A novel EGs-based framework for systematic propositional-formula simplification

Jordina Francès de Mas[1][0009−0002−6608−4051] and Juliana Bowles[1,2][0000−0002−5918−9114]⋆

[1] School of Computer Science, University of St Andrews, Scotland, UK
{jfdm2,jkfb}@st-andrews.ac.uk
[2] Software Competence Centre Hagenberg (SCCH), Austria

Abstract. This paper presents a novel simplification calculus for propositional logic derived from Peirce's Existential Graphs' rules of inference and implication graphs. Our rules can be applied to arbitrary propositional logic formulae (not only in CNF), are equivalence-preserving, guarantee a monotonically decreasing number of clauses and literals, and maximise the preservation of structural problem information. Our techniques can also be seen as higher-level SAT preprocessing, and we show how one of our rules (TWSR) generalises and streamlines most of the known equivalence-preserving SAT preprocessing methods. We further show how this rule can be extended with a novel n-ary implication graph to capture all known equivalence-preserving preprocessing procedures. Finally, we discuss the complexity and implementation of our framework as a solver-agnostic algorithm to simplify Boolean satisfiability problems and arbitrary propositional formula.

Keywords: Simplification · Existential graphs · Equivalence-preserving preprocessing · Propositional satisfiability · Diagrammatic reasoning · Knowledge representation · Implication graphs · Hypergraphs.

1 Introduction

Propositional logic simplification is closely related to the reduction of complex Boolean algebraic expressions, logic circuits' minimisation, and Boolean satisfiability (SAT) preprocessing techniques. Simplification is crucial to reduce the complexity of a problem, which makes it easier to understand, reason about, and work with. Minimising the size of a problem also reduces memory usage and speeds up solving times [30], as fewer steps are required to reach a solution or proof. Unfortunately, existing minimisation algorithms for Boolean expressions such as Karnaugh maps [18] or the Quine–McCluskey algorithm [22] become increasingly inefficient as the number of variables grows, making them unusable for large formulas. To minimise larger Boolean problems, we can only resort to suboptimal heuristic methods, such as the ESPRESSO logic minimiser [27].

⋆ Bowles is partially supported by Austrian FWF Meitner Fellowship M-3338 N.

Intuitively, all equivalence-preserving simplifications can also be achieved by applying the axioms of Boolean algebra or rules of inference in nontrivial ways, or by utilising proof systems such as sequent calculus or natural deduction; however, there are currently no known generic or systematic methods of doing so.

Many simplification techniques in the field of SAT use heuristics too, but a greater emphasis is placed on efficiency so that they can be used on very large problems. Preprocessing has actually become an essential part of SAT solving, with some of the most powerful SAT solvers interleaving preprocessing with solving steps, a technique referred to as *inprocessing* [17,12]. Despite the already vast body of literature on preprocessing (see Chapter 9 in the latest Handbook of Satisfiability [7]), there is still much ongoing research into finding efficient rewriting and simplification techniques to expedite or better inform the solving phase (see, e.g., [1,20,6,21]), which evidences the importance and complexity of this topic. However, SAT problems commonly need to be translated into a standard form (usually conjunctive normal form (CNF)) before solving, and most procedures apply to this form only. Consequently, most preprocessing techniques in the literature only study or work on CNF formulae. Moreover, this encoding process can be non-equivalence-preserving, result in bigger problems, or lead to the loss of important structural properties of the original problem, such as symmetries, which can detrimentally impact the preprocessing and solving phases [11,1].

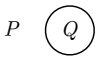
In this paper, we present novel simplification techniques for zeroth-order logic derived from Peirce's existential graphs' rules of inference and implication graphs. Our rules can be seen as SAT equivalence-preserving preprocessing techniques applicable to arbitrary propositional-logic formulae (not only in CNF) that guarantee a monotonically decreasing number of variables, clauses and literals and maximise the preservation of structural problem information. Existential graphs offer a fresh view of preprocessing never explored before that allowed us to independently rediscover many simplification techniques known in the world of SAT preprocessing, understand the underlying connections between apparently distinct methods, and generalise many of them. We compare our rules with the state-of-the-art and discuss their advantages and limitations. In particular, our last rule (TWSR) can efficiently emulate complex combinations of preprocessing techniques, and we propose even more powerful extensions to it.

