Minimal XOR Circuits: The One True Shape is a Binary Tree

Marco Carmosino*[1], Ngu Dang†[2], and Tim Jackman‡[3]

[1]IBM

[2,3]Department of Computer Science, Boston University

February 2024

Abstract

Determining the complexity of circuit minimization is a fundamental and longstanding open problem. To better understand the landscape of minimal circuits, we can ask: given a Boolean function f and complexity measure μ, is there a structural characterization of the μ-optimal set of circuits computing f? For some functions and measures — such as n-bit OR and DeMorgan circuit size — the question is easy. For other functions — such as the n-bit XOR function and DeMorgan circuit size counting NOT gates — such characterization seem to require a deep understanding of circuit lower bounds for XOR proved via gate elimination (Kombarov, 2011).

In this paper, we show that even when NOT gates are free, minimal XOR circuits are "shaped like" binary trees of fan-in 2 XOR sub-circuits. As a corollary, we obtain an efficient algorithm for testing if a given circuit is an optimal XOR circuit. Our argument must carry out the intricate case analysis of gate elimination quite differently from Kombarov's, to handle the additional flexibility afforded by free NOT gates. We formalize the simplification steps used by gate elimination as a graph rewriting system, and demonstrate that our system is "well behaved" via a partially machine-checkable proof; this perspective on gate elimination may be of independent interest.

We are also motivated by the recent result that the Partial Function Circuit Minimization Problem is hard, assuming the Exponential Time Hypothesis (Ilango, 2020). That proof leveraged a characterization of the optimal circuits for n-bit OR and a novel perspective on gate elimination. By characterizing the structure of n-bit XOR, we take a step towards extending the proof of hardness to total MCSP. Crucially, previous results about the structure of optimal XOR circuits are incompatible with Ilango's proof — the implicit complexity measure must ignore NOT gates.

*mlc@ibm.com
†ndang@bu.edu
‡tjackman@bu.edu

1 Introduction

Circuits model the computation of Boolean functions on fixed input lengths by acyclic wires between atomic processing units — logical "gates." To measure the circuit complexity of a function f, we fix a set of gates $\mathcal{B}$ — called a *basis* — and count the number of $\mathcal{B}$-gates required to compute f. This work studies circuits over the DeMorgan basis: fan-in 2 AND, fan-in 2 OR, and fan-in 1 NOT gates. We do not restrict the wiring pattern. Basic questions have been open for decades; we cannot even rule out the possibility that every problem in NP is decided by a sequence of linear-size DeMorgan circuits.[1] Despite this, the ongoing search for circuit complexity lower bounds has fostered rich and surprising connections between cryptography, learning theory, and algorithm design [HS17; GIIKKT19; San20; CKLM20; RS21; HIR23]. The *Minimum Circuit Size Problem* (MCSP, [KC00]) appears in all of these areas, asking:

> Given an n-input Boolean function f as a 2^n-bit truth table, what is the minimum s such that a DeMorgan circuit with s gates computes f ?

The **existential** question — do functions that require "many" gates exist? — was solved in 1949: Shannon proved that almost all Boolean functions require circuits of near-trivial[2] size $\Omega(\frac{2^n}{n})$ by a simple counting argument [Sha49]. The current best answer to the **explicit** question — is such a hard function in NP? — is a DeMorgan circuit lower bound of $5n - o(n)$, proved via *Gate Elimination* [IM02; ILMR02]. This is far from the popular conjecture that NP-complete problems require superpolynomial circuit size.

The **algorithmic** question — is MCSP NP-hard? — remains open after nearly fifty years [Tra84], but recently many natural variants of MCSP have been proven NP-hard. For instance, DNF-MCSP [Mas79], MCSP for OR-AND-MOD Circuits [HOS18], and MCSP for multi-output functions [ILO20] are now known to be NP-hard. Furthermore, MCSP for partial functions [Ila20] is hard under the Exponential Time Hypothesis (ETH), later extended to unconditional NP-hardness [Hir22]. Curiously, Ilango's proof of ETH-hardness for partial MCSP combined a novel perspective on Gate Elimination with a characterization of the set of optimal circuits for the n-bit OR function [Ila20]; this is one motivation for the present work.

We study the **design** question — given a Boolean function f, what is the shape of every optimal circuit computing f? For some functions, this is easy to answer: minimal circuits for n-bit OR are simply trees of $(n-1)$ OR gates. For even slightly more complex functions — like XOR — characterizing the minimal circuits seems to require intricate case analysis of Gate Elimination. Previous work showed that, when NOT-gates count towards the complexity measure, optimal XOR_n circuits are binary trees of XOR_2 sub-circuits [Kom11]. This was encouraging, but the complexity measure in Ilango's proof must ignore NOT gates — see

[1] See discussion of Kolmogorov's Conjecture on page 564 of [Juk+12].
[2] From using a lookup table.

Section 1.3 for discussion of the incompatibility. So, we characterize the set of optimal XOR circuits when NOT gates are free.

Main Theorem (Summary of Theorem 11)**.** *Optimal* (¬)XOR *circuits over the DeMorgan basis* partition into *trees of* (¬)XOR_2 *sub-circuits — even when NOT gates are free.*

This is interesting because (1) XOR is a key ingredient in many complexity lower bounds. (2) Along the way, we show that a natural formalism for circuit simplification is "well behaved" — the order of elimination steps does not matter (Theorem 27). This may be of independent interest. (3) Characterizing the set of optimal circuits for XOR when NOT gates are free would be the first step in a proof that total MCSP is NP-hard via Reverse Gate Elimination.

An immediate application of our main theorem is an efficient algorithm for optimal XOR-circuit identity testing (Corollary 16). Efficient algorithms for other circuit identity testing problems have dramatic consequences. An efficient deterministic algorithm for determining whether an arithmetic circuit computes the zero polynomial, the Polynomial Identity Testing problem, would imply strong circuit lower bounds [KI03]. Similarly, determining whether a circuit does not compute the constant 0 function, CircuitSAT, is a well-known NP-complete problem and even slight improvements over exhaustive search yield breakthrough complexity separations [GJ79; Wil10]. Perhaps because we can only identity-test for an *optimal* and not an *arbitrary* XOR circuit, we do not obtain new lower bounds from this algorithm. We hope that future work can extract more consequences from the highly constructive gate-elimination proofs of certain circuit lower bounds.

1.1 Gate Elimination & Hardness of Partial MCSP

Gate Elimination is the most general technique known for proving circuit lower bounds. In particular, the current unconditional lower-bounds known for functions in $\mathcal{B}_2$ or $\mathcal{U}_2$ basis are proven using Gate Elimination, and this technique can leave rooms for improvements. For example, the $5n - o(n)$ lower-bound for strongly two-Dependent functions over the $\mathcal{U}_2$ basis by Iwama and Morizumi [IM02] was an improvement from the $4.5n - o(n)$ lower-bound by Lachish and Raz [LR01] under the same setting. By fixing some input bits of a function f we can simplify circuits computing f. This can be useful for minimization: depending on f, the resulting circuit and the function it computes may satisfy some desirable properties, but the original circuit has at least as many gates as were removed.

Indeed, the heart of the hardness proof for Partial MCSP applies a gate elimination argument but in a *reversed* manner. Specifically, *Reverse Gate Elimination* flips the perspective of Gate Elimination as follows: given an optimal circuit, we add variables and gates to it circuit in a way that is "dual" to how these gates

would be eliminated if we were to apply standard gate elimination on the "extended" resulting circuit (i.e. applying gate elimination on the added gates will result in the original circuit). Formally, g is a *k-Simple Extension of f* if optimal circuits for g can be obtained from optimal circuits of f by adding exactly k gates and variables via Reverse Gate Elimination. This induces a natural computational problem: given the truth tables of two Boolean functions f on n variables and g on $n+k$ variables, is g a k-Simple Extension of f? One can easily notice that this decision problem can be efficiently solved given access to a MCSP-oracle. This observation gives rise to a MCSP-hardness proof framework, i.e. if one can show that deciding whether g is a *k-Simple Extension of f* is NP-hard, then MCSP is also NP-hard. Indeed, this framework was implicitly used in Partial MCSP's hardness proof.

In particular, the proof of hardness for partial-MCSP reduces an ETH-hard problem to the k-Simple Extension Problem for OR-function, which then reduces to partial-MCSP. Under ETH, the $2n \times 2n$ Bipartite Permutation Independent Set Problem (BPIS, [LMS18]) cannot be solved faster than brute-forcing over all $n!$ permutations. Because optimal OR_n circuits are simply binary trees of fan-in 2 OR gates with exactly n leaves, there are at least $n!$ many optimal circuits — one for each permutation of the n input variables. A more concrete discussion regarding how the hardness proof aligns with the Simple Extension Framework can be found in Section 1.3.3 of [Ila20].

For a high-level idea, the reduction works as follows: given a yes instance of $2n \times 2n$ BPIS, the reduction outputs the truth-table of a partial function that is a Simple Extension of OR_{4n} which can be associated with a permutation from the input instance. Thus, the hardness of partial MCSP comes from the fact that deciding whether a given partial function is a Simple Extension of OR_n is hard since one must "complete" the partial function and it cannot be done faster than brute-forcing over all possibilities given by the permutation. Furthermore, by definition, an optimal circuit computing a k-Simple Extension of OR_{4n} must also be a binary tree of $4n+k$ leaves with $4n-1$ internal nodes labeled as OR gates, which is exactly the desired number of OR gates required for an optimal circuit computing OR_{4n}, within it. In other words, a key step of showing the output partial function is the correct k-Simple Extension of OR_{4n} requires arguing what the structure of optimal circuit computing it must have. Therefore, knowing the structure of optimal circuits computing the OR-function is crucial for the hardness proof (for full details, see the proof of Lemma 16 in [Ila20]).

Could one use the same reduction to get a hardness result for total MCSP? No: optimal circuits for OR_n are so well-structured that the problem of deciding whether a *total* function is a Simple Extension of OR_n is actually easy (see the discussion in Section 1.2.2 in [Ila20]). OR_n only requires a *read-once formulas* — each of the n variables is read exactly one and each internal gate has fan-out 1. Simple Extensions of OR_n are also read-once formulas, and learning whether a given Boolean function is a read-once formula is *easy* [AHK93; GMR06].

This motivates our work. Understanding the structure of optimal circuits for *more complex* functions whose lower-bounds are well-known, such as XOR_n, the parity of n Boolean variables [Red73; Sch74], could provide a missing ingredient for proving hardness of MCSP and related problems via reverse gate elimination.

1.2 Related Work

Circuit Lower-Bounds Via Gate Elimination. In 1974, Schnorr showed that $\mathsf{MOD}^n_{3,r}$ — the function that outputs 1 if the sum of the n input bits mod 3 is equal to r — has circuit complexity $\geq 2n-4$. The inductive argument shows that, in an optimal circuit for $\mathsf{MOD}^n_{3,r}$, there is some x_i for which substituting in 0 and simplifying eliminates at least 2 gates. The resulting circuit computes $\mathsf{MOD}^{n-1}_{3,r}$ and thus the size of optimal circuits for $\mathsf{MOD}^n_{3,r}$ must have at least two more gates than $\mathsf{MOD}^{n-1}_{3,r}$. He then extended this $\Omega(2n)$ lower bound to other functions which have the same structural properties as $\mathsf{MOD}^n_{3,r}$ [Sch74].

Schnorr also showed that any optimal circuit computing XOR_n must have a size lower-bound of $3(n-1)$. The proof of this lower-bound leverages the fact that at the bottom level of the optimal circuit, a variable must be connected to two different gates, and one of those two must connect to a third gate that is not the output gate. Thus, by the nature of gate elimination, there must exist a setting for this variable such that all of the three aforementioned gates are eliminated. Then, the lower-bound proof follows via an inductive argument on the number of variables n. When NOT gates also contribute to circuit size, Red'kin showed that the size of an optimal circuit computing XOR_n is $4(n-1)$ [Red73].

Finally, there has been recent progress showing slightly better than $3n$ explicit lower bounds via Gate Elimination [FGHK16; LY22]. Stronger lower bounds achieved from Gate Elimination require more than single bit substitutions and counting gates. Advanced proofs track more complicated complexity measures, such as the sum of the number of gates and the number of dependent inputs, and substitute whole sub-functions. For example, [Sto77] achieved a $\Omega(2.5n)$ bound by substituting two variables for arbitrary functions and [DK11] tracked the number of inputs as well as the number of gates which yielded a lower bound of $3n-o(n)$. For more recent results see Section 2 of [GHKK18]. For a thorough survey of classical results, see the textbook by Wegener [Weg87].

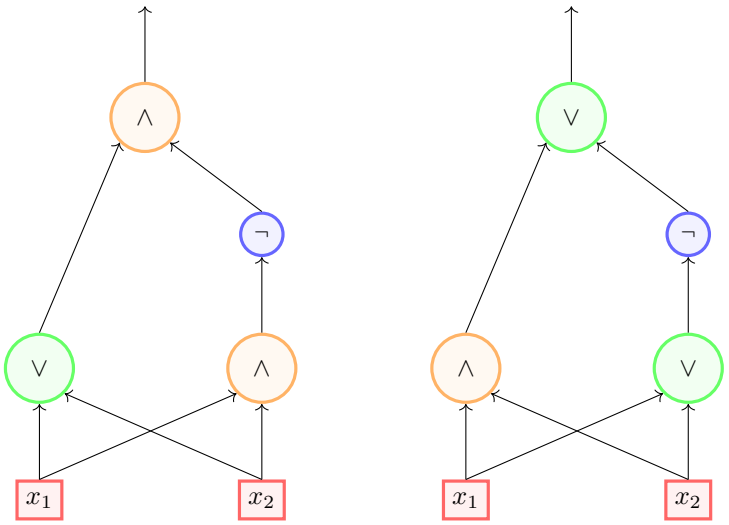
Limits of Gate Elimination Technique. Progress has been slow in proving circuit lower-bounds using Gate Elimination technique, regardless of the fact that it has been the most successful technique known so far to prove unconditional Boolean circuit lower-bounds. In the nearly 50 years since Schnorr's $2n$ lower bound proof for $\mathsf{MOD}^n_{3,r}$, the best unconditional lower bound is just smaller than $3.1n$ [LY22]. This led researchers to speculate that the technique could not prove nonlinear bounds [Weg87]. Recently, this intuition has been proven true by Golovnev, Hirch, Knop, and Kulikov. Namely, they constructed functions that are "resistant"

to gate elimination: any constant number of substitutions reduces their circuit complexity measure only by a constant, and thus, Gate Elimination cannot be used to prove lower-bounds better than $5n$ [GHKK18]. They also construct circuits which are resistant with respect to slightly more advanced complexity measures like those used in [DK11]. Thus, in order to prove stronger bounds, either new techniques will need to be developed, stronger assumptions must be made, or both. For example, Golovnev, Kulikov, and Williams have developed new unrestricted depth-three circuit reductions by assuming a hypothesis about DeMorgan formulas [GKW21].

