A NORMAL CIRCUIT WITH A REDUCIBLE TERM: NON-LEFT-LINEARITY IN CONVERGENT GATE ELIMINATION

KOREA SUPERINTELLIGENCE LABS

Abstract. Carmosino, Dang and Jackman (arXiv:2602.17942v1) present circuit simplification as a convergent term graph rewriting system: sixteen rewrite rules $\mathcal{R}_B$ on DeMorgan formulas, obtained by Knuth–Bendix completion, lifted to circuits-as-hypergraphs in two ways — a system $\mathcal{S}$ that includes Plump's collapse rule and a system $\mathcal{S}'$ that does not — and then used to give a constructive form of Schnorr's $3(n-1)$ lower bound for XOR_n, through a refuter that runs on $\mathcal{S}'$-normal circuits. We exhibit a four-input, four-gate DeMorgan circuit that meets every precondition of that algorithm and is normal for $\mathcal{S}'$, but whose unfolded formula $(((\neg x_1 \wedge \neg x_1) \vee x_2) \wedge x_3) \vee x_4$ is $\mathcal{R}_B$-reducible. Each of the four input variables is carried by exactly one hyperedge; what is duplicated is a pair of distinct *free* negation gates. On this circuit the algorithm's assertion that its topologically minimal binary gate reads $(\neg)x_p$ and $(\neg)x_q$ with $p \neq q$ fails: the minimal gate is unique, is first in every topological order, and reads two distinct nodes that both represent $\neg x_1$, so every pair p, q of the asserted shape has $p = q$, and the algorithm as printed has no next move. That plain term graph rewriting is incomplete for non-left-linear rules is Plump's Example 1.4, and we claim no novelty for it; what we add is the instantiation. Six of the sixteen rules of $\mathcal{R}_B$ are non-left-linear; the normal-form property invoked to justify the assertion is stated for $\mathcal{S}$ while the algorithm normalises under $\mathcal{S}'$; and the result cited to license $\mathcal{S}'$ is Plump's Theorem 1.7.12, whose relevant clause concerns rewriting modulo bisimilarity, where collapsing is built in, plain rewriting being governed by his Theorem 1.7.11, which requires left-linearity. One pass of hash-consing restores the blocked redex, changes neither the computed function nor the binary-gate count, and comes with an explicit non-injective root-preserving morphism: the witness *does* collapse in the source's own sense, which locates the defect exactly at the omission of collapse and suggests that the source's theorem is repairable there. We also record proofs of the convergence of $\mathcal{R}_B$, of $\mathrm{CC}_D(\mathsf{XOR}_2) = 3$ by a complete enumeration whose completeness is itself proved, and of an explicit circuit family with exactly $3(n-1)$ binary gates, verified to compute XOR_n for $n \leq 6$. Every theorem and proposition below is machine-checked in Lean 4.

1. Introduction

The objects. Gate elimination is the only technique known to prove lower bounds on the size of unrestricted Boolean circuits computing explicit functions. Its arguments all have the same shape: take a circuit that is too small, restrict an input variable so as to kill several gates, simplify the result, and recurse. The simplification step is where the informality lives — "simplify" is usually left to the reader, and the structural properties of the simplified circuit are asserted rather than derived.

A recent preprint of Carmosino, Dang and Jackman [3] proposes to remove that informality by making simplification a *convergent rewriting system*. Their chain is:

(1) a list $\mathcal{E}_B$ of Boolean identities over the DeMorgan signature $\{\wedge, \vee, \neg, 0, 1\}$, with $\wedge, \vee$ binary, $\neg$ unary and $0, 1$ constants;
(2) Knuth–Bendix completion [6], run in KBCV [7], turning $\mathcal{E}_B$ into a sixteen-rule term rewriting system $\mathcal{R}_B$ on DeMorgan formulas, their Definition 6, claimed convergent in their Lemma 7 and certified by a proof in the CPF format for CeTA [5] in their supplementary artifact [1];
(3) a lift of $\mathcal{R}_B$ from formulas to circuits-as-hypergraphs, by citing Plump's chapter [9]: their Theorem 12 ("Corollary 1.7.4 and Theorem 1.7.12 of [9]") says that a convergent term rewriting system induces a term graph rewriting system of proper steps *plus* the collapse rule that is convergent up to hypergraph isomorphism, and a system of proper steps alone that

is "convergent up to bisimilairty [sic] classes of hypergraphs"; their Theorem 13 instantiates this at $\mathcal{R}_B$ and names the two systems $\mathcal{S}$ (with collapse) and $\mathcal{S}'$ (without);

(4) Schnorr's bound $\mathrm{CC}_D(\mathsf{XOR}_n) \geq 3(n-1)$ [10], their Theorem 18, re-presented with $\mathcal{S}$ doing the simplification; and its constructive form, their Theorem 19: given a DeMorgan circuit on n inputs of size $s < 3(n-1)$, produce in polynomial time an input on which it differs from XOR_n. The algorithm that does so is their Algorithm 2, and it works with $\mathcal{S}'$, not $\mathcal{S}$.

Throughout, *size* counts binary gates only: negations are free. A hypergraph C, with node set V_C, hyperedge set E_C, edge labelling lab_C and attachment map att_C, is a *term graph* when (1) every node is reachable from a root node, (2) C is acyclic, and (3) each node is the result node of a unique *edge*. Nothing in (1)–(3) says that two distinct edges may not carry the same label, and that is the crack this paper works in.

