Formalizing Gate Elimination

Marco Carmosino [*], Ngu Dang [†], Tim Jackman [‡]

August 2025

Abstract

Gate elimination is a widely used technique for proving lower bounds in circuit complexity, yet it remains somewhat informal and lacks a foundational treatment. In this work, we formalize gate elimination as a convergent term graph rewriting system over Boolean circuits in the DeMorgan basis. Our system defines simplification via local rewriting rules derived from Boolean identities and ensures convergence—i.e., every simplification sequence yields a unique normal form. This convergence property enables rigorous reasoning about structural properties of simplified circuits without dependence on the order of simplification. Our work aims to bridge circuit complexity and formal methods, providing a framework suitable for machine verification.

[*] marco@ntime.org, IBM Research
[†] ndang@bu.edu, Boston University
[‡] tjackman@bu.edu, Boston University

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. Some of the well-known bases include the DeMorgan basis (i.e. fan-in 2 AND, fan-in 2 OR, and fan-in 1 NOT gates), the $\mathcal{U}_2$ basis (i.e. all fan-in 2 Boolean functions except parity and equivalence), and the $\mathcal{B}_2$ basis (i.e. all fan-in 2 Boolean functions). Besides these three bases, we will consider the functionally complete basis $\{\wedge, \oplus, \neg\}$ where $\oplus$ is binary exclusive-or.

Gate elimination is a fundamental and crucial technique for proving circuit lower-bounds. In particular, the current unconditional lower-bounds known for functions in DeMorgan, $\mathcal{U}_2$ basis [Red73; Sch74; Zwi91; IM02; ILMR02] and $\mathcal{B}_2$ basis [Sch74; Sto77; DK11; FGHK16; LY22] are proven using gate elimination . The technique is quite straightforward: input variables are substituted (by constants or by other functions) and then the resulting circuit is simplified. During simplification, complexity measures, like the number of gates, are tracked and then often lifted to lower bounds using induction and downward self-reducibility. Schnorr's tight $3(n-1)$ bound for XOR_n (and $\neg\mathsf{XOR}_n$) exemplifies this nicely: substituting for an input with a constant allows removal of at least three gates and results in a circuit computing XOR_{n-1} (or $\neg\mathsf{XOR}_{n-1}$). Successive applications of this argument show that the original XOR_n circuit must have had at least $3(n-1)$ gates.

Despite its widespread use, gate elimination remains somewhat informal and its arguments ad-hoc. Substitutions and studied complexity measures vary depending on the base function. Formalizing the technique will allow more formal study: we can categorize its strengths and weaknesses rigorously.

1.1 Our Results

In this work, we develop a *formal* foundation for gate elimination as a ***convergent* term graph rewriting system**. In particular, we define a simplification system for Boolean circuits over two bases using ideas from term rewriting [BN98] and graph-based computation models [Plu99], and derive our rules from a selected list of Boolean identities. Then, by applying the Knuth-Bendix algorithm [KB70; Hue81], we generate convergent formula simplification systems that can be lifted into rewriting systems on circuits via Plump's term graph rewriting framework. The resulting systems are *convergent*, that is, any valid sequence of simplifications terminates in a unique normal form, regardless of order.

Theorem (Theorem 18)**.** *Gate elimination is convergent in the DeMorgan (*$\{\wedge, \vee, \neg\}$*) and* $\{\wedge, \oplus, \neg\}$ *bases.*

This convergence property is useful for arguments via gate elimination as it ensures that the structural properties of the simplified circuits are *invariant* under the simplification. As such, convergence permits formal reasoning about circuits up to their normal form and eliminates the need for case-by-case justification for each choice of simplification sequence. Furthermore, by casting gate elimination within a convergent rewriting framework, we hope to bring this powerful and widely use technique into alignment with the contemporary machine verifiability. There has been recent work which formalizes term graph rewriting for use with proof assistants [WHU23], bridging complexity theory and formal methods. As a starting example, we reproved Schnorr's lower bound for XOR_n [Sch74] using the system (Section 3.2) and in a structured and verifiable manner (Appendix C).