However these limits are orthogonal to our work. We use Gate Elimination *indirectly*, to extract the properties of optimal circuits whose lower-bound was proven using Gate Elimination as well as the ultimate "shape" that all such optimal circuits share.

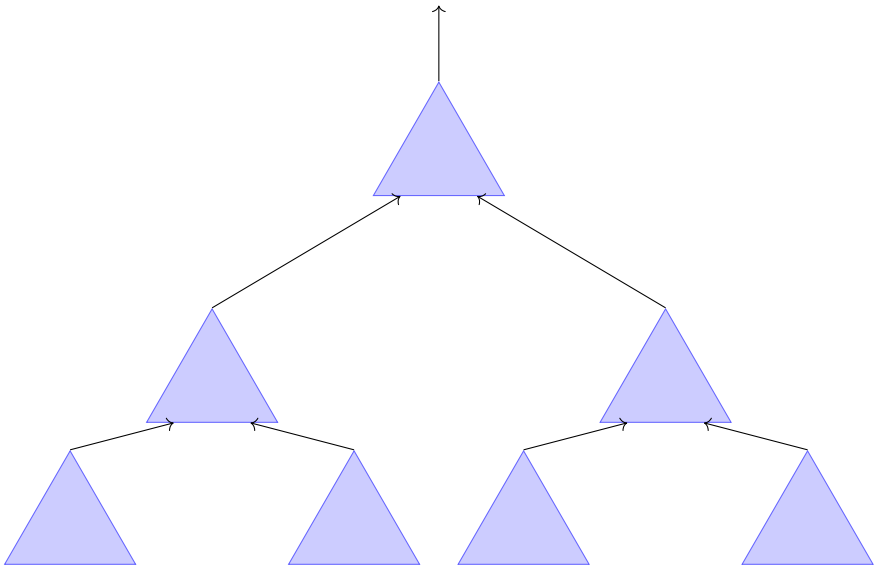
1.3 Comparison to Prior Study of Optimal XOR Circuits

We recall that under a different complexity measure — where NOT gates also contribute to circuit size — this problem has already been studied and a similar result holds. Namely, Kombarov showed that optimal circuits computing XOR_n when NOT gates count must consist of $n-1$ non-intersecting $(\neg)\mathsf{XOR}_2$-circuits [Kom11], and furthermore, the one shape that all such optimal circuits share looks like a "binary tree" of blocks where each looks like one of 1a and 1b below. We will discuss the main ideas of the Kombarov's proof technique later in this section, but in short, the proof exploits the fact that XOR is *downward self-reducible* to conduct an inductive argument over the number of input variables. Furthermore, given the $4(n-1)$ lower bound by Red'kin, the proof utilizes Gate Elimination to argue that each block must look like either 1a or 1b since otherwise, the lower bound will be violated. Our proof exploits the same ideas, but we use a *rewriting system* to handle the flexibility of the free NOT gates which is the main difference between our complexity measurement compared to the one used by Kombarov. We will also provide a quick discussion on our motivation for the choice of complexity measurement in this work at the end of this section.



(a) XOR_2 block

(b) $\neg\mathsf{XOR}_2$ block



(c) $(\neg)\mathsf{XOR}_8$ optimal circuit shape which consists of $8-1=7$ non-intersecting blocks where each is either Figure 1a or 1b

Figure 1: An example of the "true shape" of optimal circuits computing XOR_n proved in this work and also in [Kom11]

Figure 2

Furthermore, Kombarov extended this result to other complete bases, i.e. both the complexity and the structure of optimal circuits for XOR were established on these bases [Kom18]. This work and ours helps unify different bases and circuit complexity measurements — in all settings, the shape of minimal XOR circuits remains the same. Progress on other bases continues. Recently, a generalized lower-bound result on Boolean linear functions on arbitrary complete bases was introduced by Red'kin [Red20], and later, Kombarov expanded the study of circuit lower-bounds for XOR to the regime of infinite fan-in circuits

[Kom22].

An overview of the proof technique in [Kom11]. The XOR circuit structure proof by Kombarov proceeds via case analyses and induction on gate elimination. At a high level, it uses the following observations.

- If $\wedge$, $\vee$, and NOT-gates for each block are not "wired" together as shown in Figure 1a or 1b, then the optimality of the XOR-circuit size will be violated. This can be shown using case analysis on a Gate Elimination argument where one argues that if the gates are not placed correctly, then there exists a one-bit restriction will eliminate *too many* gates which contradicts the circuit-lower bound by [Red73].
- The XOR function is *downward self-reducible*, i.e. given a circuit computing XOR_n, a one-bit restriction yields a new circuit computing XOR_{n-1}. This observation is useful for an inductive argument to argue that the ultimate shape of a circuit computing XOR_n must look like a binary tree.

We exploit the same observations, using a *rewriting system* rather than direct case analysis to save "nesting depth" of cases and handle the additional flexibility in circuit structure afforded by free NOT gates. Therefore, the ultimate shape of optimal XOR-circuits is determined by how we place the $\wedge$ and $\vee$-gates only, although it seemed plausible to believe that one can take advantage of the *free* NOT-gate resources to shape the circuit differently. This raises the obvious question,

Should NOT Gates be Costly? What are the consequences of giving away NOT gates for free? It turns out, both complexity measures are well-motivated.

- On the one hand, the number of NOT-gates plays an important role in the study of learning the exact identification of functions computed by circuits/formulas with queries — a research topic that has been active since the 90s [AHK93; BCKT94; BHH95; HPRW96]. Fairly recently, a result by Blais et al showed the significance of NOT-gates in learning the exact identify of the function computed by a given circuit [BCOST14]. Specifically, they upper-bound the number of NOT-gates that the given circuit contains where the learning task still remains "easy." Once that threshold is *slightly* passed, the task is "hard" as the number of required queries becomes exponential. Thus, charging for NOT gates is essential in this setting.
- On the other hand, NOT-gates do not seem like a primary contribution to circuit lower bounds. Specifically, Schnorr's lower bound for XOR_n is $3(n-1)$ without counting NOT-gates compared to the $4(n-1)$ lower bound by Red'kin. Furthermore, the proof of Schnorr is much less complicated than Red'kin's as the latter required deep case analyses to argue where NOT-gates can be used. This means

that ignoring NOT-gates can simplify the lower bound proofs which could help to improve consequences of (indirect) gate elimination.

Simple Extension Requires Free NOT Gates. Our motivation for identifying the one true shape of optimal XOR-circuits is to study hardness of the XOR Simple Extension problem. Informally, a Boolean function g is a Simple Extension of another Boolean function f if we can obtain an optimal circuit C_g computing g by adding **exactly** k costly gates and variables to an optimal circuit C_f computing f. Counting NOT-gates in our measurement would break the definition of Simple Extension because NOT is a unary function; it cannot be "spliced" into a wire to add a single variable at the cost of one gate.

1.4 Proof Techniques

The proof of Theorem 11 has two parts. First, we formulate circuit simplification as *abstract graph rewriting* — a general notion of graph transformation according to fixed rules. We show that circuit simplification is a *convergent* abstract rewriting system; this means *the order of elimination steps does not matter:* the same restriction always simplifies to a unique circuit normal form after gate elimination. The technical bulk of this proof is machine-checkable, because it follows from running the Knuth-Bendix algorithm on a system of equations that describe gate elimination over the DeMorgan basis. The contribution is conceptual: we borrow ideas and results from Theory B to show that a basic tool of Theory A is "well behaved". This perspective on gate elimination proofs could be of independent interest.

Second, we carry out an elementary but intricate case analysis of restricting and eliminating gates from optimal XOR circuits. Essentially we extract more information from Schnorr's lower bound, which can be seen as reasoning about "templates" that must be found in any optimal XOR circuit. We push this process to the limit, fully characterizing the "shape" of all such circuits. Convergence of our circuit simplification system allows us to both work with circuits in convenient normal forms and avoid any case analysis that would arise from varying the order of simplification steps.

1.5 Open Problems

An immediate open problem is whether we can establish the structure of optimal circuits for other functions whose lower-bounds are known. For instance, in the regime of DeMorgan formulas, we have a lower bound of $\Omega(n^{2.5-o(1)})$ for Andreev's function [And87] [3]. Thus, it would be interesting to see a how much this lower bound would change in the circuit regime as well as the structure of optimal circuit computng Andreev's

[3] this lower bound was later improved to $\Omega(n^{3-o(1)})$ by analyzing the *shrinkage exponent* of DeMorgan formulas [PZ93; IN93; sta98]

function.

Lastly, can we use Theorem 11 to address an open problem proposed by Ilango in [Ila20], i.e. can we prove hardness of total MCSP via the k-Simple Extension problem for XOR_n?

1.6 Paper Outline

The organization of the rest of the paper is as follows. In Section 2, we describe circuits as *term graphs* and gives some basic properties of XOR. In Section 3, we explain how to formulate proofs using Gate Elimination as *term graph rewriting* (a more detailed exposition can be found in Appendix E) followed by a demonstration via Schnorr's lower-bound for XOR. Lastly, in Section 4, we prove our main Theorem 11 and then apply it to speed up optimal XOR circuit identification.

2 Preliminaries

2.1 Circuits as Term Graphs

We study general circuits over the *DeMorgan basis* $\mathcal{B} = \{\wedge, \vee, \neg, 0, 1\}$ of Boolean functions: binary $\wedge$ and $\vee$, unary $\neg$, and zero-ary (constants) 1 and 0. Circuits take zero-ary variables in $X = \{x_1, x_2, \ldots, x_n\}$ for some fixed n as inputs. Usually, circuits are described as DAGs with nodes labeled by function symbols or variables and edges as "wires" between the gates. Here, to apply results from term graph rewriting, we describe circuits as *hypergraphs*, tuples $C = \langle V_C, E_C, \mathrm{lab}_C, \mathrm{att}_C \rangle$ where V_C and E_C are finite sets of *vertices* (or nodes) and *hyperedges*, $\mathrm{lab}_C : E_C \to \mathcal{B} \cup X$ is an edge-label function recording the type of each edge, and $\mathrm{att}_C : E_C \to V_C^{\leq 3}$ is an attachment function which assigns a non-empty string of nodes to each hyperedge e such that $|\mathrm{att}_C(e)| = 1 + \mathrm{arity}(\mathrm{lab}_C(e))$. In this setting, hyperedges represent logic gates and nodes are "connection points" on the "circuit board" between gates.

2.2 $(\neg)\mathsf{XOR}_n$: The Parity Functions

We define the parity functions XOR_n formally as:

Definition 1. For an n bit input $\vec{x}$, we define:

$$\mathsf{XOR}_n(\vec{x}) = \begin{cases} 1 & \text{if an odd number bits of } \vec{x} \text{ are 1 and,} \\ 0 & \text{otherwise.} \end{cases}$$

Observe this definition extends XOR_n to $n = 1$ where $\mathsf{XOR}_1(x) \equiv x$. While XOR_n is often defined starting

at $n \geq 2$ [Weg87; Red73; Kom11], this deviation is not unnatural and will prove convenient for our inductive arguments. For the rest of the paper, $(\neg)f$ means "f or $\neg f$" where f is a Boolean function. We now give some basic facts about $(\neg)\mathsf{XOR}_n$ that are immediate consequences of the definition above.

Fact 2 ($(\neg)$XOR is Fully DSR)**.** XOR_n *is* fully downward self-reducible*, i.e. for any input* $x \in \{0,1\}^n$*, any non-empty sets* S *and* T *partitioning* $[n]$*,*

$$\mathsf{XOR}_n(x) = \mathsf{XOR}_2(\mathsf{XOR}_{|S|}(x_S),\ \mathsf{XOR}_{|T|}(x_T))$$

where $x_S = \{x_i : i \in S\}$ *and* $x_T = \{x_i : i \in T\}$*. Furthermore, this means for any partial assignment* $\vec{\alpha}_S$ *of variables in* x_S*,* $\mathsf{XOR}_n(x)|_{x_S = \vec{\alpha}_S} = (\neg)\mathsf{XOR}_{|T|}(x_T)$*. The same is also true of* $\neg\mathsf{XOR}_n(x)$

Fact 3 (All Subfunctions of $(\neg)$XOR are Non-Degenerate)**.** $(\neg)\mathsf{XOR}_n$ *not only depends on all of its inputs but it is also* maximally sensitive*, i.e. for all* $i \in [n]$*, for all assignments* α*,* $(\neg)\mathsf{XOR}_n(\alpha) \neq (\neg)\mathsf{XOR}_n(\alpha \oplus e_i)$*.*

3 Well-Behaved Circuit Simplification

Proofs by gate elimination often repeat the following argument to show that a circuit C has property $\mathcal{P}$.

1. Assume that C does **not** have property $\mathcal{P}$.
2. Select a variable x_i and constant α for substitution $\{x_i \mapsto \alpha\}$ using $\neg\mathcal{P}$ and the structure of C.
3. ***Simplify*** C under the substitution $\{x_i \mapsto \alpha\}$, to obtain a constant-free circuit C'.
4. Argue that a critical property $\mathcal{P}'$ of C' implies a contradiction, therefore C **must** have property $\mathcal{P}$.

We formalize the simplification procedure used in step three above. Usually, this is not necessary: the critical property is something like $\mathcal{P}'$ = "simplification eliminated four gates" and it is clear that every sequence of simplification steps reaches a circuit C' with property $\mathcal{P}'$. However, we must assert post-simplification properties like $\mathcal{P}'$ = "input x_j has fanout one," where x_j was the sibling of x_i in C. These more delicate properties are not so easily seen to hold after *every* terminated simplification.

