mid a, b \in \mathbb{F}\}$$

be the set of all lines. We easily generalize the upper bound on horizontal lines to all of $\mathcal{L}$.

Lemma 5.3. *There exists a set family $\mathcal{F}$ of size $O(p^{3/2})$ that realizes $\mathcal{L}$.*

Proof. The proof is similar to that of Lemma 5.2. Let $r \doteq \lceil \sqrt{p} \rceil$. For $b, x \in \mathbb{F}$ and integers

$0 \leq i, j < r$, let

$$Q^b_{i,x} \doteq \{(x, i + 0 \cdot r + bx), (x, i + 1 \cdot r + bx), \ldots, (x, i + (r-1) \cdot r + bx)\}, \text{ and}$$
$$S^b_{j,x} \doteq \{(x, 0 + j \cdot r + bx), (x, 1 + j \cdot r + bx), \ldots, (x, (r-1) + j \cdot r + bx)\},$$

where addition is over the integers and when the second element of a tuple is greater or equal to p we drop that tuple.

Similarly, define $Q^b_i \doteq \cup_{x \in \mathbb{F}} Q^b_{i,x}$ and $S^b_j \doteq \cup_{x \in \mathbb{F}} S^b_{j,x}$, and let $\mathcal{F} \doteq \{Q^b_i\}_{i,b} \cup \{S^b_j\}_{j,b}$. We can establish correctness: For every $a, b \in \mathbb{F}$, with $a = i + j \cdot r$ and $i, j \in \{0, 1, \ldots, r-1\}$,

$$Q^b_i \cap S^b_j = \{(x, \overbrace{i + j \cdot r}^{a} + bx) \mid x \in \mathbb{F}\} = L_{a,b}.$$

We conclude by observing $|\mathcal{F}| = 2rp = O(p^{3/2})$. ∎

5.3 Lower Bound

In this subsection we argue that size $\Omega(p^{3/2})$ is necessary to realize $\mathcal{L}$. Recall some standard facts about lines in $\mathbb{F}^2$ that follow easily from their definition:

- Two lines that are *parallel* and distinct do not intersect.
- Two lines that are *skew* must intersect at exactly one point.
- Two lines that have two distinct points in common are *identical*.

Observe that these three facts essentially use the property that fields are domains, that is, for $a, b \in \mathbb{F}$, if $ab = 0$ then $a = 0$ or $b = 0$.

Definition 5.3 (Covering). *For $S \subseteq \mathbb{F}^2$. Say S* covers *a line, $L_{a,b}$ $a, b \in \mathbb{F}^2$, if $S \supseteq L_{a,b}$.*

With these facts and definitions we can prove the following lemma that, intuitively, says that if there are two sets Q and S that cover several lines and realize some line in common, then the sets each must have similar structure.

Lemma 5.4. *Let the sets Q and S each cover at least four distinct lines, and Q and S realize L (i.e., $Q \cap S = L$). S has exactly one of the following types:*

1. *All lines covered by S are parallel.*
2. *All lines covered by S are parallel, except for L, which is skew with respect to the others.*
3. *There exists exactly one point in $\mathbb{F}^2$, which every pair of lines that S covers intersect at.*

Moreover, Q and S have the same type.

Proof. Since the sets Q and S cover at least four lines each, the three types in the statement of the lemma are inherently distinct. Since S and Q realize L, S and Q cannot cover any other lines in common. We proceed by case analysis on the lines that S covers.

Type 1. Suppose S covers a distinct line s that is parallel to L. Then Q cannot cover any line q skew to s because the intersection point between s and q will not be on L, contradicting the fact that $Q \cap S = L$. Therefore the other lines that Q covers must all be parallel to s (and L). Hence Q covers a distinct line q that is parallel to L. This argument can be repeated starting with q to conclude that all the lines covered by S are also parallel. This is exactly type (1). See Figure 5.1.(a).

Therefore, we can assume that all the lines covered by S and Q are now skew to L.