1.2 Separating "Equivalent" Bases

An interesting consequence of this formalization is that it separates seemingly "equivalent" bases. When NOT gates are *free* (i.e. do not contribute to circuit size or depth), the DeMorgan basis and the $\mathcal{U}_2$ basis are identical for all but one non-degenerate function (in particular, $f(x) = \neg x$). All non-trivial Boolean circuits in $\mathcal{U}_2$ can be translated into equivalent size and depth DeMorgan circuits and vice-versa. We show this folklore result explicitly in A. This equivalence also holds between $\mathcal{B}_2$ and $\{\wedge, \oplus, \neg\}$ when NOT gates are *free*. Indeed, circuit size arguments freely move between these formulations, like [Pau75] which first classifies its $\mathcal{B}_2$ gates as "AND-like" and "XOR-like" gates before proceeding. However, with respect to gate elimination, these bases and their "equivalent" counterparts are qualitatively distinct.

Gate elimination is *not* convergent in either the $\mathcal{U}_2$ or $\mathcal{B}_2$ basis, even when restricted to optimal circuits (Appendix B). This failure arises from the presence of negation gates which need to be pushed "up" and/or "down" during simplification, causing divergent simplification paths and creating non-isomorphic circuits (see Figures 5 and 6 in Appendix B).

Even different bases are often treated similarly since translation often only sees constant changes to size and depth. As researchers are interested in proving non-linear lower bounds, this cost is seemingly inconsequential. However, this results highlights how consequential a basis choice can be; arguments in the DeMorgan basis or $\{\wedge, \oplus, \neg\}$ can be *easier* than their $\mathcal{U}_2$ or $\mathcal{B}_2$ counterparts. This is especially true if implemented in a proof assistant—formalizing in these bases may cut down search space depending on the specific argument.

1.3 Related Work

Gate Elimination: A Meta-level Analysis As the fundamental tool for proving general circuit lower bounds, gate elimination as a technique has begot prior study. In order to demonstrate its limitations, [GHKK18] designed functions whose circuits are "resistant" to gate elimination: any single substitution reduces their complexity by merely a constant. As such, a constant number of substitutions cannot reduce their complexity by a superlinear factor. As all gate elimination arguments have relied on only a small constant number of substitutions, their work showed a barrier to proving non-linear bounds on general classes of circuits using gate elimination.

Verifying Term Graph Optimizations with Proof Assistants. Term rewriting and term graph rewriting have been extensively studied in algebraic settings [BN98; Plu99], with recent work by Webb, Hayes, and Utting [WHU23] exploring formal verification of term graph optimizations in Isabelle which is a higher-order logic (HOL) theorem prover. However, to our knowledge, our work is the first to formally encode gate elimination as a term graph rewriting system. This step opens the door to mechanized verification of circuit lower bounds.

Characterizing Optimal Circuits Beyond the much-studied quantitative circuit complexity problems of size and depth lies a natural *qualitative* question: what do the optimal circuits for a particular function look like? Empirically, answering this question seems difficult [Weg87]. Some of the earliest work on this question include [Sat81] and [BS84] which investigated when optimal circuits for 2-output Boolean functions compute each output independently of the other. [Kom11] continued this line of work by characterizing the the optimal circuits computing XOR in the DeMorgan basis when not gates *contribute* to the circuit size. They extended this to other bases in [Kom18]. A more recent result of [Ila20] characterizes the optimal structure of circuits computing $\bigvee_{i\in[n/2]}(x_{2i-1} \wedge x_{2i})$. This characterization is then leveraged to prove a breakthrough result: the partial function minimum circuit size problem (MCSP*) is hard, assuming the exponential time hypothesis (ETH).

These scarce results for such a fundamental question in circuit complexity serve as a strong motivator for our work. These proofs often involve a simple albeit lengthy case analysis relying heavily gate elimination. Formalizing this tool and implementing it in a proof assistant could yield more fruitful searches for optimal structures.

1.4 Future Work

The natural follow-up to this work is to implement this system in a proof assistant, formalize known gate elimination arguments like Schnorr's $3(n-1)$ bound, and use the tooling to prove tighter bounds. In particular, computer assistance may produce significant gains for structural characterization proofs which are often plagued by long, intricate, albeit simple case work that is ripe for automation. Extending the system may further this goal. For instance, the addition of "absorption laws" (e.g. $p \wedge (p \vee q) \equiv p$) does not disrupt convergence. Additional simplification rules may help capture more sophisticated gate elimination arguments.