What the source leaves open, and what is settled here. The source is explicit about why it uses $\mathcal{S}'$ rather than $\mathcal{S}$: "Unfortunately, it is not clear if the collapse rule can be implemented efficiently. Therefore, we use the plain rewriting system (without collapse rule) when constructive arguments are required" ([3], §3.3). It is equally explicit about what it does not do: "Verifying convergence of our system and Schnorr's argument using a proof assistant like Lean4 is a natural next step" ([3], §1.1). A successor preprint by the same authors [4] gives a refuter for XOR_n without the term-graph apparatus; nothing below bears on it.

This paper settles the first of those. Section 4 exhibits a DeMorgan circuit C^* on four inputs with four binary gates — so $n = 4 > 3$ and $4 < 3(n-1) = 9$, which is exactly the `Require` line of Algorithm 2, and all four variables are read — such that

- no proper rewrite step of $\mathcal{S}'$ applies at any node of C^*, so the algorithm's first line, $C_0 \leftarrow$ Normalize C, returns C^* unchanged; while
- the formula $\mathrm{term}(C^*) = (((\neg x_1 \wedge \neg x_1) \vee x_2) \wedge x_3) \vee x_4$ that C^* unfolds to *is* $\mathcal{R}_B$-reducible, by the rule $g \wedge g \to g$.

The duplication in C^* is not a duplicated input: each of the four input variables is carried by exactly one hyperedge, and what occurs twice is a *free* negation gate, which the source's cost model neither charges for nor requires to be shared. On C^* the assertion on line 8 of Algorithm 2 fails. That line reads

> **Assert:** h has inputs $(\neg)x_p, (\neg)x_q$ for $p, q \in V$ and $p \neq q$ *(the assertion holds because C_i is in normal form)*

where line 7 has just set h to the topologically minimal $\{\wedge, \vee\}$-gate of C_i. On C^* that gate is unique and is first in *every* topological order, and its two arguments are two distinct nodes both representing $\neg x_1$; so readings of the asserted shape exist but force $p = q$, and the algorithm as printed has no next move (Theorem 4.3).

What is new here, and what is not. The general phenomenon is not new, and we want to be exact about that. Plump's chapter states it in its Example 1.4 [9]:

> One obstacle to the completeness of plain term graph rewriting are non-left-linear term rewrite rules. For instance, the rule $eq(x, x) \to true$ cannot be applied to the tree $\Delta eq(0, 0)$ because there is no graph morphism $\diamond eq(x, x) \to \Delta eq(0, 0)$ (see Figure 1.10). Hence, $\Delta eq(0, 0)$ is not reducible by $\Rightarrow$ or $\Rightarrow_{copy}$ although the represented term is reducible. Figure 1.10 also shows how to overcome the problem by collapsing: identifying the two occurrences of 0 enables a subsequent application of the rewrite rule.

Nothing below should be read as discovering that non-left-linearity breaks plain term graph rewriting. What is new is the *instantiation*, in three parts, none of which is in the source or, as far as we could determine, in the literature.

(1) *Six of the sixteen rules of $\mathcal{R}_B$ are non-left-linear* — $g \wedge g \to g$, $g \vee g \to g$ and the four tautology rules. The words "left-linear" and "linearity" do not occur in [3].
(2) *A single circuit can meet all of Algorithm 2's preconditions at once* and still exhibit the gap: legality as a term graph, $n > 3$, size below $3(n-1)$, every variable read, $\mathcal{S}'$-normality, and one hyperedge per input variable. Each constraint alone is easy; the conjunction is what makes the witness a legal input rather than a curiosity.
(3) *The justification of the assertion has two independent gaps*, located in Section 4.1: the normal-form property invoked, "No identity gates — sub-circuits $(\gamma \wedge \gamma)$ and $(\gamma \vee \gamma)$ do not occur", is stated in [3] for circuits "in normal form according to our system $\mathcal{S}$", while Algorithm 2 normalises under $\mathcal{S}'$, which [3] defines as $\mathcal{S}$ "identical except for omission of the collapse rule"; and the clause of Plump's Theorem 1.7.12 that the citation reaches concerns rewriting modulo bisimilarity, in which, by his Definition 1.4.9, "collapsing and copying are 'built in' ", whereas plain rewriting is governed by his Theorem 1.7.11, which assumes left-linearity and after which he writes "left-linearity cannot be dropped either" (the chapter attributes both theorems to Plump's joint work with Ariola and Klop [2]).

We do *not* claim that Theorem 12(2) or Theorem 13(2) of [3] is false. Read as a statement about bisimilarity classes it is Plump's Theorem 1.7.12(2), and $\mathcal{R}_B$ is convergent, so it holds. What we show is that Algorithm 2, which needs plain rewriting on concrete, collapse-free circuits for its running-time claim, has a legal input on which its stated invariant does not hold. Section 4.2 shows one pass of hash-consing restoring the blocked redex without changing the computed function or the gate count, and exhibits an explicit non-injective root-preserving hypergraph morphism out of C^* — so C^* collapses in the source's own sense, is $\mathcal{S}'$-normal and $\mathcal{S}$-abnormal, and the defect sits exactly at the omission of collapse. We make no claim about the cost of collapse, so the source's own reason for omitting it is left standing as a question.

Sections 3 and 5 record the surrounding material, all of it either the source's or classical, in machine-checked form: termination and confluence of $\mathcal{R}_B$ with unique normal forms, the fact that rewriting a formula with bits substituted evaluates it, $\mathrm{CC}_D(\mathsf{XOR}_2) = 3$ by a complete enumeration whose completeness is itself proved, and an explicit DeMorgan family of exactly $3(n-1)$ binary gates, verified to compute XOR_n for $n \leq 6$. We are not aware of an earlier proof-assistant development of gate elimination, of Schnorr's bound, or of any unrestricted Boolean circuit lower bound: the one formalized circuit lower bound we could find in a major library is Thiemann's monotone-circuit bound for Clique [12], and the closest term-graph work is Webb, Hayes and Utting's verification of term graph optimizations [13]. Schnorr's bound itself, $\mathrm{CC}_D(\mathsf{XOR}_n) \geq 3(n-1)$ for all n, is not proved here; Remark 5.4 records what is known computationally about its next cell.