Type 2. Suppose S covers distinct lines s_1 and s_2 that are parallel to each other and skew to L. Further suppose, for the sake of contradiction, that Q covers a line q that is distinct

from L and not parallel to s_1 and s_2. Then q intersects both s_1 and s_2. Since q is distinct from L at least one of the intersection points must not be on L, this contradicts the fact that $Q \cap S = L$. Therefore we can conclude that all the other lines that Q covers must be parallel to s_1 and s_2. See Figure 5.1.(b). Since Q covers at least three lines we can repeat this argument starting from Q to get that all the lines that S covers other than L must also be parallel. This is type (2).

Therefore, we can further assume that all the lines covered by S (and Q) are pairwise skew.

Type 3. Suppose S covers lines s_1 and s_2, which are skew to each other and to L. Suppose all intersections between L, s_1 and s_2 occur at a single point. Then all lines covered by Q go through the same intersection point and are distinct from the other lines of S, otherwise they will intersect either s_1 or s_2 at a point not on L, which contradictions $Q \cap S = L$. This is type (3). See Figure 5.1.(c).

Otherwise, the intersections between L, s_1, and s_2 occur at three distinct points. It can be shown by inspection that Q can only cover 3 lines: L, the line through the intersection point of L and s_1 that is parallel to s_2, and the line through the intersection point of L and s_2 that is parallel to s_1. This is contradicts the fact that S and Q cover at least four lines. See Figure 5.1.(d).

We have argued that S must have exactly one of the types and Q must match it.

One consequence of Lemma 5.4 is that sets that cover several lines and have different types cannot be used together to realize lines. Intuitively this partitions the sets into equivalence classes. This allows us to show that the number of sets required to realize all of $\mathcal{L}$ must be large.

Lemma 5.5. *Any family $\mathcal{F}$ of sets that realizes $\mathcal{L}$ must be of size $\Omega(p^{3/2})$.*

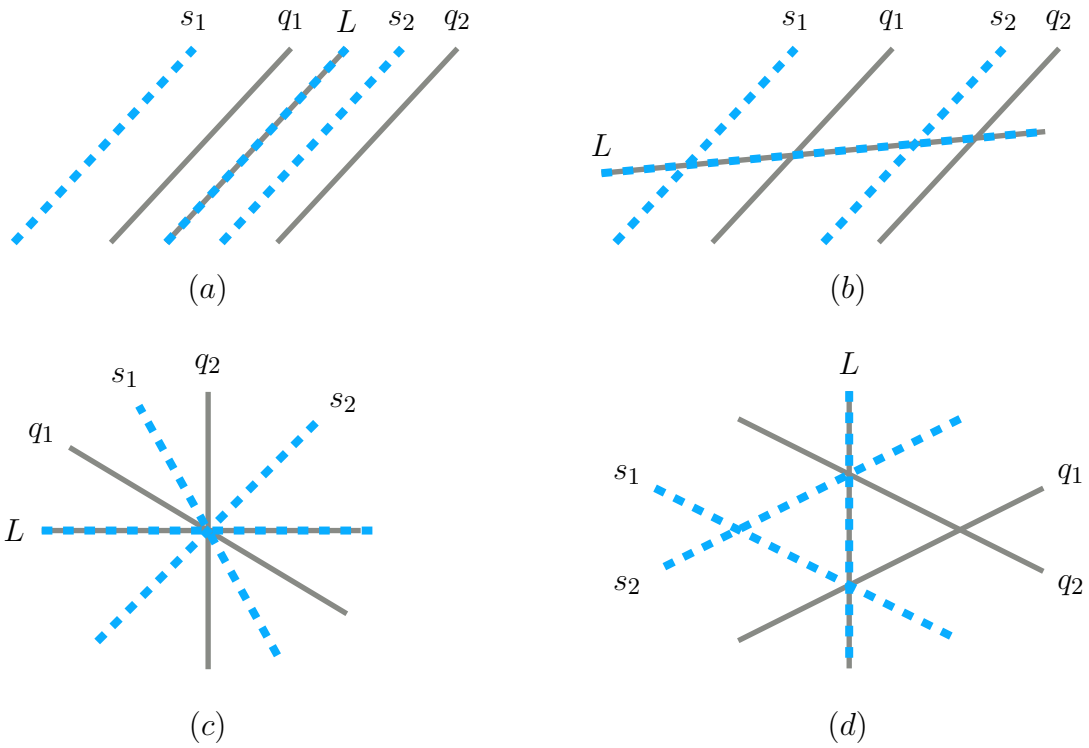


Figure 5.1: Cases of Lemma 5.4.