Lastly, there are other bases that are occasionally considered, for instance {NAND}. Modeling gate elimination in them as term graph rewriting systems and analyzing their respective properties could prove useful. Is there a unifying reason for why gate elimination is sometimes convergent? Can we determine whether a basis admits convergence without running the Knuth-Bendix algorithm?

2 Preliminaries

2.1 Circuits as Term Graphs

We study general circuits over two bases. The first is 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. The second basis is $\{\wedge, \oplus, \neg, 0, 1\}$, where $\vee$ has been replaced by the binary exclusive-or function $\oplus$. 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* respectively, $\mathrm{lab}_C : E_C \to \mathcal{B} \cup X$ 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 vertices 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 vertices are "wires" between gates — essentially dual to the standard encoding of circuits as DAGs.

2.2 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. In this paper, we will develop a system $\mathcal{S}$ where A consists of Boolean circuits over the DeMorgan basis and $C \to C'$ holds when C simplifies to C' via a single step of gate elimination. To this end, we introduce some terminology about and desirable properties of abstract rewriting systems.

Definition 1 (Paths Through Rewrite Relations, Definition 2.1.3 of [BN98])**.** 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'. 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 $b' \xrightarrow{*} c$ and

convergent iff it is both confluent and terminating.

An object a is *in normal form* if there is no b such that $a \to b$. Objects in normal form are often more tractable to reason about. For example, circuits C in normal form according to $\mathcal{S}$ have:

- No double negations.
- No constants unless C only computes a constant function — e.g., $(x_i \vee \neg x_i)$ normalizes to 1.
- No identity gates — sub-circuits $(\gamma \wedge \gamma)$ and $(\gamma \vee \gamma)$ do not occur.
- No redundant sub-circuits — each intermediate Boolean function computed by C is unique.

Since each rewrite rule of $\mathcal{S}$ either decreases or preserves the number of costly gates, the system is terminating. Indeed, given a circuit C with bits $b = b_1, \ldots, b_n$ substituted for all input variables, rewriting C using $\mathcal{S}$ until termination just evaluates C on b. These properties of $\mathcal{S}$ are straightforward to establish, so the substantial work in showing that $\mathcal{S}$ is "well behaved" goes into proving confluence.

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. Furthermore, we often wish to carefully sequence and analyze only the *first few steps* of gate elimination, and then apply "all remaining simplifications" without considering them in detail. It is critical that $\mathcal{P}'$ emerge no matter how the steps of elimination are sequenced.

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 Circuit Simplification: System $\mathcal{S}$

There are three parts to $\mathcal{S}$: (1) notions of redundancy and pattern matching for sub-circuits (Definitions 3 and 4), (2) a set of circuit rewrite rules given as pairs of patterns (Figures 1 – 4), and (3) the procedure for pattern-substitution in a circuit (Algorithm 1). System $\mathcal{S}$ is then the binary relation on circuits induced by setting $C \to C'$ when either (a) C matches the left-hand side l of a rule $\langle l \mapsto r \rangle$ and C' is the result of substituting pattern r for l in C, or (b) C has a redundant sub-circuit that is "collapsed" in C'. Because $\mathcal{S}$ is a Term Graph Rewriting System, convergence can be algorithmically certified. See Section 4 for the full construction of system $\mathcal{S}$ as well as proofs and hyperlinks for verification. We now describe $\mathcal{S}$ in sufficient detail to use in proofs by gate elimination.

Definition 2 (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, so f_E sends every edge γ of G to an edge of H with matching label — $\mathrm{lab}_G(\gamma) = \mathrm{lab}_H(f_E(\gamma))$ and

attachments, so for every edge γ of G, f_V is an order-preserving map from the vertices attached to γ to the vertices attached to $f_E(\gamma)$ in H — recalling that $\mathrm{att}(\gamma)$ is a string:

$$\forall \gamma \in E_G \ \forall i \in |\,\mathrm{att}_G(\gamma)| \quad f_V(\mathrm{att}_G(\gamma))[i] = \mathrm{att}_H(f_E(\gamma))[i].$$

Definition 3 (Collapsing Redundant Sub-Circuits)**.** Circuit C *collapses* to circuit D if there is a non-injective hypergraph morphism $C \to D$ mapping root_C to root_D, denoted $C \succ D$.