To avoid ad-hoc arguments and lengthy case analyses, we develop a *convergent* simplification procedure $\mathcal{S}$ for circuits: every valid run of $\mathcal{S}$ on C with α substituted for any input x_i terminates with the *same* circuit C'. Therefore, to carry out the argument template above, one need only exhibit a particular run of $\mathcal{S}$ and argue that the resulting C' has critical property $\mathcal{P}'$.

3.1 Rewriting Systems: Definitions & Desiderata

An *abstract rewriting system* is just a set of objects A together with a binary relation $\to$ on A called the *rewrite* relation. We give a system where A contains Boolean circuits over the DeMorgan basis and $C \to C'$ holds when C simplifies to C' via a single step of gate elimination. We'll first introduce some terminology about abstract rewriting systems as well as define some desirable properties. For elements $a, a' \in A$, write $a \xrightarrow{*} a'$ to mean that there is a finite path of rewrite steps from a to a', and say that a is *in normal form* if there is no b such that $a \to b$.

Definition 4 (Definition 2.1.3 of [BN98])**.** The rewrite relation $\to$ is called

terminating iff there is no infinite path $a_0 \to a_1 \to \ldots$

confluent iff for every triple of objects $a, b, b' \in A$, if $a \xrightarrow{*} b$ and $a \xrightarrow{*} b'$, then there is a $c \in A$ such that both $b \xrightarrow{*} c$ and $b' \xrightarrow{*} c$ —

convergent iff it is both confluent and terminating.

3.2 Circuit Simplification: System $\mathcal{S}$

Let A be the set of finite DeMorgan circuits encoded as hypergraphs. Our system $\mathcal{S}$ is a special case of Term Graph Rewriting following [Plu99] and detailed in Appendix E. There are three parts to the system: (1) *hyper-graph morphisms* — a notion of pattern matching for sub-circuits, (2) a set of circuit rewrite rules $\mathcal{R}_B$ formulated as left- to right-hand pattern pairs (Definition 24), and (3) a procedure for replacing patterns in circuits (Definition 25). System $\mathcal{S}$ is then the binary relation induced by setting $C \to C'$ when C matches the left-hand side l of a rule $\langle l, r \rangle$ and C' is the result of substituting pattern r for l in C. We run the Knuth-Bendix algorithm (Theorem 22) and invoke Corollary 1.7.4 of [Plu99] to show:

Theorem 5. *$\mathcal{S}$ is a convergent abstract rewriting system.*

Definition 6 (Hypergraph Morphism)**.** For hypergraphs G and H, a *hypergraph morphism* $f : G \to H$ is a pair of functions $f_E : E_G \to E_H$ and $f_V : V_G \to V_H$ that preserve

labels for every hyperedge e of G, f_E sends e to an edge of H with matching label — $\text{lab}_G(e) = \text{lab}_H(f_E(e))$ and

attachments $\forall e \in E_G \quad f_V^*(\text{att}_G(e)) = \text{att}_H(f_E(e))$

Definition 7 (Pattern & Redux in Circuits)**.** Circuit D is an *instance* of pattern L if there is a morphism $p : L \to D$ sending root_L to root_D. Then, given a vertex α in circuit C and a rule $L \mapsto R$, the pair $\langle \alpha, L \mapsto R \rangle$ is a *redex* if $C[\alpha]$ is an instance of L.

Algorithm 1 Step of Circuit Simplification System $\mathcal{S}$, defining $C \to C'$

Require: C is a circuit containing the redex $\langle \alpha, R \mapsto L \rangle$

$C_1 \leftarrow C - \{a\}$ where $a = \mathrm{res}^{-1}(\alpha)$ ▷ *Remove the unique gate with output wire α*

$C_2 \leftarrow C_1 + R$ ▷ *Disjoint union: rhs of the matched rule with C*

$C_3 \leftarrow$ Identify vertex α with root_R of C_2 ▷ *Connect R to the appropriate element(s) of C*

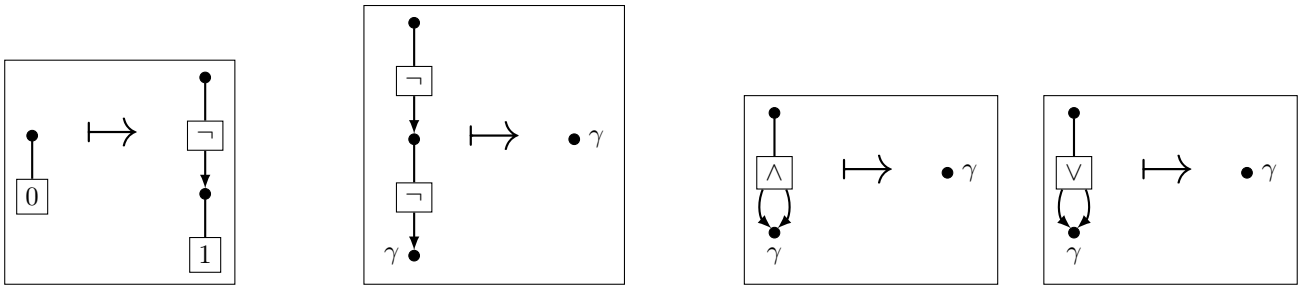
if $\gamma \in R$ **then** ▷ *Does R reuse a subcircuit?*

    $C_4 \leftarrow$ Identify vertex $p(\gamma)$ with γ of C_3 ▷ *Yes — connect C to the appropriate element of R*

else

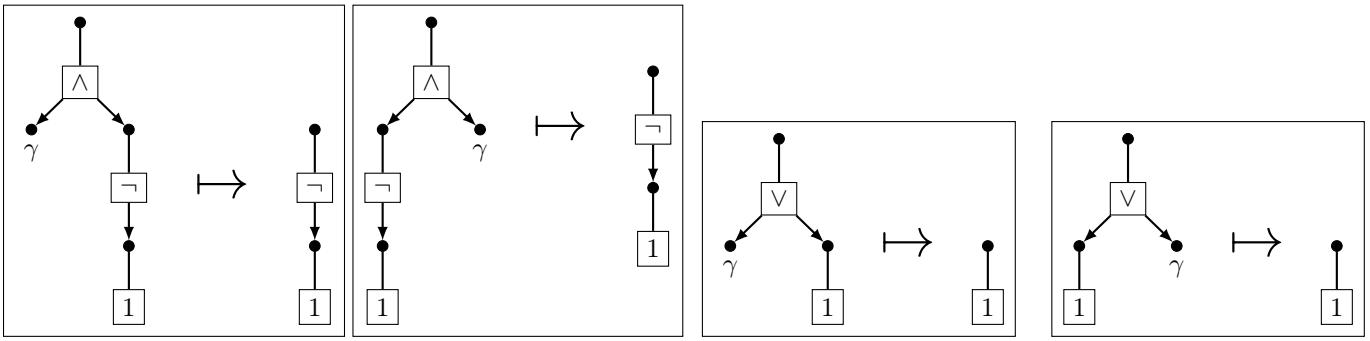
    $C_4 \leftarrow C_3$ ▷ *R does not reuse any subcircuits — do nothing*

$C' \leftarrow$ Garbage collection: remove all vertices and edges unreachable from root_{C_4}



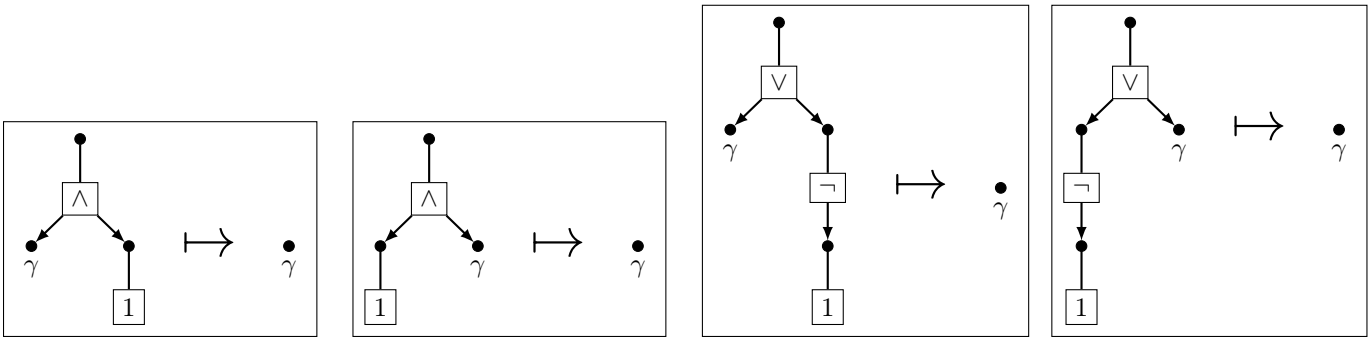
(a) Zero Elimination (b) 2xNegation Elimination (c) AND-De-Dupe (d) OR-De-Dupe

Figure 3: **Normalizing Rules**



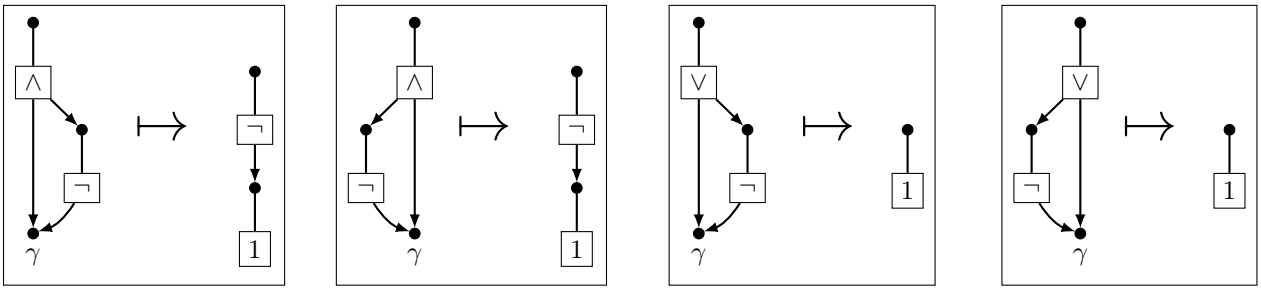
(a) AND-Left (b) AND-Right (c) OR-Left (d) OR-Right

Figure 4: **Fixing Rules**



(a) AND-Left (b) AND-Right (c) OR-Left (d) OR-Right

Figure 5: **Passing Rules**



(a) AND-Left (b) AND-Right (c) OR-Left (d) OR-Right

Figure 6: **Tautology Rules**

3.3 Circuit Simplification in Action: Warming Up With Schnorr's Lower Bound

We now prove Schnorr's lower bound using $\mathcal{S}$. The purpose of this is three-fold. It will

1. demonstrate that $\mathcal{S}$ is powerful enough to capture traditional gate elimination arguments,
2. refresh the reader on Schnorr's argument which lays the foundation for the proof of our Main Theorem,
3. and introduce notions such as costly, topological sorting of gates, successors and fanout that we will use repeatedly throughout our proofs in Section 4.

On the final point we collect these notions conveniently for the reader in Definition 9 at the end of the section. We also provide an alternative presentation of this proof in the structured proof format of Lamport [Lam12] in Appendix A. Due to the case analysis and repeated proofs by contradiction present here and in the proof of Theorem 11, the alternative format may be easier to verify.

Theorem 8 (Schnorr, [Sch74])**.** XOR_n *requires* $3(n-1)$ *gates in the DeMorgan basis.*

Proof. Let C be an optimal circuit for XOR_n with $n \geq 2$. In the original proof, the goal was to find some **setting** of an input *node* such that gate elimination would remove at least 3 *costly* gate *nodes* ($\wedge$ and $\vee$). Besides notational differences, our goal remains the same. We will find a **substitution** of an input *hyperedge* which causes at least 3 *costly* gate *hyperedges* (those labeled $\wedge$ or $\vee$) to be removed during **rewriting**. Following Schnorr we build up the circuit locally around an input by repeated proofs by contradiction; we perform substitutions and rewrites to contradict C's optimality or the downward self-reducibility of XOR_n thereby forcing C to have the desired structure.

In order to smooth the transition from viewing circuits as graphs to viewing them as term graphs, we will simply refer to input hyperedges as inputs and gate hyperedges as gates. We describe the substitution and rewriting steps at a high level. For explicit demonstration of a simplification step, see Appendix B.

We wish to first sort the gates in "topological order". In the traditional view of circuits as DAGs (where gates are nodes) this notion is straightforward. However, since our gates are edges, we must do so indirectly. We can sort the vertices of C topologically and, since each node is the result node of a unique edge, we then order the gates according to their result nodes. From this point on when we refer to sorting inputs or gates topologically, formally we are doing this process.

Fix a topological order and let h be the first costly gate in C. Under the traditional view of circuits, we would conclude h has x_i (or $\neg x_i$) and x_j (or $\neg x_j$) as inputs for some $i, j \in [n]$. Formally, this means h has two argument nodes whose terms are x_i (or $\neg x_i$) and x_j (or $\neg x_j$). Before we continue, however, we streamline our verbiage. We notate the possibility of $\neg$ gates by defining the shorthand $(\neg)f$ to mean "f (or

$\neg f$)." If f' has an argument node whose term is $(\neg)f$ we say that f *feeds* into f' and that f' is a *successor* of f. Lastly if the label of a gate f is $\wedge$ or $\vee$ we say f is *costly*. Combining these allows us to instead say h is the costly successor of two inputs x_i and x_j for some $i, j \in [n]$ — maintaining the formalism of our system while being more inline with the original proof.

We assert $i \neq j$. Otherwise $h \equiv (\neg)x_i \diamond (\neg)x_i$ for some $\diamond \in \{\wedge, \vee\}$. If this occurred, we could apply a normalizing or tautology rule; rewriting C would then delete h. Since h is a costly gate this would decrease the the size of C, contradicting C's optimality.

We now wish to say that x_i has *fanout* at least 2 where we define *fanout* to the number of *costly* successors a gate or input has. Again, assume the contrary: h is x_i's only costly successor. We can then substitute $x_j = \alpha$ where α is set so that during rewriting, we can apply a fixing rule to h. This would mean that $C|_{x_j=\alpha}$ does not depend on x_i violating Fact 3 (All Subfunctions of $(\neg)\mathsf{XOR}$ are Non-Degenerate).