Proof. Consider a family $\mathcal{F}$ of sets. We count the number of lines in $\mathcal{L}$ it can realize.

Consider the number of lines realized by pairs of sets in $\mathcal{F}$ where one of the sets covers at most 3 lines. This quantity is at most $3|\mathcal{F}|$. The remaining lines that are realized by $\mathcal{F}$ must be realized by pairs of sets that each cover at least four lines.

Since we are now only dealing with pairs of sets covering at least four lines we can apply Lemma 5.4. Recall that this lemma partitions all sets that cover at least four lines into three disjoint types. Furthermore, the result implies that if $S, Q \in \mathcal{F}$ realize a line L they must have the same type.

We now observe that the sets of $\mathcal{F}$ can be partitioned into disjoint families, and, moreover, we note that the number of lines realized by the each individual family is at most p.

Type 1. For a slope $b \in \mathbb{F}$ let $\mathcal{F}_b^{(1)}$ be the sets in $\mathcal{F}$ that have type (1) where all the realized sets in have slope b. Then $\mathcal{F}_b^{(1)}$ can only realize the $|\mathbb{F}| = p$ lines of $\mathcal{L}$ with slope b.

Type 2. For a slope $b \in \mathbb{F}$, and line L that does not have slope b, let $\mathcal{F}^{(2)}_{L,b}$ be the sets in $\mathcal{F}$ that have type (2) with skew line L and parallel line slope b. The only line realizable within $\mathcal{F}^{(2)}_{L,b}$ is exactly L.

Type 3. For a point $(\alpha, \beta) \in \mathbb{F}^2$, let $\mathcal{F}^{(3)}_{(\alpha,\beta)}$ be the sets in $\mathcal{F}$ that have type (3) where all the sets have common intersection point (α, β). Then $\mathcal{F}^{(3)}_{(\alpha,\beta)}$ can only realize the $|\mathbb{F}| = p$ lines of $\mathcal{L}$ that go through (α, β).

By this analysis the large sets in $\mathcal{F}$ can be partitioned into disjoint families:

$$\mathcal{F} \supseteq (\sqcup_b \mathcal{F}^{(1)}_b) \sqcup (\sqcup_{L,b} \mathcal{F}^{(2)}_{L,b}) \sqcup (\sqcup_{\alpha,\beta} \mathcal{F}^{(3)}_{\alpha,\beta}).$$

We argued above that each family can realize at most p lines in $\mathcal{L}$. Therefore, the number of lines realized by any of one of these disjoint families $\mathcal{F}_i$ is at most $\min(|\mathcal{F}_i|^2, p)$. The best a family can do is realize p lines using $\sqrt{p}$ sets. Assuming that every family realizes independent lines, which is an over-estimate, the strategy to realize the largest number of lines is to maximize the number of families contributing the maximum number of realizations. Therefore we can assign $\frac{|\mathcal{F}|}{\sqrt{p}}$ families the responsibility of each covering p lines using $\sqrt{p}$ sets. This realizes at most $\frac{|\mathcal{F}|}{\sqrt{p}} \cdot p = |\mathcal{F}|\sqrt{p}$ lines in $\mathcal{L}$.

Combining this with the number of lines realized by small sets and using the fact that $\mathcal{F}$ realizes all of $\mathcal{L}$ we have:

$$3|\mathcal{F}| + |\mathcal{F}|\sqrt{p} \geq p^2.$$

We conclude that $|\mathcal{F}| = \Omega(p^{3/2})$. ∎

This immediately implies the main theorem for this section: A tight bound on the size of set families realizing $\mathcal{L}$.

Theorem 18 (Theorem 6 – restated).