2. Preliminaries

Formulas and the system $\mathcal{R}_B$. Let T be the set of terms with variables $x_1, x_2, \ldots$ over the DeMorgan signature $\{\wedge, \vee, \neg, 0, 1\}$, in which $\wedge$ and $\vee$ are binary, $\neg$ is unary and $0, 1$ are constants; we call these *DeMorgan formulas*. The system $\mathcal{R}_B$ of [3], Definition 6, consists of the following sixteen rules, in which g ranges over T (every rule has exactly one variable, so a substitution instance is just an arbitrary formula in that position):

normalizing:	$0 \to \neg 1$,	$\neg\neg g \to g$,
	$g \wedge g \to g$,	$g \vee g \to g$,
fixing:	$g \wedge \neg 1 \to \neg 1$,	$\neg 1 \wedge g \to \neg 1$,
	$g \vee 1 \to 1$,	$1 \vee g \to 1$,
passing:	$g \wedge 1 \to g$,	$1 \wedge g \to g$,
	$g \vee \neg 1 \to g$,	$\neg 1 \vee g \to g$,
tautology:	$g \wedge \neg g \to \neg 1$,	$\neg g \wedge g \to \neg 1$,
	$g \vee \neg g \to 1$,	$\neg g \vee g \to 1$.

We write $\to$ for the one-step rewrite relation of $\mathcal{R}_B$ (a rule applied under a context) and $\to^*$ for its reflexive–transitive closure. A rule $\ell \to r$ is *left-linear* when no variable occurs more than once in ℓ. Six rules of $\mathcal{R}_B$ are not: $g \wedge g \to g$, $g \vee g \to g$ and the four tautology rules.

Circuits as term graphs. Following [3] and [9], a DeMorgan circuit is a hypergraph $C = \langle V_C, E_C, \mathrm{lab}_C, \mathrm{att}_C \rangle$ whose edges carry labels from the signature together with the input variables, with $\mathrm{att}_C(e) = \mathrm{res}(e)\,\mathrm{args}(e)$ and $|\mathrm{args}(e)| = \mathrm{arity}(\mathrm{lab}_C(e))$, together with a distinguished node root_C; it is a *term graph* when every node is reachable from the root, the hypergraph is acyclic, and each node is the result node of a unique edge. We write $\mathrm{term}(C)$ for the DeMorgan formula obtained by unfolding the root, and $\mathrm{size}(C)$ for the number of $\{\wedge, \vee\}$-edges: negations and variable edges are free.

Condition (3) constrains *nodes*, not labels. Two distinct edges may carry the same label — two edges labelled x_1, or two edges labelled $\neg$ over one and the same argument node — and the resulting hypergraph is still a term graph.

Definition 2.1 (proper redex)**.** For a rule $\ell \to r$ of $\mathcal{R}_B$, let $\diamond\ell$ be the parse tree of ℓ with all occurrences of its variable collapsed to a single open vertex (Plump's notation). A pair $\langle v, \ell \to r \rangle$ is a *proper redex* of C when there is a hypergraph morphism $\diamond\ell \to C$ sending the root of $\diamond\ell$ to v. We call C $\mathcal{S}'$*-normal* when it has no proper redex, and write $\mathcal{S}$, $\mathcal{S}'$ for the two rewriting systems of [3], Theorem 13 — proper steps together with the collapse rule, and proper steps alone.

Remark 2.2. Definition 2.1 is our reading of the redex notion of [3] (its Definition 15, which specialises to circuits the term graph rewriting of its Definition 10, itself Plump's Definition 1.4.5), and it is decidable in the form we use. Every non-open node of $\diamond\ell$ is the result node of a unique edge, so a morphism, if it exists, is determined by following attachments from the root; the check therefore reduces, for each of the sixteen rules, to a comparison of labels along the pattern together with node-identity tests wherever the pattern variable is repeated. For the six non-left-linear rules, the two positions carrying g must be mapped to *one* node — this is exactly the phenomenon of Plump's Example 1.4, and it is the only place where the graph-level check is stricter than the formula-level one.

Straight-line programs and CC_D**.** For Section 5 we use the equivalent presentation of DeMorgan circuits with free negations as straight-line programs: a circuit on n inputs of size s is a list of s gates, each an $\wedge$ or an $\vee$ of two earlier signals with an optional negation on each input, plus an output index and an optional output negation. Collapsing a chain of $\neg$ gates into polarity bits is the standard normalisation, and we take the above as the definition of the model; the results of Section 5 should be read as statements about it. We write $\mathrm{CC}_D(f)$ for the least s such that some circuit of that form computes f, so that $\mathrm{CC}_D(\mathsf{XOR}_n) \geq 3(n-1)$ is Schnorr's bound [10].

3. The system $\mathcal{R}_B$

The results of this section are the source's (Lemma 7 of [3]) or elementary; we record them because everything in Section 4 is a statement about $\mathcal{R}_B$-reducibility and must rest on a checked system. Two of the proofs differ from the source's certificate in ways that matter only for the mechanisation, and we say what they are.