Let f be another costly successor of x_i. We can conclude that $(\neg)f$ is not the output of the circuit (i.e. the root node). If it were, then we could substitute $x_i = \beta$ and fix $(\neg)f$ during rewriting. This would fix the output of the circuit so that $C|x_i = \beta$ is constant contradicting Fact 2 ($(\neg)\mathsf{XOR}$ is fully DSR).

Let f' be a costly successor of f and observe $h \neq f'$ since $f < f'$ in our topological ordering and h was the first costly gate in the ordering. We now observe that if we set $x_i = \beta$ so that f is fixed, then during rewriting we eliminate h (using a fixing or passing rule), f (using a fixing rule), and f' (using a fixing or passing rule). This is because once f is fixed, then the fixing $(\neg)1$ now feeds into f'. In this case we say that $(\neg)1$ *inherited* f' as a successor after this rewrite applies, and thus another rewriting rule will apply deleting f'.

Thus XOR_n requires at least 3 more costly gates than XOR_{n-1}. Since XOR_1 requires zero costly gates, we have using induction that XOR_n requires at least $3(n-1)$.

□

Definition 9 (Formalization of Nomenclature)**.** We collect the terms and concepts defined in the proof of Schnorr (Theorem 8) above.

costly gate refers to a gate whose label is $\wedge$ or $\vee$.

topological sorting the gates of C: translates to topologically sort the result nodes of C and then order the gates by their unique result nodes.

f **feeds into** h if h has an argument node whose term is $(\neg)f$.

h **is a successor of** f if f feeds into h.

the fanout of f is the number of costly successors of f.

f **inherits** h if during rewriting h becomes a successor of f after a rewrite rule is applied.

4 Optimal (¬)XOR Circuit Structure

The $3(n-1)$ lower bound shown by Schnorr in Theorem 8 has a matching upper bound and the construction is straightforward: take any binary tree with n leaves labeled $x_1, \ldots, x_n$ where the $n-1$ interior nodes have been labeled by $\oplus$ and replace the $\oplus$ nodes with any circuit of size 3 that computes XOR_2. Furthermore, since ¬ gates do not increase circuit size, any circuit for XOR_n can easily be transformed into an equal size circuit computing $\neg\mathsf{XOR}_n$ and vice versa. Combining these observations yields the following:

Corollary 10 ([Sch74; Weg87])**.** *A circuit C computing $(\neg)\mathsf{XOR}_n$ is optimal if and only if $|C| = 3(n-1)$.*

In this section we will show that binary trees of optimal $(\neg)\mathsf{XOR}_2$ subcircuits are the *only* optimal circuits for computing XOR_n. Formally,

Theorem 11. *Optimal* (¬)XOR *circuits* partition into *trees of* $(\neg)\mathsf{XOR}_2$ *sub-circuits — even when NOT gates are free. Formally, for every circuit C with the minimum number of AND,OR gates computing XOR_n, there is a partition of the gates of C into $(n-1)$ blocks together with a multi-labelling of each wire w in C by tuples $\langle i, t\rangle$ where $t \in \{\mathsf{in}, \mathsf{out}, \mathsf{core}\}$ describes the role that w plays in block i, such that:*

1. *Each block is a three-gate XOR_2 sub-circuit, with distinguished* input*,* output*, and* core *wires.*
2. *The* input *wires of every block are also the output wires of a different block or the input gates.*
3. *Edge-contraction of C by all* core *wires results in a binary tree.*

The wording of Theorem 11 is versatile; it holds in both the traditional view of circuits and in our graph rewriting system. Here *wires* refers to the *attachment points* between gates, i.e. the nodes in our hypergraphs. Before we prove our main theorem, however, we first introduce some auxiliary results that we will repeatedly leverage in our main proof.

4.1 Useful Properties of XOR

Combining Facts 2 and Facts 3, if we substitute for a single variable in a $(\neg)\mathsf{XOR}_n$ circuit where $n \geq$ and perform rewriting we are guaranteed to get a circuit which computes $(\neg)\mathsf{XOR}_{n-1}$. Applying the tight version of Schnorr, Corollary 10, we see that this rewriting cannot remove more than three costly gates. Formally,

Corollary 12 (Elimination Rate Limit for Optimal XOR Circuits)**.** *Let C be an optimal circuit computing $(\neg)\mathsf{XOR}_n$ where $n \geq 2$. Let C' be the circuit after substituting $x_i = \alpha$ for some $i \in [n]$ and $\alpha = \{0, 1\}$ and performing rewriting. We have that $|C'| \geq |C| - 3$ and C' is not constant.*

We will repeatedly apply this corollary in our proof: deviation from the prescribed structure will often allow us to substitute and remove more than three distinct costly gates. The other main source of contradictions will be substitutions and rewrites that disconnect inputs (violating Fact 3) or that leave inputs with exactly one costly successor. This violates the fact that XOR_n reads each of it's inputs twice. We give a formal proof for completeness.

Lemma 13 ((¬)XOR is Read-Twice (Folklore))**.** *Let C be a normalized optimal circuit computing $(\neg)\mathsf{XOR}_n$ where $n \geq 2$. The fanout of every variable C is exactly 2.*

Proof. Let C be an optimal normalized circuit computing $(\neg)\mathsf{XOR}_n$ for arbitrary n. Suppose there is a variable x_i whose fanout is not 2. There are two cases: (1) the fanout of x_1 is 1 and (2) the fanout of x_1 is at least 3.

(1) Assume h is x_i's unique costly successor and let f be the other gate or variable whose successor is h. Let F be the set of input variables that f depends on (i.e. that are reachable from f) and observe that $x_i \notin F$. Consider the truth-table of f and observe that f must be a non-constant function of the variables of F. If it were constant, then we could substitute f with this constant and remove h with a passing or fixing rule, reducing the size of the circuit and violating optimality. Therefore, there is an assignment of the variables in F such that if we substitute and rewrite, eventually a constant will feed into h that allows us to remove it with a fixing rule. This disconnects x_i from the circuit, which violates Fact 3 since the circuit is computing $(\neg)\mathsf{XOR}_{n-|F|}$ but does not depend on x_i.

(2) Suppose x_i has more than two costly successors. Let h_1, h_2 and h_3 be three of these and without loss of generality assume these are indexed in ascending topological order. Observe that $(\neg)h_3$ is not the output of the circuit as otherwise we could substitute $x_i = \alpha$ to fix h_3 and make the circuit constant. This would violate Corollary 12. Let f_3 be the costly successor of h_3 and notice that, since h_1, h_2 and h_3 are in ascending topological order, f_3 is a new distinct gate. Thus if we substitute $x_i = \alpha$ to fix h_3, during rewriting h_1, h_2, h_3 and f_3 will all be removed, with a passing or fixing rule applying to f_3 after applying a fixing rule to h_3. This violates Corollary 12.

Both cases reach a contradiction and therefore every input has two costly successors in any normalized optimal circuit computing $(\neg)\mathsf{XOR}_n$. □

4.2 Optimal $(\neg)\mathsf{XOR}_n$ Circuits Are Binary Trees of $(\neg)\mathsf{XOR}_2$ Blocks

We now have the tools necessary to prove Theorem 11. The proof will proceed via induction, however most of the work will be in proving that we can find a block in any XOR_n circuit which we can "peel off" with a single variable substitution. The resulting circuit will compute XOR_{n-1} allowing us apply our inductive

hypothesis in order to get a partition we can lift back up to the original circuit. For this reason we will first prove the following lemma:

Lemma 14. *Let C be a circuit computing XOR_n for $n \geq 3$. There exists two inputs x_i and x_j that feed into a block B in C as described in Theorem 11.*

As in the proof of Theorem 8 (Schnorr) and Lemma 13 ((¬)XOR is read-twice), our strategy will be to build up a local view around some input variables, forcing the optimal circuit to have the desired structure by arguing that any deviation would contradict Facts 2, 3, Lemma 13, and Corollary 10 (optimal circuit size for (¬)XOR is $3(n-1)$). Since this will involve a delicate case analysis, we also present a structured proof in Appendix C which both makes the cases more explicit and may be easier to verify.

Proof. Let C be an optimal normalized circuit computing XOR_n where $n \geq 3$. We first identify two variables which will be inputs to block B (which we will later prove that $B \equiv (\neg)\mathsf{XOR}_2$). Let h be the topologically-first gate of C. As in the proof of Theorem 8, we know that h must be the successor of two distinct inputs x_i and x_j for some $i, j \in [n]$ since otherwise we can apply normalizing or tautology rules if we rewrite C, which contradicts that it is an optimal normalized circuit.

By Lemma 13, we know that both x_i and x_j have exactly two costly successors. Let h_i' be the other successor of x_i and h_j' be the other successor of x_j. Let f_i be the other input to h_i' and let f_j be the other input to h_j'. Our goal will be to prove that these successors are the same gate, i.e. $h_i' = h_j'$, however, we must first prove that all of the h gates have exactly one costly successor. To this end, we first observe that h must have at least one costly successor. If it did not (and h or $\neg h$ was the output of the circuit), then we could substitute $x_i = \alpha$ to fix h. This would make the circuit constant contradicting Corollary 12 (Elimination Rate Limit for Optimal XOR Circuits). Therefore h has at least one costly successor p. Via the same argument, we can also show h_i' has at least one costly successor.

Let r be a costly successor of h_i'. We now argue that r is the only costly successor of h_i'. Suppose for the sake of contradiction that h_i' also feeds into another costly gate r'. Again, if we substitute $x_i = \beta$ which fixes h_i' during rewriting, we find h, h_i', r and r' are all removed. Notice h and h_i' are not equal to either r or r' since circuits are acyclic and h is first in our topological order. The removal of four costly gates from a single substitution violates Corollary 12. We can argue symmetrically to show h_j' has exactly one costly successor u.

Our next step is to show that p is the only costly successor of h. Assume otherwise and let p' be another costly gate fed by h. We perform a case analysis on the identities of p, p', h_i' and h_j' and whether $p \stackrel{?}{=} h_i'$, $p \stackrel{?}{=} h_j'$, $p' \stackrel{?}{=} h_i'$, $p' \stackrel{?}{=} h_j'$ and $h_i' \stackrel{?}{=} h_j'$. Any satisfiable set of these equalities falls into one of the three cases listed below. For a full description of which sets of equalities fall into which cases see Figure 13 in

Appendix C. In each case, we apply suitable substitution for x_i which eliminates four distinct gates and violates Corollary 12.

1. Both p and p' are distinct from h'_i and h'_j: Substitute $x_i = \alpha$ to fix h. During rewriting, h'_i, h, p, p' are removed.

2. Exactly one of p and p' is equal to h'_i or h'_j. Without loss of generality, assume $p' = h'_j$ and that $p \neq h'_i$. Again we can substitute $x_i = \alpha$ to fix h. Rewriting then removes h'_i, h, p, p'.

3. Both p and p' are equal to one of h'_i and h'_j. Substituting $x_i = \alpha$ to fix h therefore eliminates h'_i, h, h'_j and r. The last elimination comes from the fact that after fixing h, both of h'_i's inputs are constant — thus regardless of whether h'_i is removed with a passing or fixing input, a constant will then feed into r. Lastly r is distinct from h'_j in this case since h and x_j are the two inputs of h'_j.

We now will show that $h'_i = h'_j$. Assume the contrary. Since h'_i and h'_j are distinct, p is not equal to at least one of them. Without loss of generality, assume $p \neq h'_i$. Again, substitute $x_i = \alpha$ so that h is fixed during rewriting. We also see that h'_i and p are removed by this. No other costly gates can be eliminated due to Corollary 12. Let C' be the circuit after rewriting, and notice $|C'| = |C| - 3$ and that C' is normalized. Therefore, C' is a normalized circuit computing $(\neg)\mathsf{XOR}_{n-1}$ and by Lemma 13, we should have that x_j has two costly successors. However, this is not the case. Since h was removed via a fixing rule, x_j loses a successor upon h's removal. Furthermore, since we are assuming $h'_i \neq h'_j$, we know $x_j \neq f'_i$. This means x_j cannot gain a successor even if h'_i is removed with a passing rule. Lastly, x_j can only gain a successor upon p's removal if $p = h'_j$. However in this case, since h'_j only has one costly successor, x_j still sees a net loss of one costly successor. We reach a contradiction since x_j does not feed into two costly gates in C', an optimal circuit for $(\neg)\mathsf{XOR}_{n-1}$ and therefore $h'_i = h'_j$.

Define $h' := h'_i = h'_j$ and $p' := r = u$. We now wish to show $p = p'$. Again, assume they are distinct, and set $x_i = \alpha$ to fix h. This also will eliminate h' but there are two cases depending on which rule eliminates it:

1. h' is eliminated via a fixing rule: in this case p' will also be removed, and thus h, h', p, p' are four distinct gates which are eliminated violating Corollary 12.

2. h' is eliminated via a passing rule: Then x_j inherits p' as it's only successor. However, since three gates are eliminated (h, p, h'), x_j should feed into two costly gates since the circuit is an optimal normalized $(\neg)\mathsf{XOR}_{k-1}$ circuit which contradicts Lemma 13.

Let B be the block consisting of h, h' and $(\neg)p$—where we include $\neg p$ in B if p only feeds into $\neg$ p. It remains to show that $(\neg)p$ has two costly successors in C. Substitute $x_i = \alpha$ and perform rewriting. Let

C' be the circuit after rewriting. While it is obvious that h and h' have been removed, notice p is as well since h is removed using a fixing rule and thus a constant feeds into p. Furthermore by Corollary 12, no other gates are removed. Therefore, we can see that x_j must inherit p's successors. We observe that C' is a normalized optimal circuit computing $(\neg)\mathsf{XOR}_{k-1}$ since C' has three fewer gates than C. By Lemma 13, x_j has two costly successors in C' which must have originally been exactly $(\neg)p$'s costly successors.

□

We now conclude the proof of our main theorem. For $n = 1$ and $n = 2$ the theorem trivially holds: $(\neg)x_i$ is the unique normal optimal circuit for $(\neg)\mathsf{XOR}_1$ and normal optimal $(\neg)\mathsf{XOR}_2$ circuits trivially define a single block.