Definition 4 (Pattern & Redux in Circuits)**.** A *pattern* is just a circuit L where vertices γ without a unique gate g such that $\mathrm{att}_L(g)[0] = \gamma$ may occur; we call these *open vertices*. Circuit D is then 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 the sub-circuit of C rooted at α (denoted $C[\alpha]$) is an instance of L.

Most rewrite rules displayed in Figures 1 – 4 use the open vertex labeled γ to match "the rest of the sub-circuit D." That is, a morphism p will send γ to each node of D not explicitly mentioned to "match" the pattern. Each family of rules then plays a different role in gate elimination.

Normalizing enforces that each circuit does not contain "duplicate" gates — such as double negation or trivial identity. The "zero elimination" rule is included to ensure confluence of rewriting. A "one elimination" rule would have served just as well and the choice is arbitrary.

Fixing applies when a gate computes a constant function because of one input.

Passing applies when a gate computes the identity function because of one input.

Tautology removes redundant (costly) gates which trivially compute constants.

Distinct rules are required for fixing, passing, and tautology to handle left and right inputs because inclusion of "AND is commutative" and "OR is commutative" as rewrite rules makes confluence impossible [Soc91].

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

Require: C is a circuit containing the redex $\langle \alpha, R \mapsto L \rangle$, with $p : R \to C[\alpha]$ the witnessing morphism

$C_1 \leftarrow C - \{a\}$ where $a = \text{res}^{-1}(\alpha)$ ▷ *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

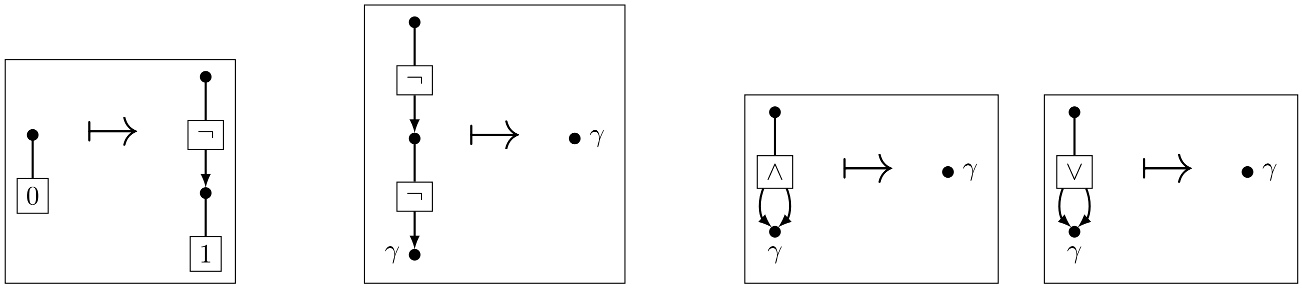
if $\gamma \in R$ **then** ▷ *Does* R *reuse a subcircuit?*

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

else

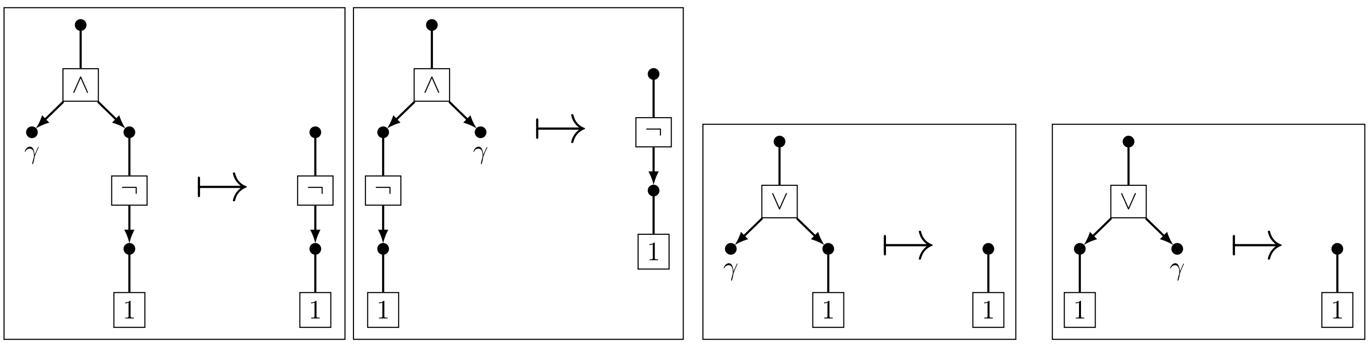
  $C_4 \leftarrow C_3$ ▷ R *does not reuse any subcircuits — do nothing*