Let $\mathbb{F} \doteq \mathrm{GF}(p)$*, for* $a, b \in \mathbb{F}$ *let* $L_{a,b} \doteq \{(x, a + bx) \mid x \in \mathbb{F}\}$*, and let* $\mathcal{L} \doteq \cup_{a,b\in\mathbb{F}} L_{a,b}$*. Then the number of sets sufficient and necessary to realize* $\mathcal{L}$ *is* $\Theta(p^{1.5})$*.*

Proof. Combine Lemma 5.3 and Lemma 5.5. ∎

Bibliography

[AB87] N. Alon and R. B. Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7(1):1–22, 1987.

[AB03] M. Agrawal and S. Biswas. Primality and identity testing via Chinese remaindering. *Journal of the ACM (JACM)*, 50(4):429–443, 2003.

[Agr03] M. Agrawal. On derandomizing tests for certain polynomial identities. In *Proceedings of the IEEE Conference on Computational Complexity (CCC)*, pages 355–359, 2003.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[Ajt83] M. Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):1–48, 1983.

[Ajt89] M. Ajtai. First-order definability on finite structures. *Annals of Pure and Applied Logic*, 45(3):211–225, 1989.

[AM10] V. Arvind and P. Mukhopadhyay. The ideal membership problem and polynomial identity testing. *Information and Computation*, 208(4):351–363, 2010.

[And85] A. E. Andreev. On a method for obtaining lower bounds for the complexity of individual monotone functions. *Doklady Akademii Nauk*, 31(3):1033–1037, 1985. Translation in *Dokl. Math.* 31:350-354.

[ASSS11] M. Agrawal, C. Saha, R. Saptharishi, and N. Saxena. Jacobian hits circuits: Hitting-sets, lower bounds for depth-d occur-k formulas & depth-3 transcendence degree-k circuits. Technical Report 143, Electronic Colloquium on Computational Complexity (ECCC), 2011.

[AV08] M. Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 67–75, 2008.

[AvM10] S. Aaronson and D. van Melkebeek. A note on circuit lower bounds from derandomization. Technical Report 105, Electronic Colloquium on Computational Complexity (ECCC), 2010.

[AvMSS11] M. Anderson, D. van Melkebeek, N. Schweikardt, and L. Segoufin. Locality of queries definable in invariant first-order logic with arbitrary built-in predicates. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 368–379, 2011.

[AvMV11] M. Anderson, D. van Melkebeek, and I. Volkovich. Derandomizing polynomial identity testing for multilinear constant-read formulae. In *Proceedings of the IEEE Conference on Computational Complexity (CCC)*, pages 273–282, 2011.

[AW09] S. Aaronson and A. Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory (TOCT)*, 1(1):2, 2009.

[Bau00] L. F. Baum. *The Wonderful Wizard of Oz.* G. M. Hill, 1900.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM Journal on Computing (SICOMP)*, 4:431, 1975.

[BHLV09] M. Bläser, M. Hardt, R. Lipton, and N. Vishnoi. Deterministically testing sparse polynomial identities of unbounded degree. *Information Processing Letters (IPL)*, 109(3):187–192, 2009.

[BOT88] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, pages 301–309, 1988.

[BP89] D. Beauquier and J.-É. Pin. Factors of words. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 63–79, 1989.

[BS83] W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science (TCS)*, 22(3):317–330, 1983.

[BS09] M. Benedikt and L. Segoufin. Towards a characterization of order-invariant queries over tame structures. *Journal on Symbolic Logic (JSL)*, 74(1):168–186, 2009.

[BS10] M. Benedikt and L. Segoufin. Regular tree languages definable in FO and FO_{mod}. *ACM Transactions on Computational Logic (TOCL)*, 11(1), 2010.

[Cai89] J-Y. Cai. With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy. *Journal of Computer and System Sciences (JCSS)*, 38(1):68–85, 1989.

[Cai01] J-Y. Cai. $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, page 620. IEEE Computer Society, 2001.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[DL78] R. DeMillo and R. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters (IPL)*, 7(4):193–195, 1978.

[DLM07] A. Durand, C. Lautemann, and M. More. Counting results in weak formalisms. In *Circuits, Logic, and Games*, number 06451 in Dagstuhl Seminar Proceedings, 2007.