Theorem 3.1 (termination)**.** *Define $W : T \to \mathbb{N}$ by $W(0) = 3$, $W(1) = W(x_i) = 1$, $W(\neg t) = 1 + W(t)$ and $W(t \wedge u) = W(t \vee u) = 1 + W(t) + W(u)$. If $t \to u$ then $W(u) < W(t)$. Consequently $\to$ is well founded, and there is no infinite reduction sequence $t_0 \to t_1 \to \cdots$.*

Proof sketch. W is additive over contexts, so it suffices to check the sixteen root rules. Writing $w = W(g) \geq 1$, the drop $W(\ell) - W(r)$ is: 1 at $0 \to \neg 1$; 2 at $\neg\neg g \to g$, $g \wedge 1 \to g$ and $1 \wedge g \to g$; 3 at $g \vee \neg 1 \to g$ and $\neg 1 \vee g \to g$; $1 + w$ at $g \wedge g \to g$, $g \vee g \to g$ and the four fixing rules; $2w$ at $g \wedge \neg g \to \neg 1$ and $\neg g \wedge g \to \neg 1$; and $1 + 2w$ at $g \vee \neg g \to 1$ and $\neg g \vee g \to 1$. Every one of the sixteen is positive — the two with drop $2w$ because $W \geq 1$, the rest outright; the least drop is 1, at $0 \to \neg 1$

alone, then 2 at eleven rules and 3 at the remaining four. Well-foundedness follows by pulling back the order on $\mathbb{N}$, and $W(t_n) + n \le W(t_0)$ gives the second sentence. □

Remark 3.2. The weight is a replacement for the recursive path order used in the certificate shipped with [1] (precedence $1 < \neg < 0 = \wedge$), not an improvement on it; its only virtue is that it needs no order machinery. Note that a plain symbol count does not work: the rule $0 \to \neg 1$ increases it, as [3] itself observes, and the weight $w(0) = 2$ makes that rule tie.

Theorem 3.3 (confluence and unique normal forms)**.** *Define* $\mathrm{nf} : T \to T$ *bottom-up by* $\mathrm{nf}(0) = \neg 1$, $\mathrm{nf}(1) = 1$, $\mathrm{nf}(x_i) = x_i$, $\mathrm{nf}(\neg t) = \sigma_\neg(\mathrm{nf}(t))$, $\mathrm{nf}(t \wedge u) = \sigma_\wedge(\mathrm{nf}(t), \mathrm{nf}(u))$ *and* $\mathrm{nf}(t \vee u) = \sigma_\vee(\mathrm{nf}(t), \mathrm{nf}(u))$*, where the three smart constructors apply the corresponding root rules if they match and build the node otherwise. Then* $\mathrm{nf}(t)$ *is irreducible,* $t \to u$ *implies* $\mathrm{nf}(t) = \mathrm{nf}(u)$*, and* $t \to^* \mathrm{nf}(t)$*. Consequently* $\to$ *is confluent, and any two irreducible terms reachable from a common term are equal.*

Proof sketch. The three displayed properties are proved by induction on t and on the derivation of the step; because nf is a fold, the context cases of the second are immediate and the sixteen root cases are one identity each about the smart constructors. Confluence follows by joining at $\mathrm{nf}(t)$. Neither the Critical Pair Lemma nor Newman's lemma is used; this is the route rather than the result, and the result is the source's Lemma 7, whose certificate in [1] closes local confluence by joining all critical pairs (that file states no count; an independent enumeration of the ordered overlaps finds 61, all joinable). □

Theorem 3.4 (soundness, and evaluation)**.** *Let* eval_ρ *be the Boolean value of a formula under an assignment* ρ *of the variables. If* $t \to u$ *then* $\mathrm{eval}_\rho(t) = \mathrm{eval}_\rho(u)$ *for every* ρ*: every rule of* $\mathcal{R}_B$ *is a Boolean identity, so rewriting preserves the function computed. Moreover, if* t *is closed then* $\mathrm{nf}(t) \in \{1, \neg 1\}$*, and* $\mathrm{nf}(t) = 1$ *if and only if* $\mathrm{eval}_\rho(t) = \mathrm{true}$*.*

Proof sketch. The first claim is sixteen Boolean identities plus congruence. For the second, closedness is preserved by nf, and the only closed irreducible terms are 1 and $\neg 1$; combine with $\mathrm{eval}_\rho(\mathrm{nf}(t)) = \mathrm{eval}_\rho(t)$. This is the sentence of [3], §3 — "given a circuit C with bits $b = b_1, \dots, b_n$ substituted for all input variables, rewriting C using $\mathcal{S}$ until termination just evaluates C on input b" — which is stated there without proof. □

Proposition 3.5 (controls on the rule set)**.** *(1)* $0 \to \neg 1$ *and* $|0| < |\neg 1|$ *in plain symbol count, so the plain symbol count does not prove Theorem 3.1; and the weight with* $w(0) = 2$ *gives equal values on that rule while* W *decreases. (2)* $\mathcal{R}_B$ *extended by commutativity of* $\wedge$ *is not terminating: for distinct variables* x, y*, the formula* $x \wedge y$ *is not even accessible for the extended relation, although* W *takes the same value on* $t \wedge u$ *and* $u \wedge t$ *and so cannot detect this. (3)* nf *does not collapse everything:* $\mathrm{nf}(x \wedge y) = x \wedge y$*, which is neither* 1 *nor* $\neg 1$*, while* $x \wedge x$ *is reducible and* $x \wedge y$ *is not.*

Proof sketch. (1) and (3) are evaluations of the definitions. (2) is the two-cycle $x \wedge y \to y \wedge x \to x \wedge y$: no element of a two-cycle is accessible. The source excludes commutativity for a different reason, that it makes confluence impossible (citing [11]); non-termination is stronger and elementary. □