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



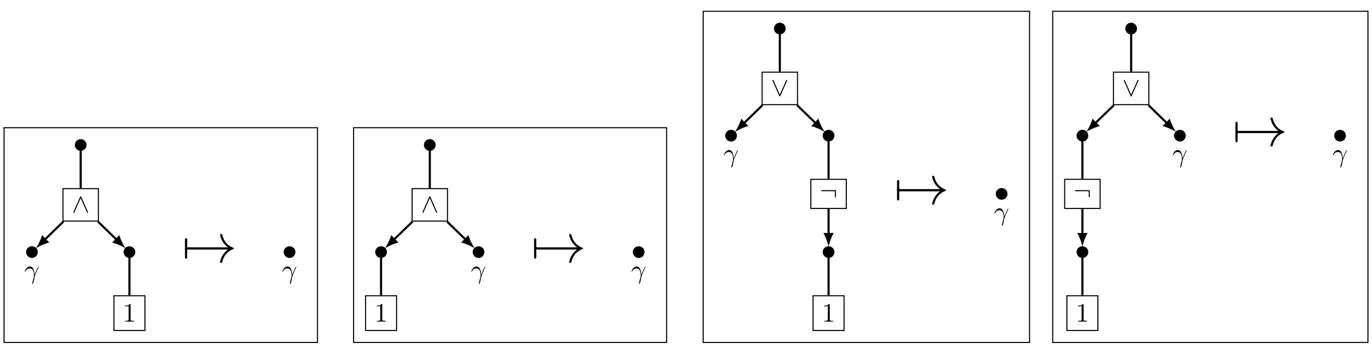
(a) Zero Elimination (b) 2xNegation Elimination (c) AND-De-Dupe (d) OR-De-Dupe

Figure 1: **Normalizing Rules**



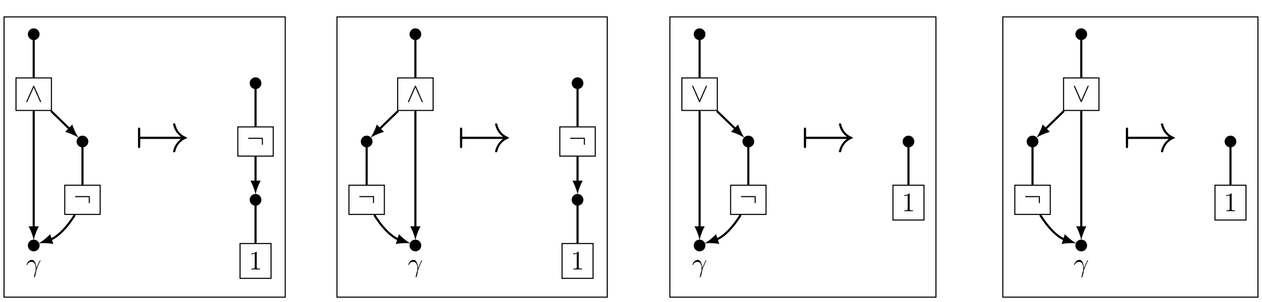
(a) AND-Left (b) AND-Right (c) OR-Left (d) OR-Right

Figure 2: **Fixing Rules**



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

Figure 3: **Passing Rules**



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

Figure 4: **Tautology Rules**

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

In this section, we propose a proof for Schnorr's lower-bound for the parity function XOR using the system $\mathcal{S}$. To begin with, we define the parity functions XOR_n formally as:

Definition 5. 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 6 (($\neg$)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 $\neg\mathsf{XOR}_n(x)$

Let e_i denote the Boolean vector that is zero everywhere except for position i.

Fact 7 (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, $(\neg)\mathsf{XOR}_n(\alpha) \neq (\neg)\mathsf{XOR}_n(\alpha \oplus e_i)$.*

We now prove Schnorr's lower bound using $\mathcal{S}$. The purpose of this is to demonstrate that $\mathcal{S}$ is powerful enough to capture traditional gate elimination arguments.