[DS07] Z. Dvir and A. Shpilka. Locally decodable codes with two queries and polynomial identity testing for depth 3 circuits. *SIAM Journal on Computing (SICOMP)*, 36(5):1404–1434, 2007.

[DSY08] Z. Dvir, A. Shpilka, and A. Yehudayoff. Hardness-randomness tradeoffs for bounded depth arithmetic circuits. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, pages 741–748. ACM, 2008.

[DvM09] H. Dell and D. van Melkebeek. Private communication, 2009.

[DvM10] H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, pages 251–260. ACM, 2010.

[EF99] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1999.

[FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Theory of Computing Systems*, 17(1):13–27, 1984.

[FSV95] R. Fagin, L. J. Stockmeyer, and M. Y. Vardi. On monadic NP vs. monadic co-NP. *Information and Computation*, 120(1):78–92, 1995.

[Gai82] H. Gaifman. On local and non-local properties. In J. Stern, editor, *Proceedings of the Herbrand Symposium, Logic Colloquium*, pages 105–135, 1982.

[GK11] M. Grohe and S. Kreutzer. Methods for algorithmic meta theorems. In M. Grohe and J. Makowsky, editors, *Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*. American Mathematical Society, 2011.

[GN07] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, March 2007.

[Gro09] M. Grohe. Fixed-point definability and polynomial time. In *Computer Science Logic (CSL)*, pages 20–23, 2009.

[GS00] M. Grohe and T. Schwentick. Locality of order-invariant first-order formulas. *ACM Transactions on Computational Logic (TOCL)*, 1(1):112–130, 2000.

[Han65] W. Hanf. Model-theoretic methods in the study of elementary logic. In J. W. Addison, L. Henkin, and A. Tarski, editors, *The theory of models*, pages 132–145. North-Holland, 1965.

[Hås86] J. Håstad. *Computational Limitations for Small-Depth Circuits*. PhD thesis, MIT, 1986.

[HLN99] L. Hella, L. Libkin, and J. Nurmonen. Notions of locality and their logical characterizations over finite models. *Journal on Symbolic Logic (JSL)*, 64(4):1751–1773, 1999.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1-3):86–104, 1986.

[Imm87] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing (SICOMP)*, 16(4):760–778, 1987.

[Imm99] N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1999.

[Isa09] I. M. Isaacs. *Algebra: A Graduate Course*. Graduate Studies in Mathematics. American Mathematical Society, 2009.

[JQS10] M. J. Jansen, Y. Qiao, and J. M. N. Sarma. Deterministic identity testing of read-once algebraic branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:84, 2010.

[Kan82] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1):40–56, 1982.

[KI04] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity (CC)*, 13(1):1–46, 2004.

[KL82] R. M. Karp and R. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(2):191–209, 1982.

[KMSV10] Z. Karnin, P. Mukhopadhyay, A. Shpilka, and I. Volkovich. Deterministic identity testing of depth-4 multilinear circuits with bounded top fan-in. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, pages 649–658, 2010.

[KS01] A. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, pages 216–223, 2001.

[KS07] N. Kayal and N. Saxena. Polynomial identity testing for depth 3 circuits. *Computational Complexity (CC)*, 16(2):115–138, 2007.

[KS08] Z. Karnin and A. Shpilka. Black box polynomial identity testing of generalized depth-3 arithmetic circuits with bounded top fan-in. In *Proceedings of the IEEE Conference on Computational Complexity (CCC)*, pages 280–291, 2008.

[KS09] N. Kayal and S. Saraf. Blackbox polynomial identity testing for depth 3 circuits. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2009.

[KvMS09] J. Kinne, D. van Melkebeek, and R. Shaltiel. Pseudorandom generators and typically-correct derandomization. In *Proceedings of the International Workshop on Randomization and Computation (RANDOM)*, pages 574–587, 2009.

[Lib04] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[LMN93] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform, and learnability. *Journal of the ACM (JACM)*, 40(3):620, 1993.

[LN00] L. Libkin and J. Nurmonen. Counting and locality over finite structures: A survey. In *Generalized Quantifiers and Computation, European Summer School in Logic, Language, and Information (ESSLLI)*, volume 1754 of *Lecture Notes in Computer Science*, pages 18–50. Springer, 2000.