The remainder of our paper is structured as follows. Section 2 introduces basic concepts and notation on existential graphs, propositional logic and implication graphs that will be used in the rest of the paper. In Section 3, we present our simplification rules and explain their properties. In Section 4, we discuss the implementation of our approach and point to future work, including a generalisation of our TWSR rule. Section 5 ends the paper with some concluding remarks.

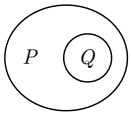
2 Background and notation

Modern formal logics, including formulations of SAT problems, are usually presented in a symbolic and linear fashion. **Existential graphs** (EGs) [31,29], by

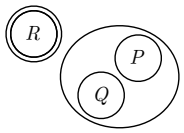
contrast, are a *non-symbolic* and *non-linear* system for logical expressions, where propositions are drawn on a two-dimensional sheet, negation is represented by oval lines (aka *cuts*), and all elements contained in the same area (delimited by negation lines) are in implicit conjunction. We say that an area is *evenly* (resp. *oddly*) *nested* if it is enclosed by an even (resp. odd) number of cuts. Note that a blank sheet of assertion denotes *true* ($\top$), has nesting level 0 and therefore it is assumed to be evenly enclosed. In Fig. 1, we present a few introductory examples illustrating that the combination of these simple primitives suffices to express any propositional-logic formula. Extensions of EGs allow for the representation of first-order-logic and higher-order-logic formulas, but these are beyond the scope of this paper (for more details, see e.g. [9,26]).



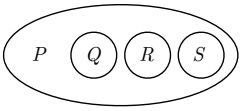
(a) EG of $P \wedge \neg Q$ or, equivalently, $\neg Q \wedge P$. This EG is also the canonical form of infinitely many equivalent formulas, such as $\neg(P \rightarrow Q)$ or $\neg(\neg(\neg P \vee Q))) \wedge (R \vee \neg R)$.



(b) EG of $\neg(P \wedge \neg Q)$ or, equivalently, $\neg P \vee Q$, or $P \rightarrow Q$.



(c) EG of $\neg(\neg R) \wedge \neg(\neg Q \wedge \neg P)$ or, equivalently, $(P \vee Q) \wedge R$, or $R \wedge (\neg P \rightarrow Q)$, or $R \wedge (\neg Q \rightarrow P)$.



(d) EG of $\neg(P \wedge \neg Q \wedge \neg R \wedge \neg S)$, which can be linearly represented by many equivalent formulations, such as $P \rightarrow (Q \vee R \vee S)$, or $(P \wedge \neg Q) \rightarrow (R \vee S)$, or $(P \wedge \neg Q \wedge \neg R) \rightarrow S$, or $\neg P \vee Q \vee R \vee S$.

Fig. 1: Examples of existential graphs of propositional-logic formulae.

EGs cannot only represent propositional-logic formulae, but they are, in fact, equivalent to sentential languages [25,33,26]. Even if this diagrammatic notation is less commonly used, it offers a simple, elegant, and easy-to-learn alternative that comes with a sound and complete deductive system, which has many advantages over the traditional Gentzen's rules of *natural deduction* and *sequent calculus* and, thus, over all other rewriting methods based on the latter. Most significantly, EGs' inference rules are symmetric, and no bookkeeping of previous steps is required in the process of inferring a graph from another. Furthermore, as illustrated in Fig. 1, EGs' graphical non-ordered nature provides a canonical representation of a logical formula that, in linear notation, can take many equivalent forms. Another key and perhaps underexplored advantage is that the visual representation allows for the recognition of patterns that would otherwise be obscured in nested clausal form. Finally, since EGs are easily transferred to any notation, they can help clarify the relationships between diverse reasoning

methods, such as resolution and natural deduction and, ultimately, help understand the foundations of proof theory.

We assume the reader is familiar with basic notions of *propositional logic* and the *Boolean satisfiability problem* (**SAT**). In what follows, we will use a subscript to indicate the size of a clause (e.g. C_8 refers to a clause of size 8) and, given a clause $C_n = (l_1 \vee l_2 \vee \cdots \vee l_n)$, we refer to the set of its literals as $\text{lit}(C_n) = \{l_1, l_2, \cdots, l_n\}$.

As shown in Fig. 1, nested cuts can be interpreted as implication, so it is easy to see that every binary clause, e.g. $C_2 = (x \vee y)$, admits the following two interpretations: $(\neg x \to y)$ and its logically equivalent contrapositive $(\neg y \to x)$. Thus, binary clauses provide the information needed to build what is known as the **binary implication graph** (BIG) [2] of the formula, which is a directed graph where edges represent implication, and nodes are all the literals occurring in binary clauses and their negations (see Fig. 2).