4. An $\mathcal{S}'$-normal circuit whose formula is reducible

Definition 4.1. Let C^* be the term graph on ten nodes $0, \dots, 9$ with root 9 and edges

$$\begin{array}{llll} 0 : x_1, & 1 : \neg(0), & 2 : \neg(0), & 3 : x_2, \\ 4 : x_3, & 5 : x_4, & 6 : \wedge(1,2), & 7 : \vee(6,3), \\ 8 : \wedge(7,4), & 9 : \vee(8,5), & & \end{array}$$

where $v : \lambda(w_1, \dots)$ means that v is the result node of an edge labelled λ with those arguments. Nodes 1 and 2 are two *distinct* $\neg$-edges over the same node 0. Let C^{**} be the nine-node variant in which the duplication is at an input instead: two distinct edges labelled x_1 with result nodes 0 and 1, then $2 : x_2$, $3 : x_3$, $4 : x_4$, $5 : \wedge(0,1)$, $6 : \vee(5,2)$, $7 : \wedge(6,3)$, $8 : \vee(7,4)$, root 8.

Theorem 4.2. *C^* has the following properties.*

(1) *It is a term graph: every node is reachable from the root, it is acyclic, each node is the result node of a unique edge, and every edge's attachment length matches its label's arity.*
(2) *It reads the variables x_1, x_2, x_3, x_4, all four of them, and $\mathrm{size}(C^*) = 4$. With $n = 4$ we have $n > 3$ and $\mathrm{size}(C^*) < 3(n-1) = 9$, so C^* meets the* `Require` *line of Algorithm 2 of* [3].
(3) *No proper redex of $\mathcal{S}'$ exists at any of its ten nodes: C^* is $\mathcal{S}'$-normal, and Normalize$(C^*) = C^*$.*
(4) $\mathrm{term}(C^*) = (((\neg x_1 \wedge \neg x_1) \vee x_2) \wedge x_3) \vee x_4$, *and this formula is $\mathcal{R}_B$-reducible; explicitly,*

$$\mathrm{term}(C^*) \;\to\; ((\neg x_1 \vee x_2) \wedge x_3) \vee x_4$$

by $g \wedge g \to g$ applied under three contexts.
(5) *Each of x_1, x_2, x_3, x_4 is carried by exactly one hyperedge of C^*.*

The variant C^{**} satisfies (1), (2) and (3) as well, and $\mathrm{term}(C^{**}) = (((x_1 \wedge x_1) \vee x_2) \wedge x_3) \vee x_4$ *is likewise $\mathcal{R}_B$-reducible; it fails (5) by construction, two of its edges being labelled x_1.*

Proof sketch. (1), (2), (3) and (5) are finite checks on a ten-node hypergraph, decided by evaluation of the definitions. The check behind (3) is the only one worth quantifying: the redex predicate of Definition 2.1 is evaluated at each of the ten nodes against all sixteen rules of $\mathcal{R}_B$, that is, 160 pattern tests, and every one fails. The single interesting failure is at node 6: the pattern $\diamond(g \wedge g)$ requires the two argument positions to be mapped to one node, and $\mathrm{att}(6) = 6\,1\,2$ with $1 \neq 2$. In (4) the unfolding is computed by the definition and the displayed step is exhibited term by term, so no search is involved. □

Theorem 4.3. *Let h be the topologically minimal $\{\wedge, \vee\}$-gate of C^*, as selected on line 7 of Algorithm 2 of* [3]. *Then h is well defined and unique: the binary gates of C^* are $6, 7, 8, 9$; node 6 is the only one of them with no binary gate below it, and each of $7, 8, 9$ has node 6 below it, so 6 comes first in every topological order. Its two arguments are the distinct nodes 1 and 2, and each of them represents $\neg x_1$. Consequently, for all $b_1, b_2 \in \{0, 1\}$ and all p, q, if the first argument of h represents $(\neg)^{b_1} x_p$ and the second represents $(\neg)^{b_2} x_q$, then $p = q$. The assertion on line 8 of Algorithm 2 — "h has inputs $(\neg)x_p, (\neg)x_q$ for $p, q \in V$ and $p \neq q$", justified in* [3] *by "the assertion holds because C_i is in normal form" — therefore fails on the legal input C^*, and the algorithm as printed has no next move.*

Proof sketch. All the data are finite: the list of binary gates, the set of binary gates reachable below each of them, and the reading of nodes 1 and 2 as $(\neg)x_i$ are computed from Definition 4.1. Both readings return $\neg x_1$, so any p and q of the asserted shape equal 1. By Theorem 4.2(2), (3), C^* passes the `Require` line, is returned unchanged by the normalisation on line 1, enters the loop ($|V| = 4 > 2$), and passes the degeneracy test on lines 4–5, since all four variables are read; line 6 asserts normality and non-degeneracy, which lines 1 and 4–5 have just supplied, so line 8 is reached. □

Remark 4.4 (line numbering). The lines of Algorithm 2 are numbered here as they are numbered in the PDF compiled from the v1 e-print source, where the `algorithmic` environment numbers comment lines too: line 7 selects h, line 8 is the $p \neq q$ assertion, line 9 is its justifying comment, and line 10 is the fanout test. This agrees with the paper's own prose at line 7 ("in line 7, the algorithm selects the topologically minimal binary gate"); its later phrase "If this is not the case on line 9" refers to the fanout test, which the printed numbering puts at line 10. Nothing below depends on the numbering: the assertion is identified by its text.

4.1. **Where the justification does not carry.** The assertion is justified in one clause, "because C_i is in normal form". Two independent things go wrong with that clause, and neither is visible from the algorithm alone.