[Lov79] L. Lovász. On determinants, matchings and random algorithms. In *Fundamentals of Computation Theory*, volume 79, pages 565–574, 1979.

[Mak97] J. A. Makowsky. Invariant definability (extended abstract). In *Computational Logic and Proof Theory, Proceedings of the 5th Kurt Gödel Colloquium (KGC'97)*, volume 1289 of *Lecture Notes in Computer Science*, pages 186–202. Springer, 1997.

[Mak98] J. A. Makowsky. Invariant definability and P/*poly*. In *Computer Science Logic (CSL)*, pages 142–158, 1998.

[Ott00] M. Otto. Epsilon-logic is more expressive than first-order logic over finite structures. *Journal on Symbolic Logic (JSL)*, 65(4):1749–1757, 2000.

[Raz85] A. A. Razborov. Lower bounds on the monotonic complexity of some Boolean functions. *Doklady Akademii Nauk*, 281(4):798–801, 1985. Translation in *Dokl. Math.* 31:354-357.

[Raz09] R. Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *Journal of the ACM (JACM)*, 56(2):8, 2009.

[Ros07] B. Rossman. Successor-invariant first-order logic on finite structures. *Journal on Symbolic Logic (JSL)*, 72(2):601–618, 2007.

[Ros08] B. Rossman. On the constant-depth complexity of k-clique. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, pages 721–730. ACM, 2008.

[Ros10] B. Rossman. *Average-Case Complexity of Detecting Cliques*. PhD thesis, MIT, 2010.

[RR97] A. A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.

[Sax08] N. Saxena. Diagonal circuit identity testing and lower bounds. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 60–71, 2008.

[Sax09] N. Saxena. Progress on polynomial identity testing. *Bulletin of the EATCS*, 99:49–79, 2009.

[Sch80] J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.

[Sch96] T. Schwentick. On winning Ehrenfeucht games and monadic NP. *Annals of Pure and Applied Logic*, 79(1):61–92, 1996.

[SS09] N. Saxena and C. Seshadhri. An almost optimal rank bound for depth-3 identities. In *Proceedings of the IEEE Conference on Computational Complexity (CCC)*, pages 137–148, 2009.

[SS10a] N. Saxena and C. Seshadhri. From Sylvester-Gallai configurations to rank bounds: Improved black-box identity test for depth-3 circuits. Technical Report 13, Electronic Colloquium on Computational Complexity (ECCC), 2010.

[SS10b] N. Schweikardt and L. Segoufin. Addition-invariant FO and regularity. In *Proceedings of the Conference on Logic in Computer Science (LICS)*, pages 273–282, 2010.

[SS11] N. Saxena and C. Seshadhri. Blackbox identity testing for bounded top fanin depth-3 circuits: The field doesn't matter. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, pages 431–440, 2011.

[SV08] A. Shpilka and I. Volkovich. Read-once polynomial identity testing. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, pages 507–516, 2008.

[SV09] A. Shpilka and I. Volkovich. Improved polynomial identity testing for read-once formulas. In *Proceedings of the International Workshop on Randomization and Computation (RANDOM)*, pages 700–713, 2009.

[SV11] S. Saraf and I. Volkovich. Black-box identity testing of depth-4 multilinear circuits. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, 2011.

[SY10] A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science (FTTCS)*, 5(3–4):207–388, 2010.

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing (SICOMP)*, 20:865, 1991.

[TW85] D. Thérien and A. Weiss. Graph congruences and wreath products. *Journal of Pure and Applied Algebra*, 36(205–215), 1985.

[Val79] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science (TCS)*, 8(2):189–201, 1979.

[Var82] M. Vardi. The complexity of relational query languages. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, pages 137–146, 1982.

[Wil11] R. Williams. Non-uniform acc circuit lower bounds. In *Proceedings of the IEEE Conference on Computational Complexity (CCC)*, pages 115–125. IEEE Computer Society, 2011.

[Yao85] A. Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 1–10, 1985.

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. *Symbolic and Algebraic Computation*, pages 216–226, 1979.