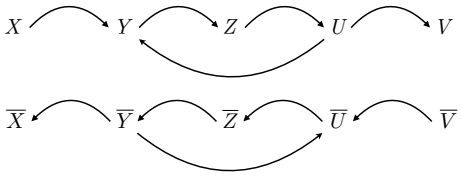


Fig. 2: BIG of $\varphi = (\overline{X} \vee Y) \wedge (\overline{Y} \vee Z) \wedge (\overline{Z} \vee U) \wedge (\overline{U} \vee V) \wedge (\overline{U} \vee Y)$.

If there exists a directed path from vertex x to vertex y, we say that y is a **descendant** of (or *reachable* from) x, and we say that x is an **ancestor** of y. We will denote the set of all descendants (resp. ancestors) of a vertex z by $\text{des}(z)$ (resp. $\text{anc}(z)$). We say that two vertices u and v in a directed graph G are strongly connected iff there are directed paths from u to v and from v to u. A subgraph $S \subseteq G$ is a **strongly connected component** if all its vertices are strongly connected and they are not strongly connected with any other vertex of $G \setminus S$ (i.e. every vertex in S is reachable from every other vertex in S and S is maximal with this property wrt G). Every strongly connected component corresponds to an **equivalence class**. Two elements belong to the same equivalence class iff they are equivalent. We can choose one of the elements in the class as a **class representative** to refer to the whole equivalence class.

3 Preprocessing framework

3.1 Simplification rules

Charles S. Peirce (1839-1914) provided the following sound and complete set of inference rules for EGs (for more details, see e.g. [24,31,29,9]), where an *EG-element* is any arbitrary clause[1] expressed as an EG:

[1] We refer here to the generic notion of 'clause', understood as a propositional formula consisting of a finite collection of literals and logical connectives.

1. (i) **Insertion:** in an odd area (nesting level $2k+1, k \in \mathbb{N}$), we can draw any EG-element.
   (e) **Erasure:** any EG-element in an even area (nesting level $2k, k \in \mathbb{N}$) can be deleted.
2. (i) **Iteration:** any EG-element in an area a can be duplicated in a or in any nested areas within a.
   (e) **Deiteration:** any EG-element whose occurrence could be the result of iteration may be erased.
3. (i) **Double Cut Insertion:** a double negation may be drawn around any collection of zero or more EG-elements in any area.
   (e) **Double Cut Erasure:** any double negations can be erased.

We investigate the reversibility properties of Peirce's EGs inference rules and their combinations in order to determine which sequences of rule applications and underlying restrictions ensure non-increasing model equivalence. Note that rules $2i$ & $2e$ and $3i$ & $3e$ are mutually reversible and, thus, preserve equivalence, whilst the first two rules are not.

To the best of our knowledge, the EG calculus has only been studied as a proof system and used to prove logical assertions by inference [9,31], but it has never been used as a simplification method nor restricted to equivalence-preserving rules.

In what follows, we present the definitions of each of our rules alongside a visual exemplification and compare them to existing preprocessing techniques. Throughout, let φ be an arbitrary propositional formula and $\mathrm{BIG}(\varphi)$ denote its binary implication graph.

Singleton wipe This rule is equivalent to a combination of Shin's *rules 1* and *2* [29], and to Peirce's EGs *deiteration rule* (2e above) restricted to single EG-elements (unit clauses) and extended to account for the generation of empty cuts. Most importantly, it can be seen as a generalisation of *unit propagation* applicable over arbitrary formulae, not only in CNF.

Definition 1 (Singleton wipe rule (SWR)). *Let x be a singleton (i.e. a literal or clause of size 1) in any clause or subclause of φ. Any other instances of x in the same clause or its subclauses can be deleted, and any generated empty subclause (i.e. $\neg()$) results in the deletion of its parent clause.*
From an EG point of view, any copy of a single EG-element in the same or any nested (inner) areas can be erased, and every area containing an empty cut (i.e. a negation oval) shall be deleted.

Note that, in linear notation, clausal levels can be distinguished by parenthesis (either explicit or implicit in accordance with the precedence of logical operators). Moreover, note that empty cuts are generated when the negation of the propagated singleton is encountered.