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.

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 7 (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 6 ($(\neg)\mathsf{XOR}$ 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 $3(n-1)$.

□

Lastly, in Appendix C, we prove Theorem 8 with a structured proof as proposed by Lamport [Lam95; Lam12]. Due to the case analysis and repeated proofs by contradiction present here, the alternative format may be easier to verify. Furthermore, we believe that this presentation style makes the intricate case analysis present both in our proof of Schnorr 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 this proof.

4 Gate Elimination as a Convergent Term Graph Rewriting System

In this section we 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.

This detailed treatment is for the DeMorgan basis ($\{\wedge, \vee, \neg\}$); the convergent system for $\{\wedge, \oplus, \neg\}$ follows similarly albeit with a different set of Boolean identities.

4.1 Boolean Identities

The identities present in $\mathcal{E}_B$ are valid for Boolean algebra and appear in standard gate elimination arguments (Definition 9). 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*[1] formally, because we transform $\mathcal{E}_B$ into a convergent rewriting system in the next subsection.

Definition 9 (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)

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

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.

4.2 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 10 (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 11 (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 12 (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. $g \wedge 1 \approx g \Longrightarrow g \wedge 1 \to g$) 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 13 (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 14. *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 9 using the open-source software Knuth-Bendix Completion Visualizer (KBCV, [SZ12]). The algorithm terminated and printed the TRS $\mathcal{R}_B$ listed in Definition 15 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 15 (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.*

4.3 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 $g \in E_G$ $\mathrm{lab}_G(g) = \mathrm{lab}_H(f_E(g))$ and $\mathrm{att}_H(f_E(g)) = f_V^*(\mathrm{att}_G(g))$ 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 16 (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 $\mathrm{res}(e) = v$

2. $G_2 \leftarrow$ the disjoint union of G_1 with $\diamond r$ where

   - v is identified with 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 17 (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 largely irrelevant—we typically start with optimal circuits and being able to apply a collapse rule would immediately violate said optimality. However, one benefit of its inclusion is that any vertex in a circuit in normal form has at most one $\neg$ gate successor because multiple negations reading the same gate can be collapsed into a single $\neg$ gate. This property can trim down case analysis when applying gate elimination arguments to normalized optimal circuits.

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

Theorem 18. *$\mathcal{R}_B$ induces a convergent Term Graph Rewriting System, denoted $\mathcal{S}$.*

References

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

[BS84] Norbert Blum and Martin Seysen. “Characterization of all Optimal Networks for a Simultaneous Computation of AND and NOR”. In: *Acta Informatica* 21 (1984), pp. 171–181. DOI: 10.1007/BF00289238. URL: https://doi.org/10.1007/BF00289238.

[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.

[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).

[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.

[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.

[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.

[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.

[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.

[Pau75] Wolfgang J. Paul. "A 2.5 n-lower bound on the combinational complexity of Boolean functions". In: *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*. STOC '75. Albuquerque, New Mexico, USA: Association for Computing Machinery, 1975, pp. 27–36. ISBN: 9781450374194. DOI: 10.1145/800116.803750. URL: https://doi.org/10.1145/800116.803750.

[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.

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

[Sat81] Jürgen Sattler. "Netzwerke zur simultanen Berechnung Boolescher Funktionen". In: *Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23-25, 1981, Proceedings*. Ed. by Peter Deussen. Vol. 104. Lecture Notes in Computer Science. Springer, 1981, pp. 32–40. DOI: 10.1007/BFB0017293. URL: https://doi.org/10.1007/BFb0017293.

[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.

[Soc91] Rolf Socher-Ambrosius. "Boolean Algebra Admits No Convergent Term Rewriting System". In: *Rewriting Techniques and Applications, 4th International Conference, RTA-91, Como, Italy, April 10-12, 1991, Proceedings*. Ed. by Ronald V. Book. Vol. 488. Lecture Notes in Computer Science. Springer, 1991, pp. 264–274. DOI: 10.1007/3-540-53904-2_102. URL: https://doi.org/10.1007/3-540-53904-2%5C_102.

[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.

[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.