First, the property is a property of $\mathcal{S}$, and the algorithm normalises under $\mathcal{S}'$. The list of structural consequences of normality in [3], §3, is introduced by "circuits C in normal form according to our system $\mathcal{S}$ will have", and its third item is "No identity gates — sub-circuits $(\gamma \wedge \gamma)$ and $(\gamma \vee \gamma)$ do not

occur". That item is exactly what line 8 needs. But $\mathcal{S}'$ is defined in the same paper as $\mathcal{S}$ "identical except for omission of the collapse rule", and it is $\mathcal{S}'$ that Algorithm 2 uses, by an explicit choice: "it is not clear if the collapse rule can be implemented efficiently. Therefore, we use the plain rewriting system (without collapse rule) when constructive arguments are required". On C^* the sub-circuit at node 6 *is* an identity gate as a formula — term of both its arguments is $\neg x_1$ — and is not one as a hypergraph, because the two arguments are distinct nodes.

Second, the citation reaches a statement about bisimilarity classes. Theorem 12 of [3] is attributed to "Corollary 1.7.4 and Theorem 1.7.12 of [9]". In Plump's chapter, Corollary 1.7.4 is about $\Rightarrow_{coll}$, rewriting *with* collapse, and supports clause (1). Theorem 1.7.12 states the equivalence of the confluence of $\Rightarrow_{bi}$, of $\Rightarrow_\sim$ and of $\to$; and $\Rightarrow_\sim$ is, by his Definition 1.4.9, the relation on bisimilarity *classes*, introduced with the words "rewriting modulo bisimilarity, where collapsing and copying are 'built in' in the sense that rewrite steps transform bisimilarity classes rather than term graphs". The result that governs plain $\Rightarrow$ is his Theorem 1.7.11:

> If $\mathcal{R}$ is left-linear, $\to$ confluent and $\Rightarrow$ normalizing, then $\Rightarrow$ is confluent modulo bisimilarity.

immediately after which he writes "Here normalization of $\Rightarrow$ cannot be relaxed to normalization of $\to$, as is witnessed by Example 1.9, and left-linearity cannot be dropped either." Six of the sixteen rules of $\mathcal{R}_B$ are not left-linear.

Neither observation contradicts Theorem 12(2) or Theorem 13(2) of [3]: on the bisimilarity-class reading, Theorem 12(2) *is* Plump's Theorem 1.7.12(2), and $\mathcal{R}_B$ is convergent by Theorem 3.3, so the statement holds. What Theorem 4.3 shows is that the passage from that statement to a structural invariant of a concrete, collapse-free circuit — which is what Algorithm 2 makes — is where the argument breaks, and that it breaks on an input the algorithm is required to accept.

4.2. **The repair, and controls.**

Proposition 4.5. *Let* $\mathrm{hc}(C)$ *be the result of one hash-consing pass over* C*: in increasing node order, redirect a node to the least node carrying the same label and the same already redirected argument list, then delete the redirected edges and rewrite the remaining attachments. Then:*

1. $\mathrm{hc}(C^*)$ *is a term graph, and node* 6 *is now a proper redex, its arguments being the single node* 1 *twice; the rule* $g \wedge g \to g$ *applies there.*
2. $\mathrm{term}(\mathrm{hc}(C^*)) = \mathrm{term}(C^*)$ *and* $\mathrm{size}(\mathrm{hc}(C^*)) = \mathrm{size}(C^*)$*: the repair changes neither the function computed nor the number of binary gates, and removes exactly one edge, a free negation.*
3. *The map* $v \mapsto 1$ *if* $v = 2$*,* $v \mapsto v$ *otherwise, is a hypergraph morphism* $C^* \to \mathrm{hc}(C^*)$*; it sends root to root and is not injective.*

For C^{**}*, the analogue of (1) and the first clause of (2) are checked at its gate node* 5*.*

Proof sketch. All three are finite checks on the two ten- and nine-node hypergraphs and their images. For (3), a morphism must preserve labels and attachments; the map is checked edge by edge. □

Part (3) is the sharp statement: it says that C^* *collapses* in the sense of [3] — there is a non-injective root-preserving hypergraph morphism out of it — so C^* is $\mathcal{S}'$-normal and $\mathcal{S}$-abnormal, and the defect is located exactly at the omission of collapse rather than anywhere else in the construction.

Remark 4.6 (what Proposition 4.5 does not claim). The pass above is a single fold with a linear scan, hence quadratic as written. Full collapse to the maximally shared form is ordinary hash-consing in topological order, which is the standard near-linear procedure with a hash table, but we prove no complexity theorem and claim none; the source's remark that "it is not clear if the collapse rule can be implemented efficiently" is left standing as a question. What Proposition 4.5 does say is that on this witness the obstruction is removed by a step of that kind at no cost in gates, so Theorem 19 of [3] may well be repairable by normalising under $\mathcal{S}$, or by inserting a sharing pass before line 7.

Proposition 4.7 (controls)**.** (1) A left-linear rule does lift on the same shape. *Let D be the term graph with edges $0:1$, $1:1$ (two distinct edges labelled by the constant 1), $2:x_2$, $3:\wedge(0,1)$, $4:\vee(3,2)$, root 4. Then $\langle 3, 1\wedge g \to g\rangle$ is a proper redex, so D is not $\mathcal{S}'$-normal, although D has the same duplication pattern as C^*. The difference is that the left-hand side $1\wedge g$ does not repeat a variable.*

(2) The assertion is not always false. *There is a four-input $\mathcal{S}'$-normal term graph of size 3 whose formula is $\mathcal{R}_B$-irreducible and whose unique topologically minimal binary gate reads x_1 and x_2, so line 8 holds there with $p = 1 \neq 2 = q$.*

(3) *Consequently the implication "C $\mathcal{S}'$-normal $\Rightarrow$ term(C) $\mathcal{R}_B$-irreducible" is false, by C^*; and its negation "C $\mathcal{S}'$-normal $\Rightarrow$ term(C) $\mathcal{R}_B$-reducible" is false as well, by (2). Some circuits do have proper redexes, for instance D and* hc(C^*).