Inductive Step of Theorem 11. Assume the statement holds for some $k - 1 \geq 2$. Let C be a normal optimal circuit computing $(\neg)\mathsf{XOR}_k$ for $k \geq 3$. By Lemma 14, we know C must have a block B that is fed by two distinct inputs $(\neg)x_i$ and $(\neg)x_j$. Let the three gates be h, h', and p where $(\neg)p$ is the output gate of B. We label the wires from $(\neg)x_i$ and $(\neg)x_j$ as `in`, the wires from $(\neg)p$ as `out` and all other internal wires of B as `core`. We first need to partition the rest of the circuit into blocks and ensure that the two `out` wires are `in` wires to the same block in the rest of C.

If we substitute $x_i = \alpha$ to fix h and rewrite, we see that x_j's successors are replaced by $(\neg)p$'s successors and that the three costly gates in B have been eliminated. Therefore the circuit computes $(\neg)\mathsf{XOR}_{k-1}$ and is optimal. Applying the inductive hypothesis, we can partition the remaining circuit into blocks which we lift back to the original. Notice that x_j's new successors are in the same block and therefore in C, $(\neg)p$'s successors are also in the same block as desired.

It remains to prove that B computes $(\neg)\mathsf{XOR}_2$. If we substitute $x_i = \alpha$ to fix h and rewrite as above we see that B reduces to $(\neg)x_j$, i.e. $B(\alpha, x_j) = (\neg)x_j$. We argue that if we instead substitute $x_i = 1 - \alpha$ that B would also $(\neg)x_j$. It cannot reduce to a constant as otherwise C, which now computes XOR_{k-1} does not depend on x_j, contradicting Fact 3. We also observe B cannot reduce to x_j in both cases (or $\neg x_j$ in both cases), as else we could replace B with x_j (or $\neg x_j$) and C would still correctly compute XOR_n with three fewer gates — contradicting that C is optimal. Therefore $B(1 - \alpha, x_j) = \neg B(\alpha, x_j)$ and the only binary Boolean functions that satisfy these two equations are XOR_2 and $\neg\mathsf{XOR}_2$ as desired. □

4.3 Optimal $(\neg)\mathsf{XOR}$ Circuit Identity Testing

In our study of the XOR-function, a natural decision problem for identity testing a circuit arises. Namely,

- Input: a normalized circuit C computing a Boolean function $f : \{0,1\}^n \to \{0,1\}$

- Question: is C an optimal circuit compute XOR_n?

We will show that this problem is *easy*. The naive brute-force solution, where we evaluate the circuit over all 2^n inputs, takes time exponential in n. Building upon Theorem 11 we can improve this to linear time. First, we need the converse of Theorem 11: binary trees of $n-1(\neg)\mathsf{XOR}_2$ blocks always compute $(\neg)\mathsf{XOR}_n$. Formally,

Claim 15. *Let C be a circuit of size $3(n-1)$ and let $f : \{0,1\}^n \to \{0,1\}$ be the Boolean function it computes. If C can be partitioned into $n-1$ blocks as described in Theorem 11 each then $f \equiv (\neg)\mathsf{XOR}_n$.*

The proof is a straightforward strong induction, albeit with a lengthy case analysis. The proof can be found in Appendix D. Together, Theorem 11 and Claim 15 provide a simple algorithm to determine whether C computes $(\neg)\mathsf{XOR}_2$: partition C into blocks and verify that each computes $(\neg)\mathsf{XOR}_2$. We then only need to perform one evaluation to differentiate which parity function C computes.

Corollary 16. *Given a normalized circuit C computing a Boolean function $f : \{0,1\}^n \to \{0,1\}$, deciding whether C computes XOR_n can be computed in time $O(n)$.*

Proof. We first verify $|C| = 3(n-1)$ by counting the number of $\wedge$ and $\vee$ gates, **rejecting** if not. We then topologically sort the the gates in C and partition the circuit into $n-1$ blocks, each of size 3, as described in the proof of Theorem 11. We then must verify that each block computes $(\neg)\mathsf{XOR}_2$. Since there are a finite number of normalized optimal $(\neg)\mathsf{XOR}_2$ blocks, we simply hardcode a list of them in our algorithm and compare the blocks against this list. If any block does not computes $(\neg)\mathsf{XOR}_2$ then we **reject**. Lastly, we evaluate C on the all 0 input: if it evaluates to 0 then C computes XOR_n rather than $\neg\mathsf{XOR}_n$ and we **accept**.

Since C is normalized each of the above steps runs in $O(n)$ time. There are at most $2 \cdot 3(n-1)+1 \leq 6n$ $\neg$ gates in the circuit (assuming each costly gate is fed by two $\neg$ gates and the output is negated), so at most $9n$ gates total in the circuit. In our circuit representation, each input and node is joined by exactly one node. Therefore topologically sorting takes $O(n)$ time. Since our hardcoded list of $(\neg)\mathsf{XOR}_2$ is of fixed size, comparing each individual block against the list takes $O(1)$ time for a total for $O(n)$ since there are $n-1$ blocks. Lastly evaluating the circuit takes $O(n)$ time.

Correctness follows from Theorem 11 and Claim 15 along with the fact $\mathsf{XOR}_n(\vec{0}) = 0 \neq \neg\mathsf{XOR}_n(\vec{0})$.

□

Remark 17. *To test whether C computes $\neg\mathsf{XOR}_n$, we can use the same proposed procedure, but with the answers flipped in the last step.*

References

[AHK93] Dana Angluin, Lisa Hellerstein, and Marek Karpinski. "Learning Read-Once Formulas with Queries". In: *J. ACM* 40.1 (1993), pp. 185–210. DOI: 10.1145/138027.138061. URL: https://doi.org/10.1145/138027.138061.

[And87] Alexander E Andreev. "On a method for obtaining more than quadratic effective lower bounds for the complexity of π-schemes". In: *Moscow Univ. Math. Bull.* 42.1 (1987), pp. 63–66.

[BCKT94] Nader H. Bshouty, Richard Cleve, Sampath Kannan, and Christino Tamon. "Oracles and queries that are sufficient for exact learning (extended abstract)". In: *Proceedings of the Seventh Annual Conference on Computational Learning Theory*. COLT '94. New Brunswick, New Jersey, USA: Association for Computing Machinery, 1994, pp. 130–139. ISBN: 0897916557. DOI: 10.1145/180139.181067. URL: https://doi.org/10.1145/180139.181067.

[BCOST14] Eric Blais, Clément L Canonne, Igor C Oliveira, Rocco A Servedio, and Li-Yang Tan. "Learning circuits with few negations". In: *arXiv preprint arXiv:1410.8420* (2014).

[BHH95] Nader H. Bshouty, Thomas R. Hancock, and Lisa Hellerstein. "Learning Boolean Read-Once Formulas over Generalized Bases". In: *J. Comput. Syst. Sci.* 50.3 (1995), pp. 521–542. DOI: 10.1006/jcss.1995.1042. URL: https://doi.org/10.1006/jcss.1995.1042.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998. ISBN: 978-0-521-45520-6.

[CKLM20] Mahdi Cheraghchi, Valentine Kabanets, Zhenjian Lu, and Dimitrios Myrisiotis. "Circuit lower bounds for MCSP from local pseudorandom generators". In: *ACM Transactions on Computation Theory (TOCT)* 12.3 (2020), pp. 1–27.

[DK11] Evgeny Demenkov and Alexander S. Kulikov. "An Elementary Proof of a 3n - o(n) Lower Bound on the Circuit Complexity of Affine Dispersers". In: *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*. Ed. by Filip Murlak and Piotr Sankowski. Vol. 6907. Lecture Notes in Computer Science. Springer, 2011, pp. 256–265. DOI: 10.1007/978-3-642-22993-0_25. URL: https://doi.org/10.1007/978-3-642-22993-0%5C_25.

[FGHK16] Magnus Gausdal Find, Alexander Golovnev, Edward A. Hirsch, and Alexander S. Kulikov. "A Better-Than-3n Lower Bound for the Circuit Complexity of an Explicit Function". In: *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. 2016, pp. 89–98. DOI: 10.1109/FOCS.2016.19.

[GHKK18] Alexander Golovnev, Edward A. Hirsch, Alexander Knop, and Alexander S. Kulikov. "On the limits of gate elimination". In: *J. Comput. Syst. Sci.* 96 (2018), pp. 107–119. DOI: 10.1016/j.jcss.2018.04.005. URL: https://doi.org/10.1016/j.jcss.2018.04.005.

[GIIKKT19] Alexander Golovnev, Rahul Ilango, Russell Impagliazzo, Valentine Kabanets, Antonina Kolokolova, and Avishay Tal. "AC0 [p] lower bounds against MCSP via the coin problem". In: *ICALP*. 2019.

[GJ79] Michael R Garey and David S Johnson. "Computers and intractability". In: *A Guide to the* (1979).

[GKW21] Alexander Golovnev, Alexander S. Kulikov, and R. Ryan Williams. "Circuit Depth Reductions". In: *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*. Ed. by James R. Lee. Vol. 185. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 24:1–24:20. DOI: 10.4230/LIPIcs.ITCS.2021.24. URL: https://doi.org/10.4230/LIPIcs.ITCS.2021.24.

[GMR06] Martin Charles Golumbic, Aviad Mintz, and Udi Rotics. "Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees". In: *Discrete Applied Mathematics* 154.10 (2006), pp. 1465–1477. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2005.09.016. URL: https://www.sciencedirect.com/science/article/pii/S0166218X06000072.

[Hir22] Shuichi Hirahara. "NP-Hardness of Learning Programs and Partial MCSP". In: *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. IEEE, 2022, pp. 968–979. DOI: 10.1109/FOCS54457.2022.00095. URL: https://doi.org/10.1109/FOCS54457.2022.00095.

[HIR23] Yizhi Huang, Rahul Ilango, and Hanlin Ren. "NP-Hardness of Approximating Meta-Complexity: A Cryptographic Approach". In: *Cryptology ePrint Archive* (2023).

[HOS18] Shuichi Hirahara, Igor Carboni Oliveira, and Rahul Santhanam. "NP-hardness of minimum circuit size problem for OR-AND-MOD circuits". In: (2018).

[HPRW96] Lisa Hellerstein, Krishnan Pillaipakkamnatt, Vijay Raghavan, and Dawn Wilkins. "How many queries are needed to learn?" In: *J. ACM* 43.5 (Sept. 1996), pp. 840–862. ISSN: 0004-5411. DOI: 10.1145/234752.234755. URL: https://doi.org/10.1145/234752.234755.

[HS17] Shuichi Hirahara and Rahul Santhanam. "On the average-case complexity of MCSP and its variants". In: *32nd Computational Complexity Conference (CCC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2017.

[Hue81] Gérard Huet. "A complete proof of correctness of the Knuth-Bendix completion algorithm". In: *Journal of Computer and System Sciences* 23.1 (1981), pp. 11–21. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(81)90002-7. URL: https://www.sciencedirect.com/science/article/pii/0022000081900027.

[Ila20] Rahul Ilango. "Constant Depth Formula and Partial Function Versions of MCSP Are Hard". In: *SIAM Journal on Computing* 0.0 (2020), FOCS20-317-FOCS20–367. DOI: 10.1137/20M1383562. eprint: https://doi.org/10.1137/20M1383562. URL: https://doi.org/10.1137/20M1383562.

[ILMR02] Kazuo Iwama, Oded Lachish, Hiroki Morizumi, and Ran Raz. "An Explicit Lower Bound of 5n - o(n) for Boolean Circuits". In: (2002).

[ILO20] Rahul Ilango, Bruno Loff, and Igor Carboni Oliveira. "NP-Hardness of Circuit Minimization for Multi-Output Functions." In: *Electronic Colloquium on Computational Complexity (ECCC)*. Vol. 27. 2020, p. 21.

[IM02] Kazuo Iwama and Hiroki Morizumi. "An Explicit Lower Bound of 5n - o(n) for Boolean Circuits". In: *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*. Ed. by Krzysztof Diks and Wojciech Rytter. Vol. 2420. Lecture Notes in Computer Science. Springer, 2002, pp. 353–364. DOI: 10.1007/3-540-45687-2_29. URL: https://doi.org/10.1007/3-540-45687-2%5C_29.

[IN93] Russell Impagliazzo and Noam Nisan. "The effect of random restrictions on formula size". In: *Random Structures & Algorithms* 4.2 (1993), pp. 121–133.

[Juk+12] Stasys Jukna et al. *Boolean function complexity: advances and frontiers*. Vol. 5. Springer, 2012.

[KB70] DONALD E. KNUTH and PETER B. BENDIX. "Simple Word Problems in Universal Algebras††The work reported in this paper was supported in part by the U.S. Office of Naval Research." In: *Computational Problems in Abstract Algebra*. Ed. by JOHN LEECH. Pergamon, 1970, pp. 263–297. ISBN: 978-0-08-012975-4. DOI: https://doi.org/10.1016/B978-0-08-012975-4.50028-X. URL: https://www.sciencedirect.com/science/article/pii/B978008012975450028X.

[KC00] Valentine Kabanets and Jin-yi Cai. "Circuit minimization problem". In: *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*. Ed. by F. Frances Yao and Eugene M. Luks. ACM, 2000, pp. 73–79. DOI: 10.1145/335305.335314. URL: https://doi.org/10.1145/335305.335314.

[KI03] Valentine Kabanets and Russell Impagliazzo. "Derandomizing polynomial identity tests means proving circuit lower bounds". In: *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. 2003, pp. 355–364.

[Kom11] Yu A Kombarov. "The minimal circuits for linear Boolean functions". In: *Moscow University Mathematics Bulletin* 66.6 (2011), pp. 260–263.

[Kom18] Yu A Kombarov. "Complexity and Structure of Circuits for Parity Functions". In: *Journal of Mathematical Sciences* 233 (2018), pp. 95–99.

[Kom22] YA Kombarov. "Circuit complexity lower bound for parity function in one infinite basis". In: *Mathematical Problems of Cybernetics* 20 (2022), pp. 81–118.

[Lam12] Leslie Lamport. "How to write a 21 st century proof". In: *Journal of fixed point theory and applications* 11 (2012), pp. 43–63.