[Zwi91] Uri Zwick. "A 4n lower bound on the combinational complexity of certain symmetric boolean functions over the basis of unate dyadic Boolean functions". In: *SIAM Journal on Computing* 20.3 (1991), pp. 499–505.

A Equivalent Bases Using Free Not Gates

Recall that $\mathcal{U}_2$ is the basis consisting of all two-input Boolean functions except XOR_2 and $\neg\mathsf{XOR}_2$ and the DeMorgan basis is $\{\wedge, \vee, \neg\}$. We show explicitly that these bases are equivalent in terms of size when $\neg$ gates are not counted. The proof that $\mathcal{B}_2$ and $\{\wedge, \oplus, \neg\}$ are equivalent when the latter's $\neg$ gates are free is identical.

Lemma 19 (Folklore)**.** *Let f be any non-degenerate Boolean function besides $f(x) = \neg x$. Then the size and depth complexity of f in the $\mathcal{U}_2$ ($\mathcal{B}_2$) basis and the DeMorgan ($\{\wedge, \oplus, \neg\}$) basis where $\neg$ gates are free is equal.*

Proof. Let $CC_U(f)$ denote the circuit size complexity of f in the $\mathcal{U}_2$ basis. Let $CC_D(f)$ denote the circuit size complexity of f in the DeMorgan basis where $\neg$ gates are free.

We begin by arguing $CC_D(f) \leq CC_U(f)$ by translating any optimal $\mathcal{U}_2$ circuit into a DeMorgan circuit of equal size. We recall the 14 binary Boolean operators that form the $\mathcal{U}_2$ basis below.

p	q	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}
T	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F
T	F	T	F	T	F	F	T	T	F	F	T	F	T	T	F
F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F
F	F	T	F	F	T	F	T	T	F	T	F	F	T	F	T

We claim that no optimal circuit for f uses u_i gates for $i \in [6]$. For u_1 and u_2, replacing these gates with the constants 0 and 1 would reduce the size of the circuit. Similarly, u_3 and u_5 gates are non-optimal since they just compute the first and second inputs respectively. If a u_3 gate appears in a circuit, we can instead feed it's first input into it's outputs for the same effect. The gates u_4 and u_6 simply negate their first and second inputs respectively. Similarly, if a u_4 or u_6 appear internally we can pass the first and second inputs up to the gates' outputs but we must change the labels of these gates. We record the transformations in Table 1 below:

		u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}
$\neg p$	q	u_{12}	u_{11}	u_{13}	u_{14}	u_8	u_7	u_9	u_{10}
p	$\neg q$	u_{13}	u_{14}	u_{12}	u_{11}	u_{10}	u_9	u_7	u_8

Table 1: The $\mathcal{U}_2$ Basis

For example, if a u_4 gate is the first input of u_7, then we can remove the u_4 gate, feed its' first input into the first input of the u_7 gate and then relabel the u_7 gate to be a u_{12} gate. If a u_4 or u_6 gate is the output of the circuit, then we observe it must be negating one of the remaining 8 Boolean functions since $f(x) \neq \neg x$. We can apply the transformations listed below to remove it.

	$\neg(u_7)$	$\neg(u_8)$	$\neg(u_9)$	$\neg(u_{10})$	$\neg(u_{11})$	$\neg(u_{12})$	$\neg(u_{13})$	$\neg(u_{14})$
Is Equivalent To	u_8	u_7	u_{10}	u_9	u_{12}	u_{11}	u_{14}	u_{13}

Table 2: Negation Transformations for u_7 through u_{14}

Lastly, the remaining 8 binary Boolean functions in the basis can all be represented in the DeMorgan basis with only one costly gate:

	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}
Is Equivalent To	$p \vee \neg q$	$\neg p \wedge q$	$\neg p \vee q$	$p \wedge \neg q$	$p \wedge q$	$\neg p \vee \neg q$	$p \vee q$	$\neg p \wedge \neg q$

Table 3: Translations Between $\mathcal{U}_2$ and DeMorgan formulas for u_7 through u_{14}

To show $CC_U(f) \leq CC_D(f)$ we show how to take any optimal DeMorgan circuit and transform it into an equivalent size $\mathcal{U}_2$ circuit. To do so, if any $\neg$ gate has fanout m where $m > 1$, we split the $\neg$ gate up into m copies, each with fanout exactly one. We also remove any double negations. Neither of these transformations change the size of the circuit.