Example 1. Let $\varphi = P \wedge ((A \wedge D \wedge P \wedge (A \rightarrow B) \wedge (\neg C \vee D)) \vee (P \wedge Q \wedge R) \vee T \vee (S \wedge T) \vee \neg(X \rightarrow (X \wedge Y \wedge Z))) \wedge \neg T$. Several singletons can be

propagated at the same or inner levels, namely $A, D, P, \neg T$, and X. This is not easy for a human reader to detect if the formula is expressed in linear form, but it becomes apparent when expressed as an EG (see Fig. 3). If we apply SWR to φ, we obtain the following simplified equivalent form: $\varphi' = SWR(\varphi) = P \wedge \neg T \wedge ((A \wedge B \wedge D) \vee (Q \wedge R) \vee (X \wedge (\neg Y \vee \neg Z)))$. Note that transforming φ to CNF would result in a formula of 14 clauses of sizes 1 to 7 with a total of 66 literals, where traditional unit propagation could only be applied to the unit clauses P and $\neg T$, and the resulting (partially) simplified formula would have 14 clauses of sizes 1 to 6 with a total of 54 literals. Instead, if we transform φ' to CNF, we obtain a formula with 14 clauses of sizes 1 to 4 and a total of 44 literals.

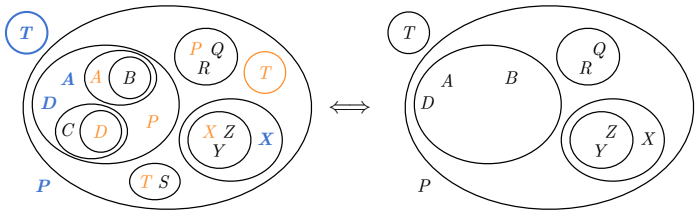


Fig. 3: EG of φ (left), where the EG-elements coloured in orange can be simplified by propagating their outermost instances, marked in blue, to obtain φ' (right).

Remark 1. SWR effectively tells us that each EG-element (literal, clause, or subclause) contributes to the truth value of the whole EG (i.e. the whole formula) at its outermost occurrence, and the presence or absence of more deeply nested instances is irrelevant.

Equivalence projection This rule can be achieved by nontrivial applications of the iteration, deiteration and cut rules ($2i$, $2e$, $3i$ and $3e$ from Section 3.1) and noticing that two propositional variables x and y are equivalent (resp. opposites) whenever we have the following two clauses: $(x \vee \overline{y}) \wedge (\overline{x} \vee y)$ (resp. $(x \vee y) \wedge (\overline{x} \vee \overline{y})$). This is obvious from the implication interpretation of these clauses as EGs that we saw in Section 2.

Trying all possible (de)iterations of binary clauses within each other is clearly impractical, and requires a nontrivial search for candidate EG-elements and backtracking. However, if we use the BIG of the formula to inform our procedure, it becomes straightforward and efficient.

Definition 2 (Equivalence projection rule (EPR)). *Every* strongly connected component *of* BIG(φ) *corresponds to an* equivalence class*, and all same-level or inner-level literals in the equivalence class can be substituted by their* class representative*. If a same-level or inner-level subclause contains multiple elements of the equivalence class, they can all be substituted by a single instance of the representative literal. If a subclause contains both an element of the equivalence class and the negation of an element in the equivalence class, it can be*

deleted. The BIG(EPR(φ)) *of the remaining binary clauses is guaranteed to be acyclic and corresponds to the* condensation *of* BIG(φ)*.*

The deletion of literals and clauses in EPR can be seen as a nested application of the SWR rule (see Definition 1) performed immediately after a substitution step. The replacement part of EPR is in fact equivalent to a well-known preprocessing technique called *equivalent-literal substitution* [14,19], but ours does not require a formula to be in CNF form. Moreover, our rule is equivalence-preserving since we keep the information of any equivalence classes, and both substitution and simplification are applied in one step. Additionally, the BIG built and updated in this preprocessing phase will inform other preprocessing techniques, so the effort of building the graph will be reused and further exploited.

Nested equivalence projection Equivalence projection can be applied not only at the formula level but also within nested cuts, which enhances its reduction powers and makes EPR applicable to formulae in forms other than CNF too (see Example 2). To do so, we maintain local implication graphs corresponding to the binary clauses present at each nesting level.