[Lam95] Leslie Lamport. "How to write a proof". In: *The American mathematical monthly* 102.7 (1995), pp. 600–608.

[LMS18] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. "Slightly Superexponential Parameterized Problems". In: *SIAM Journal on Computing* 47.3 (2018), pp. 675–702. DOI: 10.1137/16M1104834. eprint: https://doi.org/10.1137/16M1104834. URL: https://doi.org/10.1137/16M1104834.

[LR01] Oded Lachish and Ran Raz. "Explicit lower bound of 4.5n - o(n) for boolena circuits". In: *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*. STOC '01. Hersonissos, Greece: Association for Computing Machinery, 2001, pp. 399–408. ISBN: 1581133499. DOI: 10.1145/380752.380832. URL: https://doi.org/10.1145/380752.380832.

[LY22] Jiatu Li and Tianqi Yang. "3.1n - $o(n)$ circuit lower bounds for explicit functions". In: *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*. Ed. by Stefano Leonardi and Anupam Gupta. ACM, 2022, pp. 1180–1193. DOI: 10.1145/3519935.3519976. URL: https://doi.org/10.1145/3519935.3519976.

[Mas79] William J Masek. "Some NP-complete set covering problems". In: *Unpublished manuscript* (1979).

[Pla93] David A. Plaisted. "Equational reasoning and term rewriting systems". In: *Handbook of Logic in Artificial Intelligence and Logic Programming (Vol. 1)*. USA: Oxford University Press, Inc., 1993, pp. 274–364. ISBN: 019853745X.

[Plu99] Detlef Plump. "Term graph rewriting". In: *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 2: Applications, Languages and Tools* (1999), pp. 3–61.

[PZ93] Michael S Paterson and Uri Zwick. "Shrinkage of de Morgan formulae under restriction". In: *Random Structures & Algorithms* 4.2 (1993), pp. 135–150.

[Red20] Nikolay P Redkin. "The generalized complexity of linear Boolean functions". In: *Discrete Mathematics and Applications* 30.1 (2020), pp. 39–44.

[Red73] NP Red'kin. "Proof of minimality of circuits consisting of functional elements". In: *Systems Theory Research: Problemy Kibernetiki* (1973), pp. 85–103.

[RS21] Hanlin Ren and Rahul Santhanam. "Hardness of KT characterizes parallel cryptography". In: *Cryptology ePrint Archive* (2021).

[San20] Rahul Santhanam. "Pseudorandomness and the minimum circuit size problem". In: *LIPIcs* 151 (2020).

[Sch74] Claus-Peter Schnorr. "Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen". In: *Computing* 13.2 (1974), pp. 155–171. DOI: 10.1007/BF02246615. URL: https://doi.org/10.1007/BF02246615.

[Sha49] Claude. E. Shannon. "The synthesis of two-terminal switching circuits". In: *The Bell System Technical Journal* 28.1 (1949), pp. 59–98. DOI: 10.1002/j.1538-7305.1949.tb03624.x.

[sta98] Johan HÅ stad. "The shrinkage exponent of de Morgan formulas is 2". In: *SIAM Journal on Computing* 27.1 (1998), pp. 48–64.

[Sto77] Larry J. Stockmeyer. "On the Combinational Complexity of Certain Symmetric Boolean Functions". In: *Math. Syst. Theory* 10 (1977), pp. 323–336. DOI: 10.1007/BF01683282. URL: https://doi.org/10.1007/BF01683282.

[SZ12] Thomas Sternagel and Harald Zankl. "KBCV - Knuth-Bendix Completion Visualizer". In: *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*. Ed. by Bernhard Gramlich, Dale Miller, and Uli Sattler. Vol. 7364. Lecture Notes in Computer Science. Springer, 2012, pp. 530–536. DOI: 10.1007/978-3-642-31365-3_41. URL: https://doi.org/10.1007/978-3-642-31365-3%5C_41.

[Tra84] B. A. Trakhtenbrot. "A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms". In: *Annals of the History of Computing* 6.4 (1984), pp. 384–400. DOI: 10.1109/MAHC.1984.10036.

[Weg87] Ingo Wegener. *The Complexity of Boolean Functions*. Wiley-Teubner, 1987.

[WHU23] Brae J. Webb, Ian J. Hayes, and Mark Utting. "Verifying Term Graph Optimizations using Isabelle/HOL". In: *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023*. Ed. by Robbert Krebbers, Dmitriy Traytel, Brigitte Pientka, and Steve Zdancewic. ACM, 2023, pp. 320–333. DOI: 10.1145/3573105.3575673. URL: https://doi.org/10.1145/3573105.3575673.

[Wil10] Ryan Williams. "Improving exhaustive search implies superpolynomial lower bounds". In: *Proceedings of the forty-second ACM symposium on Theory of computing*. 2010, pp. 231–240.

A Structured Proof of Schnorr

Below, we prove Theorem 8 with a structured proof as proposed by Lamport [Lam95; Lam12]. We believe that this presentation style makes the intricate case analysis present both in this proof and that of Theorem 11 more explicit and readable. Furthermore, this style of proof is more amenable to verification using a computer. As there has been recent work which formalizes term graph rewriting for use with proof assistants [WHU23], it may be of independent interest to formally verify our proofs of Schnorr and Theorem 11.

Structured Proof of Theorem 8. Let C be an optimal circuit for XOR_n where $n \geq 2$.

1. Let h be the *first* AND,OR gate of C in topological order, so h has $(\neg)x_i, (\neg)x_j$ as inputs for $i, j \in \mathbb{N}$.
2. $i \neq j$
   (a) Suppose not, so $i = j$
   (b) Then $h \equiv (\neg)x_i \diamond (\neg)x_i$ where $\diamond \in \{\wedge, \vee\}$
   (c) Thus, one of the normalizing or tautology rules from Gate Elim TGRS matches h
   (d) **Rewrite** C finding h deleted
   (e) Contradiction to optimality of C
3. The fanout of x_i must be at least 2.
   (a) Suppose not, so fanout of x_i is 1.
   (b) **Substitute** $x_j = \alpha$ in C to fix h
   (c) **Rewrite** C, finding that fanout of x_i is now 0
   (d) Thus, $C\,|_{x_j=\alpha}$ does not depend on x_i
   (e) Contradiction to Fact 3.
4. Let f be the other gate taking x_i as input so $f \neq h$
5. $(\neg)f$ is not the output gate of C.
   (a) Suppose it is, so $(\neg)f$ is the output gate of C.
   (b) **Substitute** $x_i = \alpha$ in C to fix f
   (c) **Rewrite** C, finding that output of C is constant
   (d) Thus, $C\,|_{x_i=\alpha}$ is a constant function
   (e) Contradiction to Fact 2.

6. Let f' be a *costly* successor of f in C, such must exist.

7. Eliminate three distinct gates with a substitution.

   (a) **Substitute** $x_i = \alpha$ in C to fix f

   (b) **Rewrite** C, finding at least f, h, f' deleted

   (c) *argument:* Observe that x_i "touches" gate h to eliminate, and it fixes f which "touches" gate f'

   (d) there exists a 1-bit restriction eliminating ≥ 3 gates

8. Conclude XOR_n requires at least 3 more costly gates than XOR_{n-1}.

9. Observe XOR_1 requires 0 costly gates.

10. Use induction to show XOR_n requires at least $3(n-1)$.

□

B Detailed Term Graph Rewriting Demo

In this section, we show how the Term Graph Rewriting system we defined in Section 2.1 works on a small example $\neg x_i \wedge x_j$.

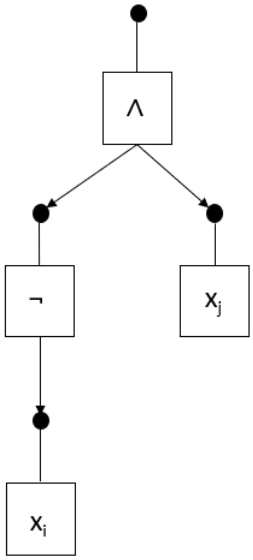


Figure 7: The Term Graph for $\neg x_i \wedge x_j$

Assume we substitute $x_j = 0$, and we then identify the rule $0 \to \neg 1$

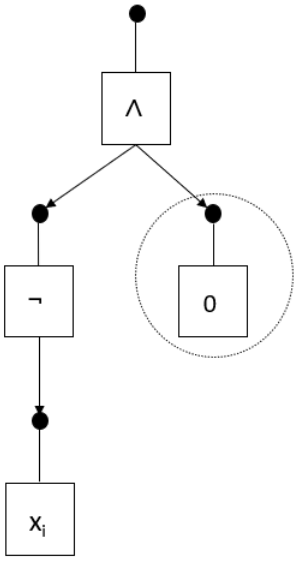


Figure 8: The Term Graph after substituting $x_j = 0$

Replace 0 with $\neg 1$ by adjusting the nodes and edges accordingly

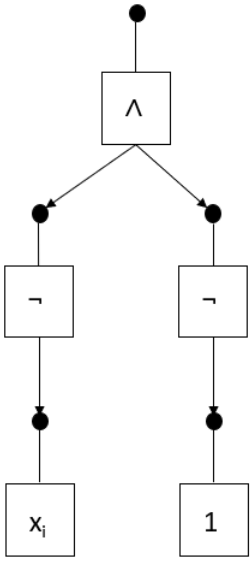


Figure 9: Identify $0 \to \neg 1$

Identify the rule $g \wedge \neg 1 \to \neg 1$

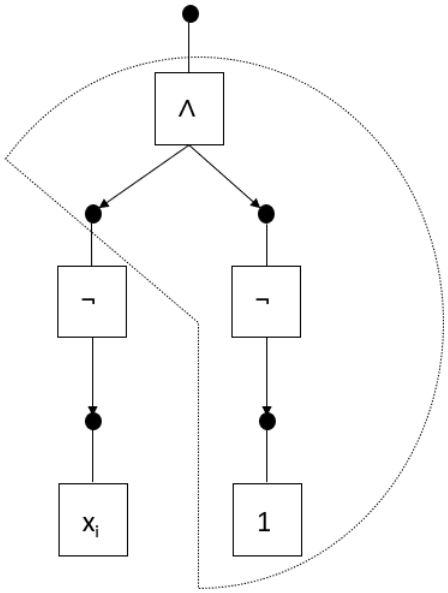


Figure 10: Rewrite 0 with $\neg 1$

Replace $g \wedge \neg 1 \to \neg 1$ with $\neg 1$ by adjusting the nodes and edges accordingly. We notice that $\neg x_i$ is now disconnected from the term graph.

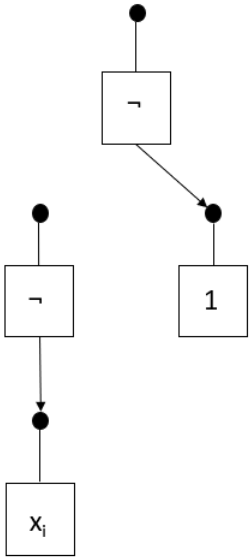


Figure 11: Rewrite $g \wedge \neg 1$ with $\neg 1$

Lastly, we clean up the disconnected part and obtain the end result of the term graph after the rewriting process

Figure 12: The end result of the term graph $\neg 1$

C Structured Proof of Theorem 11

Below, we prove Theorem 11 (our Main Theorem) with a structured proof as proposed by Lamport [Lam95; Lam12]. We believe that this presentation style makes the intricate case analysis present both in our proof of Schnorr and this proof more explicit and readable. Furthermore, this style of proof is more amenable to verification using a computer. As there has been recent work which formalizes term graph rewriting for use with proof assistants [WHU23], it may be of independent interest to formally verify our proofs of Schnorr and of our Main Theorem.

The bulk of the work is proving the inductive step. In the main body of the paper this is Lemma 14 which roughly corresponds to steps 1 - 17(f) of the inductive steps below.

Proof. Base Case:

1. Trivially holds for $k = 1$ and 2.

Assume Inductive Hypothesis for some $k - 1 \geq 1$.
Inductive Step:

1. Let C be an optimal circuit for XOR_k.
2. Let h be the *first* AND, OR gate of C in topological order, so h has $(\neg)x_i, (\neg)x_j$ as inputs for $i, j \in \mathbb{N}$.
3. $i \neq j$
   (a) Suppose not, so $i = j$
   (b) Then $h \equiv (\neg)x_i \diamond (\neg)x_i$ where $\diamond \in \{\wedge, \vee\}$
   (c) Thus, one of the normalizing or tautology rules from Gate Elim TRS matches h
   (d) **Rewrite** C finding h deleted
   (e) Contradiction to optimality of C
4. The fanout of x_i and x_j must be 2 by Lemma 13.
   (a) Let h'_i be the other costly successor of x_i and let f'_i be the other input or costly gate whose successor is h'_i.
   (b) Let h'_j be the other costly successor of x_j and let f'_j be the other input or costly gate whose successor is h'_j.
5. h is not the output of the circuit.

(a) Suppose it is, so $(\neg)h$ is the output gate of C.

(b) **Substitute** $x_i = \alpha$ in C to fix h

(c) **Rewrite** C, finding that output of C is constant

(d) Thus, $C\,|_{x_i=\alpha}$ is a constant function

(e) Contradict Corollary 12.

6. Let p be a *costly* successor of h in C and let q be the other input or costly gate whose successor is p.

7. Argue symmetrically that h_i' is not the output of the circuit. Let r be a successor of h_i' and let s be the other input or costly gate whose successor is p.

8. r is the only successor of h_i'.

   (a) Suppose not, so $\exists r' \neq r$ that is a costly successor of h_i'.

   (b) **Substitute** $x_i = \beta$ to fix h_i'

   (c) **Rewrite** C to find h_i', h, r, r' deleted

   (d) Contradict Corollary 12

9. Symmetrically h_j' is not the output of the circuit and has exactly one costly successor u. Let v be the other input or costly gate whose successor is u.

10. p is the only successor of h

    (a) Suppose not, so $\exists p' \neq p$ that is a costly successor of h.