Proof sketch. Finite checks on the displayed hypergraphs, together with Theorem 4.2(3), (4) for the two refutations in (3). □

Part (1) is what isolates non-left-linearity as the cause: it is not the presence of two edges with the same label that blocks the rewrite, but the requirement, forced by a repeated pattern variable, that they be one node.

5. Two finite cells of Schnorr's bound

This section records, in checked form, two classical facts about the quantity that Algorithm 2 exists to serve. Neither is new: the lower bound is Schnorr's [10] at $n = 2$, and the upper bound is the folklore three-gate XOR gadget. We include them because [3] calls the bound tight without exhibiting a circuit, and because a lower bound obtained by search is only a lower bound if the search is proved exhaustive.

Theorem 5.1 (the bound is attained)**.** *Let* $\mathrm{XOR}(u,v) = (u\vee v)\wedge\neg(u\wedge v)$*, three binary gates, and let X_n be the circuit chaining $n-1$ such gadgets. Then X_n is well formed and* $\mathrm{size}(X_n) = 3(n-1)$*, for every $n \geq 1$; and X_n computes* XOR_n *for $n = 1,\ldots,6$.*

Proof sketch. The size and the well-formedness are proved for all n by induction on the number of gadgets; no computation enters. The functional claim is by exhaustive evaluation on all 2^n inputs for each $n \leq 6$, that is 126 evaluations of circuits of up to 15 gates; for $n \leq 4$ this is decided by kernel reduction and for $n = 5, 6$ by compiled evaluation. □

Theorem 5.2. $\mathrm{CC}_D(\mathsf{XOR}_2) = 3$*. Explicitly: X_2 is a well-formed three-gate circuit computing* XOR_2*; and no well-formed circuit on two inputs with at most two binary gates computes* XOR_2*, or its negation.*

Proof sketch. The upper bound is Theorem 5.1 at $n = 2$. For the lower bound, let E be the explicit list of all circuits on two inputs with at most two gates, formed by taking every admissible gate list of length 0, 1 or 2, every output index, and both output polarities; E has 18,628 members. The load-bearing step is that E is *complete*: every well formed circuit on two inputs with at most two binary gates is a member of E, which is proved by case analysis on the gate list and does not rest on the enumeration itself. It is then checked, by compiled evaluation over all 18,628 members and all four inputs, that none computes XOR_2 and none computes its negation. Without the completeness step this would be a search report rather than a lower bound. □

Proposition 5.3 (controls on the enumeration)**.** *E has exactly* 18,628 *members; it contains circuits computing $x_1\wedge x_2$, and one gate suffices for that function, so the exhaustive negative of Theorem 5.2 is not vacuous. Three gates are not impossible: X_2 witnesses that Theorem 5.2 cannot be strengthened to circuits of size at most* 3*.*

Proof sketch. Evaluations over E and over the displayed one-gate circuit. □

Remark 5.4 (the next cell, outside the formal development). We record, as a computation carried out outside the machine-checked development and stated here without the status of a theorem, that $\mathrm{CC}_D(\mathsf{XOR}_3) = 6$, which is Schnorr's bound at $n = 3$ and again tight. Three independent computations agree: an exhaustive search over complement-closed classes of truth tables of three-variable functions finds XOR_3 unreachable within five gates after 21,030,568 nodes and reachable within six; the same search, run inside the proof assistant's interpreter, returns the same negative answer at budget five; and a SAT encoding of "size ≤ 5 suffices" is unsatisfiable while its size-six analogue is satisfiable. Turning this into a theorem needs a simulation lemma relating well-formed circuits of size s to derivations of length s in the class search, which we have not proved. The case $n = 4$ is out of reach of the same method.

6. Verification

Every theorem and proposition above is machine-checked in Lean 4 [8] — the one statement that is not, Remark 5.4, says so in its first line — in a development of about 2,150 lines that imports no library beyond Lean's core — in particular not Mathlib — and contains no `sorry` and no hand-written axiom; formulas, the rewrite relation, the weight, the normaliser, hypergraphs, the redex predicate of Definition 2.1, the collapse pass and the straight-line circuit model are all defined from scratch, and the three modules together check in under a minute on one core. Sections 3 and 4 are kernel-proved: the axiom sets of Theorems 3.1, 3.3, 3.4, 4.2, 4.3 and of Propositions 3.5, 4.5, 4.7 contain only propositional extensionality, quotient soundness and, in the case of Theorem 3.3, choice — no compiled evaluation is involved anywhere in the term-graph results, so the finite checks behind Theorems 4.2 and 4.3 are carried out by the kernel itself. Compiled evaluation is used only in Section 5, in three places: the functional check of Theorem 5.1 at $n = 5, 6$, the sweep over E behind Theorem 5.2, and the two counts of Proposition 5.3; Theorem 5.2 depends on exactly one such axiom, and its completeness lemma on none. Independently of the Lean development, the sixteen rules of $\mathcal{R}_B$ were read out of the Knuth–Bendix output file in the authors' supplementary artifact [1] by an XML parse and agree with Definition 6 of [3] rule for rule; and the witness of Definition 4.1, its normality, its unfolding and its repair were recomputed in an independent script before being stated in Lean. The repository is private; repository reference to be added at submission.