Then, in increasing topological order, we replace each $\wedge$ and $\vee$ gate with one of the $\mathcal{U}_2$ gates depending on whether their left or right inputs is negated. There are 8 possible configurations, and the correspondence can be seen in Table 3. Lastly, if the entire circuit is negated (i.e. the final output gate is $\neg$), then we absorb it into the $\mathcal{U}_2$ gate below using the transformations listed in Table 2.

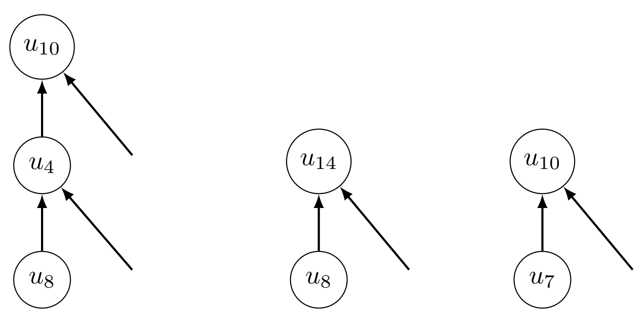
Since $CC_U(f) \leq CC_D(f)$ and $CC_D(f) \leq CC_U(f)$ we get $CC_U(f) = CC_D(f)$. Notice that none of these transformations change the *depth* of the circuits as well, and thus the bases are equivalent with respect to depth complexity as well. $\square$

B Gate Elimination in $\mathcal{U}_2$ Does Not Converge

In this section we will demonstrate that any similar formulation of gate elimination in $\mathcal{U}_2$ is **not** convergent. As $\mathcal{U}_2 \subset \mathcal{B}_2$, the same argument shows gate elimination in $\mathcal{B}_2$ is not convergent. To do this we produce a circuit and substitution which, after applying a different series of rewrites, results in two distinct circuits (i.e. non-isomorphic). We recall the table of binary functions in $\mathcal{U}_2$ below:

p	q	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}
T	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F
T	F	T	F	T	F	F	T	T	F	F	T	F	T	T	F
F	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F
F	F	T	F	F	T	F	T	T	F	T	F	F	T	F	T

As discussed in Section A, optimal circuits in $\mathcal{U}_2$ with more than a single gate do not ever contain u_i for $i \in [6]$. Recall that u_1 and u_2 are constant T and F while u_3 and u_5 simply compute their first and second inputs respectively. These gates can be easily removed from any circuit. The gates u_4 and u_6 compute the negation of their first and second inputs respectively. If there are multiple gates in the circuit, then these negations can either be "pushed" down or up by passing their input/output wires and relabeling gates. For instance,



(a) u_4 can be removed by pushing up or down

(b) Pushing up: u_{10} is relabeled as u_{14}

(c) Pushing down: u_8 is relabeled as u_7

Figure 5

This example disproves convergence if both of these rewrite rules are present in our system. Indeed, we **must** include rewrite rules that both push negations up and down; if we do not (and choose to always push negations in one direction), then superfluous u_4 and u_6 gates could survive at the extremes of the circuit. Furthermore, this issue cannot be sidestepped even if we limit ourselves to optimal circuits. This is because "passing" rules in this rewriting system must generate intermediate negation gates. Take for instance the following example:

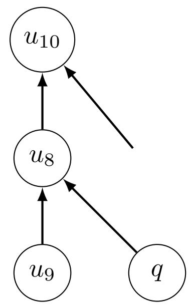


Figure 6

Since u_8 is equivalent to $\neg p \wedge q$, if we substitute $q \leftarrow 1$ we would like the equivalent of a "passing" rule to be applied. However, the output wires of u_8 should pass to a gate computing $\neg u_9$, which does not currently exist. Thus, rather than removing u_8, this substitution simply relabels u_8 to be u_4. Then u_4 can be eliminated by "pushing" it either up or down leading to two non-isomorphic circuits.

C 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 in 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.

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, $C\mid_{x_j=\alpha}$ does not depend on x_i
   (e) Contradiction to Fact 7.
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, $C\mid_{x_i=\alpha}$ is a constant function
   (e) Contradiction to Fact 6.

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 $3(n-1)$.

$\square$