    (b) We perform a case analysis depending on the the identities of p, p', h_i' and h_j' (i.e. depending on $p \stackrel{?}{=} h_i', p \stackrel{?}{=} h_j', p' \stackrel{?}{=} h_j'$ and $h_i' \stackrel{?}{=} h_j'$). Each possible set of equalities falls into one of the three cases below or is impossible, e.g. it is impossible for $p = h_i'$, $p' = h_j'$ and $h_i' = h_j'$ since this implies $p = p'$. A complete decision tree can be found in Figure 13.

       i. Case 1: p, p' are distinct from h_i' and h_j'.

          A. **Substitute** $x_i = \alpha$ to fix h.

          B. **Rewrite** C to find h_i', h, p, p' are removed.

          C. Contradict Corollary 12.

       ii. Case 2: Exactly one of p, p' is equal to h_i' or h_j'

          A. Without loss of generality assume $p' = h_j'$ and $p \neq h_i'$.

          B. **Substitute** $x_i = \alpha$ to fix h.

C. **Rewrite** C to find h'_i, h, p, p' removed.

D. Contradict Corollary 12.

iii. Case 3: p, p' are equal to h'_i and h'_j.

A. **Substitute** $x_i = \alpha$ to fix h.

B. **Rewrite** C to find h'_i, h, h'_j and s removed.

C. *argument:* Observe that after fixing h, both inputs to h'_i are constants. After removing h'_i a constant will be touching s so it will be removed.

D. *argument:* s is distinct from h'_j since h'_i is not an input to h'_j—h and x_i are in this case.

E. Contradict Corollary 12.

11. $h'_i = h'_j$

(a) Suppose not, i.e. $h'_i \neq h'_j$.

(b) p is not equal to at least one of h'_i or h'_j; without loss of generality assume $p \neq h'_i$.

(c) **Substitute** $x_i = \alpha$ in C to fix h

(d) **Rewrite** C, finding h, h'_i and p are removed.

(e) No other costly gates are removed.

(f) *argument:* otherwise this violates Corollary 12.

(g) C is now a normalized optimal circuit computing $(\neg)\mathsf{XOR}_{k-1}$,

(h) *argument:* rewriting guarantees normalization, and $|C|$ is now $3(k-1)-3 = 3(k-2)$ so it is optimal.

(i) x_j has only one costly successor in C:

(j) *argument:* We will carefully step through rewriting.

i. Remove h via a fixing rule and find x_j now has only one successor: h'_j.

ii. Remove h'_i via a passing rule and see x_j does not gain any successors since $h'_i \neq h'_j$

iii. Remove p and see there are two cases: $p = h'_j$ and $p \neq h'_j$.

A. Remove $p = h'_j$ via a passing rule and find x_j still has one successor since $p = h'_j$ has one successor.

B. Remove $p \neq h'_j$ and find x_j still has one successor, h'_j since $p \neq h'_j$.

iv. In both cases, conclude x_j has one costly successor.

(k) Contradict Lemma 13 since C is optimal.

12. Define $h' := h'_i = h'_j$ and $p' := r = u$.

13. $p = p'$.

    (a) Suppose otherwise, $p \neq p'$

    (b) **Substitute** $x_i = \alpha$ to fix h.

    (c) **Rewrite** C to see h and p have been removed and that there are two possibilities:

        i. h is removed via a fixing rule

            A. Find p' is also removed

            B. Contradict Corollary 12 since h, h', p, p' have been removed.

        ii. h is removed via a passing rule

            A. Find x_j inherits p' as it's only successor.

            B. Find C has three fewer costly gates and thus is a normalized optimal $(\neg)\mathsf{XOR}_{n-1}$ circuit

            C. Contradict Lemma 13

    (d) In both cases reach a contradiction.

14. Define B to be the block consisting of $h, h', (\neg)p$ and label the wires from $(\neg)x_i$ and $(\neg)x_j$ as `in`, the wires from $(\neg)p$ as `out`, and all others as `core`.

15. *clarification:* Here *wires* refers to the unique resultant nodes whose label is $(\neg)x_i$, $(\neg)x_j$, and $(\neg)p$ respectively.

16. *clarification:* We label $\neg x_i$, $\neg x_j$, and $\neg p$ as `in` or `out` if x_i, x_j or p only feed into $\neg x_i$, $\neg x_j$ or $\neg p$ and no other gates.

17. We can properly partition the rest of C into blocks.

    (a) **Substitute** $x_i = \alpha$ to fix h.

    (b) **Rewrite** C to find x_j successors replaced by p's successors.

    (c) *argument:* only h, h', p are removed by Corollary 12

    (d) C is an optimal circuit for $(\neg)\mathsf{XOR}_{k-1}$ since three gates were removed.

    (e) Apply the IH to partition C into $k-2$ $(\neg)\mathsf{XOR}_2$ blocks.

    (f) x_j must have exactly two costly successors by Lemma 13 and therefore p must have exactly two costly successors.

    (g) By IH, x_j's successors are in the same $(\neg)\mathsf{XOR}_2$ block.

(h) Lift the partition back to the original C.

18. B computes $(\neg)\mathsf{XOR}_2$

(a) **Substitute** $x_i = \alpha$ to fix h.

(b) **Rewrite** C.

(c) *argument*: Find B reduces to $(\neg)x_j$.

i. Suppose it reduced to a constant.

ii. C no longer depends on x_j.

iii. Contradict Fact 3.

(d) **Substitute** $x_i = 1 - \alpha$ to fix h.

(e) **Rewrite** C to find B reduces to $(\neg)x_j$ as above.

(f) *argument: C does not reduce the same way in both cases*

i. Suppose otherwise. Without loss of generality it reduces to x_j.

ii. Replace B in C with x_j and find C still computes XOR_n even though it doesn't depend on x_i.

iii. Contradict Fact 3

(g) *argument:* Find B computes $(\neg)\mathsf{XOR}_2$ since $B(0, x_j) = (\neg)x_j$ and $B(1, x_j) = \neg B(0, x_j)$.

□

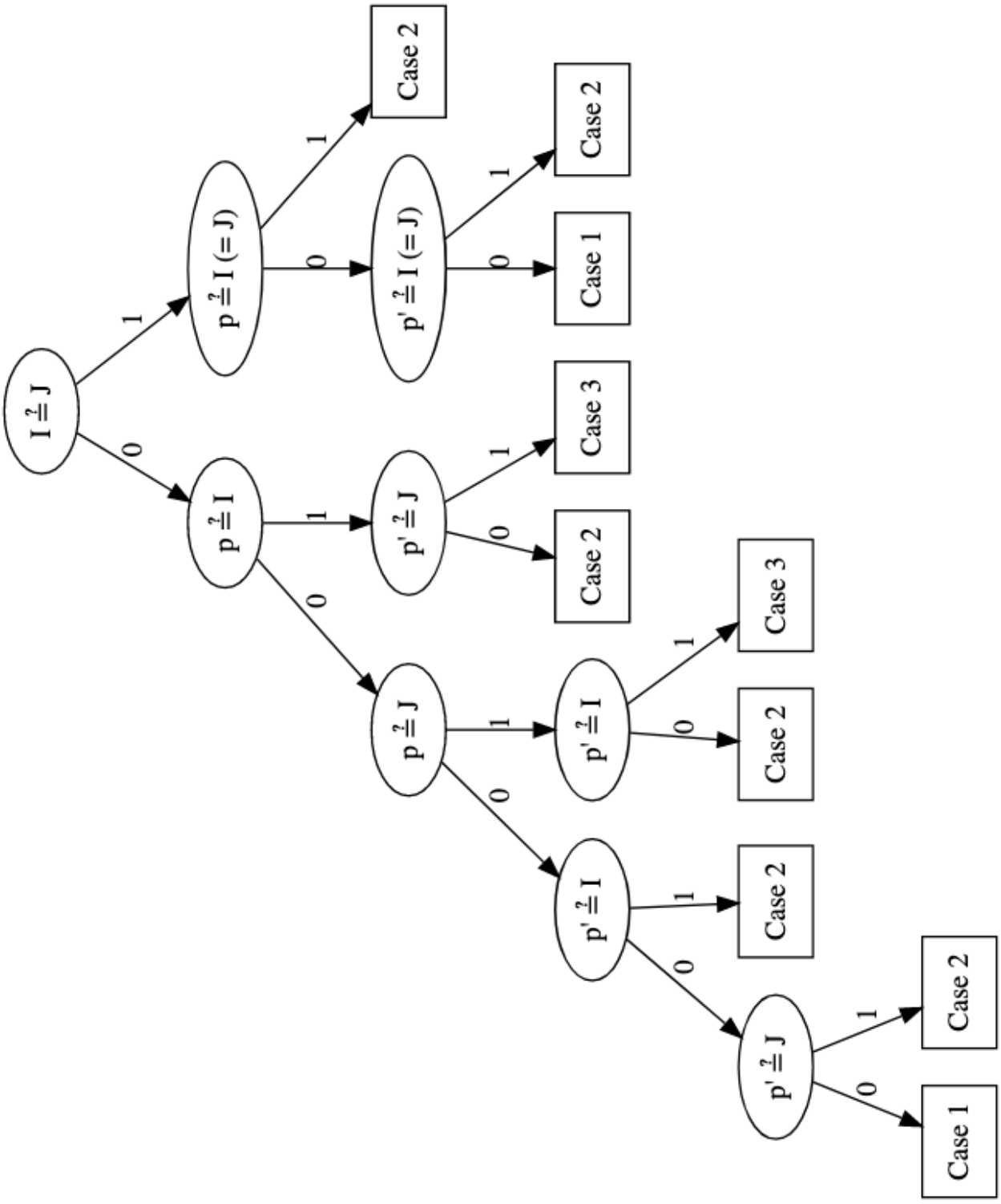


Figure 13: A decision tree for Step 10 in the Structured Proof of Theorem 11. For readability, we define $I = h'_i$ and $J = h'_j$. Notice each branch bottoms out when all remaining potential equalities can be determined using substitution as well as transitivity and symmetry of equality. This prunes the depth of some branches since certain systems of equalities (e.g. $p = h'_i, h'_i = h'_j, p \neq h'_j$) are unsatisfiable.

D Proof of Claim 15

We will prove the claim using strong induction. When $n = 1$, the claim is trivial since the only optimal normalized $(\neg)\mathsf{XOR}_1$ circuit is $(\neg)x_1$. When $n = 2$, the entire circuit is itself an optimal normalized $(\neg)\mathsf{XOR}_2$ block.

Now, assume the claim is true for $n = 1, 2, \ldots, k-1$ for some $k - 1 \geq 2$. Namely, any normalized circuit C that can be partitioned into $0, 1, \ldots, k-2$ blocks each computing $(\neg)\mathsf{XOR}_2$ computes either XOR or $\neg\mathsf{XOR}$ on its inputs. We show the claim holds when $n = k$. Let C be a normalized circuit that can be partitioned into $k - 1$ blocks that each compute $(\neg)\mathsf{XOR}_2$ and let $f : \{0,1\}^k \to \{0,1\}$ be the function C computes.

We take the root block (i.e. the highest one in the topological order of the blocks) and look at its left and right sub-circuits of blocks. Let L, R be the left and right sub-circuits respectively and define f_L, f_R to be the two functions they compute. Notice C computes $f \equiv (\neg)\mathsf{XOR}_2(f_L, f_R)$. Furthermore, let X_L, X_R be the sets of input variables of f_L, f_R respectively, and $|X_L| = s$, $|X_R| = t$. Lastly define the set of all input variables X and observe $X = X_L \cup X_R$ and $|X| = s + t = k$.

By the inductive hypothesis, we know that $f_L(X_L) = (\neg)\mathsf{XOR}_s(X_L)$ and $f_R(X_R) = (\neg)\mathsf{XOR}_t(X_R)$. We perform case analysis to argue that $\mathsf{XOR}_2(f_L, f_R)$ computes the parity of the set of all input variables X for all possible combinations of f_L and f_R. The full case analysis can be found in Table 1 below, but in short

1. when $f_L \equiv \mathsf{XOR}_s$ and $f_R \equiv \mathsf{XOR}_t$, C computes XOR_k,
2. when $f_L \equiv \mathsf{XOR}_s$ and $f_R \equiv \neg\mathsf{XOR}_t$, C computes $\neg\mathsf{XOR}_k$.
3. when $f_L \equiv \neg\mathsf{XOR}_s$ and $f_R \equiv \mathsf{XOR}_t$, C computes $\neg\mathsf{XOR}_k$.
4. and when $f_L \equiv \neg\mathsf{XOR}_s$ and $f_R \equiv \neg\mathsf{XOR}_t$, C computes XOR_k.

In all cases, $C(X)$ either computes $\mathsf{XOR}_k(X)$ or $\neg\mathsf{XOR}_k(X)$ as desired.

The analysis for $\neg\mathsf{XOR}_2(f_L, f_R)$ follows symmetrically with the bits flipped. By the principle of mathematical induction, the claim is true for all $n \geq 1$.

	No. of 1's in X_L and X_R	$f_L(X_L)$, $f_R(X_R)$	$\mathsf{XOR}_2(f_L, f_R)$
$f_L \equiv \mathsf{XOR}_s$ $f_R \equiv \mathsf{XOR}_t$	X_L has even many 1's X_R has even many 1's	$f_L(X_L) = 0$ $f_R(X_R) = 0$	0
	X_L has even many 1's X_R has odd many1's	$f_L(X_L) = 0$ $f_R(X_R) = 1$	1
	X_L has odd many 1's X_R has even many 1's	$f_L(X_L) = 1$ $f_R(X_R) = 0$	1
	X_L has odd many 1's X_R has odd many 1's	$f_L(X_L) = 0$ $f_R(X_R) = 0$	0
$f_L \equiv \mathsf{XOR}_s$ $f_R \equiv \neg\mathsf{XOR}_t$	X_L has even many 1's X_R has even many 1's	$f_L(X_L) = 0$ $f_R(x_R) = 1$	1
	X_L has even many 1's X_R has odd many 1's	$f_L(X_L) = 0$ $f_R(x_R) = 0$	0
	X_L has odd many 1's X_R has even many 1's	$f_L(X_L) = 1$ $f_R(X_R) = 1$	0
	X_L has odd many 1's X_R has odd many 1's	$f_L(X_L) = 1$ $f_R(X_R) = 0$	1
$f_L \equiv \neg\mathsf{XOR}_s$ $f_R \equiv \mathsf{XOR}_t$	X_L has even many 1's X_R has even many 1's	$f_L(X_L) = 1$ $f_R(X_R) = 0$	1
	X_L has even many 1's X_R has odd many 1's	$f_L(X_L) = 1$ $f_R(X_R) = 1$	0
	X_L has odd many 1's X_R has even many 1's	$f_L(X_L) = 0$ $f_R(X_R) = 0$	0
	X_L has odd many 1's X_R has odd many 1's	$f_L(x_L) = 0$ $f_R(x_R) = 1$	1
$f_L \equiv \neg\mathsf{XOR}_s$ $f_R \equiv \neg\mathsf{XOR}_t$	X_L has even many 1's X_R has even many 1's	$f_L(X_L) = 1$ $f_R(x_R) = 1$	0
	X_L has even many 1's X_R has odd many 1's	$f_L(X_L) = 1$ $f_R(X_R) = 0$	1
	X_L has odd many 1's X_R has even many 1's	$f_L(x_L) = 0$ $f_R(x_R) = 1$	1
	X_L has odd many 1's X_R has even many 1's	$f_L(X_L) = 0$ $f_R(X_R) = 0$	0

Table 1: The case analysis for C computing $\mathsf{XOR}_2(f_L, f_R)$

E Gate Elimination as a Convergent Term Graph Rewriting System

In this section we will formally present gate elimination as a convergent term graph rewriting system, according to the following steps.