References

[1] M. Carmosino, N. Dang and T. Jackman, supplementary artifact for [3], `https://feasible-math.org/2026-FSCD-80-supplementary.zip`; retrieved 3 September 2026 (12,149 bytes, ten files, containing no proof-assistant sources). Its Knuth–Bendix output file for the DeMorgan basis parses to exactly the sixteen rules of Definition 6 of [3], identical to the printed system up to the name of the variable in the last rule, and a sibling file holds a thirteen-rule system for the $\{\wedge, \oplus, \neg\}$ basis. The CPF files are the completion proofs for CeTA [5], whose termination step is a recursive path order with precedence $1 < \neg < 0 = \wedge$.

[2] Z. M. Ariola, J. W. Klop and D. Plump, *Confluent rewriting of bisimilar term graphs*, in: Proc. Fourth Workshop on Expressiveness in Concurrency, Electronic Notes in Theoretical Computer Science 7, Elsevier, 1997; DOI `10.1016/S1571-0661(05)80463-3`. And *Bisimilarity in term graph rewriting*, Information and Computation **156** (2000), no. 1–2, 2–24; DOI `10.1006/inco.1999.2824`. These are references [7] and [6] of [9], cited there as the sources of its Theorems 1.7.11 and 1.7.12. The chapter lists the second as forthcoming, with no volume, year or pages; the data given here are from Crossref, checked 3 September 2026.

[3] M. Carmosino, N. Dang and T. Jackman, *Convergent Gate Elimination and Constructive Circuit Lower Bounds*, arXiv:2602.17942v1 (20 February 2026), cs.CC. Read on 3 September 2026; v1 is the only version, and the listing carries no journal reference, no external DOI and no comments field. All quotations in this paper are from that version. Its results are numbered continuously, with no section prefix; the numbers used here (Definition 6, Lemma 7, Theorem 12, Theorem 13, Theorem 18, Theorem 19, Algorithm 2) and the line numbers of Algorithm 2 were read off the PDF compiled from the v1 e-print source. Each cited result is in addition identified by a verbatim quotation, which is what the arguments here actually turn on.

[4] M. Carmosino, N. Dang and T. Jackman, *Constructive Separations from Gate Elimination*, arXiv:2604.23958v1 (27 April 2026), cs.CC. A successor by the same authors which carries a full refuter for XOR_n without the term-graph apparatus; it does not cite [3]. Nothing in this paper bears on it.

[5] R. Thiemann and C. Sternagel, *Certification of termination proofs using CeTA*, in: Theorem Proving in Higher Order Logics (TPHOLs 2009), Lecture Notes in Computer Science 5674, Springer, 2009, pp. 452–468; DOI 10.1007/978-3-642-03359-9_31. This is the entry for CeTA used here; [3] names CeTA but attaches to the name a different paper, C. Sternagel and R. Thiemann, *Formalizing Knuth–Bendix orders and Knuth–Bendix completion*, RTA 2013, LIPIcs 21, pp. 287–302.

[6] D. E. Knuth and P. B. Bendix, *Simple word problems in universal algebras*, in: Computational Problems in Abstract Algebra (J. Leech, ed.), Pergamon Press, 1970, pp. 263–297.

[7] T. Sternagel and H. Zankl, *KBCV — Knuth-Bendix completion visualizer*, in: Automated Reasoning (IJCAR 2012), Lecture Notes in Computer Science 7364, Springer, 2012, pp. 530–536.

[8] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, in: Automated Deduction — CADE 28, Lecture Notes in Computer Science, Springer, 2021, pp. 625–635; DOI 10.1007/978-3-030-79876-5_37.

[9] D. Plump, *Term graph rewriting*, Chapter 1 of: H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg (eds.), Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools, World Scientific, 1999, pp. 3–61; DOI 10.1142/9789812815149_0001. All quotations here are from the accepted author manuscript, read in full on 3 September 2026 at https://pure.york.ac.uk/ws/portalfiles/portal/158746750/author_manuscript.pdf; its own pagination runs from 3 to 61, the printed chapter's page range, and its numbering is the numbering used above. Example 1.4, Definition 1.4.9, Corollary 1.7.4, Theorem 1.7.11 and Theorem 1.7.12 (printed pages 21, 24, 40 and 44) were checked on the rendered pages rather than on extracted text, because the subscripts on $\Rightarrow$ do not survive text extraction: clause (2) of Theorem 1.7.12 reads $\Rightarrow_{\sim}$, the relation of Definition 1.4.9, and not plain $\Rightarrow$.

[10] C. P. Schnorr, *Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen*, Computing **13** (1974), no. 2, 155–171; DOI 10.1007/BF02246615 (author, title, volume, issue, pages and DOI confirmed against Crossref on 3 September 2026).

[11] R. Socher-Ambrosius, *Boolean algebra admits no convergent term rewriting system*, in: Rewriting Techniques and Applications (RTA-91), Lecture Notes in Computer Science 488, Springer, 1991, pp. 264–274. Cited by [3] as its reason for excluding commutativity from $\mathcal{E}_B$.

[12] R. Thiemann, *Clique is not solvable by monotone circuits of polynomial size*, Archive of Formal Proofs, 2022, https://isa-afp.org/entries/Clique_and_Monotone_Circuits.html.

[13] B. J. Webb, I. J. Hayes and M. Utting, *Verifying term graph optimizations using Isabelle/HOL*, in: Proc. 12th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2023), ACM, 2023, pp. 320–333.