1. Identify a list of Boolean identities — $\mathcal{E}_B$ — which are sufficient for gate elimination arguments.
2. Use the Knuth-Bendix algorithm on $\mathcal{E}_B$ to produce a convergent *formula* simplification system $\mathcal{R}_B$.
3. Lift $\mathcal{R}_B$ to a convergent *circuit* simplification system $\mathcal{S}$ via Plump's account of term graph rewriting.

E.1 Boolean Identities

The identities present in $\mathcal{E}_B$ are valid for Boolean algebra and appear in standard gate elimination arguments (Definition 18). That is, for all $g \in \{0, 1\}$, each identity is true when $\approx$ is interpreted as equality on the Boolean domain. Therefore, consequences derived from $\mathcal{E}_B$ via "sound inference rules" are true. We do not treat that *equational logic*[4] formally, because we transform $\mathcal{E}_B$ into a convergent rewriting system in the next subsection.

Definition 18 (Gate Elimination — Useful Identities)**.** We denote by $\mathcal{E}_B$ the following set of identities:

$1 \wedge 1 \approx 1$	$1 \wedge 1 \approx 1$	$\neg 1 \approx 0$	$g \wedge 1 \approx g$	$g \wedge 0 \approx 0$	$g \wedge \neg g \approx 0$	$g \wedge g \approx g$
$1 \wedge 0 \approx 0$	$1 \wedge 0 \approx 1$	$\neg 0 \approx 1$	$1 \wedge g \approx g$	$0 \wedge g \approx 0$	$\neg g \wedge g \approx 0$	$g \vee g \approx g$
$0 \wedge 1 \approx 0$	$0 \wedge 1 \approx 1$	$\neg\neg g \approx g$	$g \vee 0 \approx g$	$g \vee 1 \approx 1$	$g \vee \neg g \approx 1$	
$0 \wedge 0 \approx 0$	$0 \wedge 0 \approx 0$		$0 \vee g \approx g$	$1 \vee g \approx 1$	$\neg g \vee g \approx 1$	
(tt and)	(tt or)	(tt not)	(passing)	(fixing)	(tautology)	(simplify)

There are a few basic Boolean identities that are not present in $\mathcal{E}_B$ such as commutativity of $\wedge$ and $\vee$. We exclude these for two reasons: (1) including them would produce a system that is not *convergent* and (2) these rules do not "simplify" Boolean expressions—their right hand sides do not have fewer Boolean operators. While $\mathcal{E}_B$ is not powerful enough to fully characterize Boolean algebra, it is powerful enough to capture gate elimination arguments with the added benefit that it's resulting system is well-behaved. The identities are available in machine-readable form at this hyperlink.

We will now transform this set of identities into an abstract rewriting system on Boolean formulas.

[4] See Chapter 3 of [BN98] or the exposition of Birkhoff's Theorem in [Pla93].

E.2 Convergent Term Rewriting for Boolean Formulas

An *abstract rewriting system* is just a set of objects A together with a binary relation $\to$ on A called the *rewrite* relation. We are constructing a system where A contains Boolean circuits over the DeMorgan basis and $C \to C'$ holds when C simplifies to C' via a single step of gate elimination. We'll first introduce some terminology about abstract rewriting systems as well as define some desirable properties. For elements $a, a' \in A$, write $a \xrightarrow{*} a'$ to mean that there is a finite path of rewrite steps from a to a', and say that a is *in normal form* if there is no b such that $a \to b$.

Definition 19 (Definition 2.1.3 of [BN98])**.** The rewrite relation $\to$ is called

terminating iff there is no infinite path $a_0 \to a_1 \to \dots$

confluent iff for every triple of objects $a, b, b' \in A$, if $a \xrightarrow{*} b$ and $a \xrightarrow{*} b'$, then there is a $c \in A$ such that both $b \xrightarrow{*} c$ and $b' \xrightarrow{*} c$ —

convergent iff it is both confluent and terminating.

Term rewriting is a classical special case of abstract rewriting systems, rich in motivation from algebra, logic, and programming language theory. In that setting the objects are *terms* — treelike expressions built up from function symbols, constants, and variables. We will take an intermediate step through treelike expressions to get to DAG-like expressions: circuits. Terms in general and DeMorgan formulas in particular are defined below, along with some auxiliary notions required to specify appropriate rewrite relations.

Definition 20 (DeMorgan Formulas as Terms)**.** Let Σ be a finite tuple of function and constant symbols with arities $\vec{d} \in \mathbb{N}^{|\Sigma|}$, and let Z denote an infinite set of variables. $\mathcal{T}(\Sigma, Z)$ denotes the set of all terms over Σ and Z, defined inductively:

- Every variable $z \in Z$ is a term.
- Every application of a function symbol $f_i \in \Sigma$ to d_i terms $t_1, \dots, t_{d_i}$ of the form $f(t_1, \dots, t_{d_i})$ is a term.

$F = \mathcal{T}(B, X)$ where $X = \{g, x_1, x_2, \dots\}$ is the set of DeMorgan formulas.

A *substitution* σ is a mapping between terms that may replace any finite number of variables with another term, but must leave constants and function applications fixed. So, can write substitutions as $\sigma = \{x_i \mapsto t\}$. A *term rewrite rule* is a pair of terms $\langle \ell, r \rangle$ written as $\ell \to r$, such that (1) ℓ is not a variable and (2) the set of variables in r is a subset of the variables in ℓ. A *term rewriting system* over $\mathcal{T}(\Sigma, Z)$ is a set $\mathcal{R}$ of term rewrite rules where all pairs of terms are from $\mathcal{T}(\Sigma, Z)$. Finally, we have:

Definition 21 (Term Rewriting, Definition 4.1 of [Plu99])**.** The rewrite relation $\to_{\mathcal{R}}$ on $\mathcal{T}(\Sigma, Z)$ induced by a term rewriting system $\mathcal{R}$ is defined as follows:
$t \to_{\mathcal{R}} u$ if there is a rule $\ell \to r$ in $\mathcal{R}$ and a substitution σ such that

1. The left-hand side of the rule "matches" t — $\sigma(\ell)$ is a subterm of t
2. The right-hand side "generates" u — u is obtained from t by replacing an occurence of $\sigma(\ell)$ by $\sigma(r)$

We can now give the precise type of $\mathcal{E}_B$: it is a set of pairs of terms from F. We now wish to transform $\mathcal{E}_B$ into a convergent rewriting system $\mathcal{R}_B$. Rather than manually rewriting our equations as term rewrite rules (e.g. $g \wedge 1 \approx g \implies g \wedge 1 \to g$) and then proving convergence from scratch, we use a well known algorithm designed to do just this: the Knuth-Bendix completion algorithm [KB70; Hue81].

Theorem 22 (Knuth-Bendix, [SZ12])**.** *Given as input a set of identities $\mathcal{E}$ over $\mathcal{T}(\Sigma, Z)$, if Knuth-Bendix terminates, it outputs a convergent term rewriting system $\mathcal{R}$ over $\mathcal{T}(\Sigma, Z)$ with the same consequences as $\mathcal{E}$.*

At a high level, the Knuth-Bendix completion algorithm works by ensuring that every pair of rules which overlap, so-called *critical pairs*, do not create ambiguities. If we apply the pair in either order to the same expression, we will get the same final result. Furthermore, the algorithm carefully adds new term rules as well as simplifies rules in order to create a convergent system. Manually running the algorithm in our case would require checking $\binom{25}{2}$ pairs of equations — although not every possible pair overlaps. While this is technically feasible to do by hand, we instead will use the Knuth-Bendix Completion Visualizer (KBCV) which is open-source software implementing the algorithm [SZ12].

Lemma 23. *There is a convergent term rewriting system $\mathcal{R}_B$ for simplification of DeMorgan formulas.*

Proof. We ran Knuth-Bendix on the equations $\mathcal{E}_B$ of Definition 18 using the open-source software Knuth-Bendix Completion Visualizer (KBCV, [SZ12]). The algorithm terminated and printed the TRS $\mathcal{R}_B$ listed in Definition 24 below. We have grouped the rules based on their structure and impact on the circuit. A machine-checkable transcript of the terminating execution is available at this hyperlink for verification. Therefore, $\mathcal{R}_B$ is convergent and has the same consequences as $\mathcal{E}_B$. □

Definition 24 (Term Rewriting System $\mathcal{R}_B$)**.**

$0 \to \neg 1$	$g \wedge \neg 1 \to \neg 1$	$g \wedge 1 \to g$	$g \wedge \neg g \to \neg 1$
$\neg\neg g \to g$	$\neg 1 \wedge g \to \neg 1$	$1 \wedge g \to g$	$\neg g \wedge g \to \neg 1$
$g \wedge g \to g$	$g \vee 1 \to 1$	$g \vee \neg 1 \to g$	$g \vee \neg g \to 1$
$g \vee g \to g$	$1 \vee g \to 1$	$\neg 1 \vee g \to g$	$\neg g \vee g \to 1$
(normalizing)	(fixing)	(passing)	(tautology)

We see that $\mathcal{R}_B$ is a smaller set than the original $\mathcal{E}_B$. Knuth-Bendix has made a few simplifications such as removing redundant tt identities. However, it's one additional rule, $0 \to \neg 1$, stands out. This is the only rewrite rule in the system that increases the number of Boolean operators in the formula. We argue this is a sensible addition for two reasons: (1) the addition of $\neg$ gates in our circuits will be free as they do not count towards the circuit complexity and (2) the expressions become simpler in the sense that after rewriting 0 into $\neg 1$ there is only a single type of constant present. It also does not interfere with the structure of gate elimination arguments in the DeMorgan basis. We can still substitute a variable with 0; it will just need to be replaced first by -1 during rewriting. The term rewrite rules are available in machine-readable form at this hyperlink.

All that remains is lifting this rewriting system for *formulas* to one for *circuits.*

E.3 Convergent Term Graph Rewriting for Boolean Circuits

Following [Plu99], we can lift a term rewriting system to a term *graph* rewriting system by generalizing the notion of pattern matching. We say there is a *hypergraph morphism* f between hypergraphs G and H if there are vertex and edge functions $f_V : V_G \to V_H$ and $f_E : E_G \to E_H$ that preserve labels and attachment nodes, so: for every $g \in E_G$ $\text{lab}_G(g) = \text{lab}_H(f_E(g))$ and $\text{att}_H(f_E(g)) = f_V^*(\text{att}_G(g))$ where f_V^* is the vectorized f_V. For a term t, define $\diamond t$ as the parse tree of t encoded by a hypergraph, with all repeated variables collapsed into "open" vertices — that is, the edge x_i is deleted for each x_i, but the unique result vertex remains and is referenced by every edge that was attached to x_i in the parse tree. A term graph L is an *instance* of a term l if there is a graph morphism $\diamond l \to L$ that sends the root of $\diamond l$ to the root of L. Given a node v in a term graph G and a term rewrite rule $r \to \ell$, the pair $\langle v, \ell \to r \rangle$ is a *redux* if the subgraph of G reachable from v (denoted $G[v]$) is an instance of ℓ. Finally, we define a single step of graph rewriting: essentially, a subgraph matching the left hand side of a rule is sliced out and replaced with the right-hand side.

Definition 25 (Term Graph Rewriting (Definition 1.4.5 of [Plu99]))**.** Let G be a term graph containing a

redux $\langle v, l \to r \rangle$. There is a *proper rewrite step* from G to H where H is constructed by

1. $G_1 \leftarrow G - \{e\}$ where e is the unique edge that satisfies $\text{res}(e) = v$
2. $G_2 \leftarrow$ the disjoint union of G_1 with $\diamond r$ where
   - v is identified with $\text{root}(\diamond r)$
   - Every edge labeled with a variable in $\diamond r$ is identified according to the morphism that matched ℓ to G.
3. Garbage collection: H is obtained from G_2 by deleting all nodes and edges not reachable from the root.

The following Theorem shows an immediate connection between Term Rewriting that we introduced in the previous section and Term Graph Rewriting:

Theorem 26 (Corollary 1.7.4 of [Plu99])**.** *If $\mathcal{R}$ is a convergent term rewriting system, then $\mathcal{R}$ induces a convergent term graph rewriting system with collapse.*

Collapse is an additional rule in the term graph rewriting system that allows us to merge two rooted subhypergraphs if there exists a root-preserving hypergraph morphism between them. For circuits, this operation would correspond to finding redundant subcircuits and combining them into one. This is natural simplification step to include; indeed, if our goal was to optimize non-optimal circuits then any system missing this rule would be insufficient. However, in gate elimination arguments this rule's addition will be irrelevant—we typically start with optimal circuits and being able to apply a collapse rule would immediately violate said optimality. It's addition will serve only to guarantee that the system is convergent.

Applying this lifting theorem to our term rewriting system $\mathcal{R}_B$ for simplification of DeMorgan formulas yields our system for gate elimination.

Theorem 27. *$\mathcal{R}_B$ induces a convergent Term Graph Rewriting System, denoted $\mathcal{S}$.*