Example 2. Let $\varphi = (\overline{X} \vee \overline{Y} \vee \overline{A}) \wedge \neg((A \rightarrow B) \wedge (B \rightarrow A) \wedge (\overline{X} \vee \overline{Y} \vee \overline{B}))$. We generate the BIG of any area with binary clauses, which in this case is only the big level-1 area. The nested BIG contains a strongly connected component $[A] = \{A, B\}$, so the innermost B can be substituted by the representative A. The resulting ternary clause can then be wiped by the deiteration rule (and our rules to come), leading to the negation of the equivalence found (see Fig. 4). Hence φ can be simplified to $\varphi' = \text{EPR}(\varphi) = (\overline{X} \vee \overline{Y} \vee \overline{A})$ with $[A] = \{A, \overline{B}\}$.

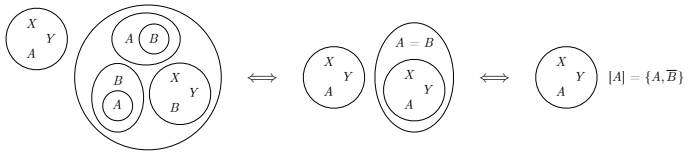


Fig. 4: EGs of $\varphi = (\overline{X} \vee \overline{Y} \vee \overline{A}) \wedge \neg((A \rightarrow B) \wedge (B \rightarrow A) \wedge (\overline{X} \vee \overline{Y} \vee \overline{B}))$ and its reduction to $\varphi' = (\overline{X} \vee \overline{Y} \vee \overline{A})$ with $[A] = \{A, \overline{B}\}$ after nestedly applying EPR and then deiteration.

Even higher reductions can be achieved if we consider equivalence —and, as we will see, implication chains— in the union of nested implication graphs (see Example 3).

Example 3. Let $\varphi = (\overline{X} \vee \overline{Y} \vee \overline{A}) \wedge (A \rightarrow B) \wedge (B \rightarrow A) \wedge \neg((C \rightarrow B) \wedge (B \rightarrow C) \wedge (\overline{X} \vee \overline{Y} \vee \overline{C}))$. We generate the BIG of any area with binary clauses, which in this case are the outermost area and the biggest level-1 area. The outer BIG contains a strongly connected component $[A] = \{A, B\}$, and the nested BIG contains a strongly connected component $[B] = \{B, C\}$. The union of both BIG results in the strongly connected component $[A] = \{A, B, C\}$, so the inner ternary clause

can be wiped by a combination of the EPR and the deiteration rule. Thus, φ can be simplified to $\varphi' = \mathrm{EPR}(\varphi) = (\overline{X} \vee \overline{Y} \vee \overline{A})$ with $[A] = \{A, B, \overline{C}\}$ (see Fig. 5).

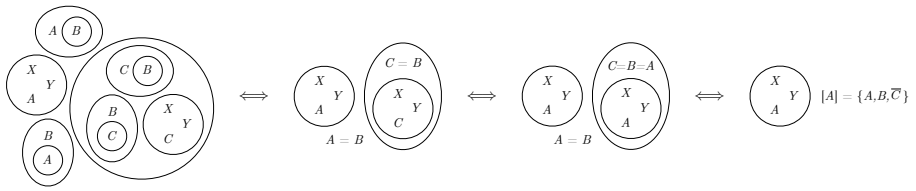


Fig. 5: EGs of $\varphi = (\overline{X} \vee \overline{Y} \vee \overline{A}) \wedge (A \to B) \wedge (B \to A) \wedge \neg((C \to B) \wedge (B \to C) \wedge (\overline{X} \vee \overline{Y} \vee \overline{C}))$ and its reduction to $\varphi' = (\overline{X} \vee \overline{Y} \vee \overline{A})$ with $[A] = \{A, B, \overline{C}\}$ after nestedly applying EPR to the union of nested BIGs and then deiteration.

Transitive reduction After applying EPR until completion, the BIG is guaranteed to be acyclic since all equivalences (and so strongly connected components) have been condensed into a representative literal (or node). However, the formula can still contain redundant binary clauses. In order to remove those, we compute the *transitive reduction* of the BIG (TRR), which in this case coincides with the minimum equivalent graph. As in the EPR, the same results can be achieved by nontrivial applications of the iteration, deiteration and cut rules, but these are only efficient if guided by the BIG. Moreover, the EGs viewpoint allows us to apply this rule in a nested form and so we can detect transitive redundancies in arbitrary formulas (see Example 4).

Example 4. Let $\varphi = (\overline{A} \vee B) \wedge (\overline{B} \vee C) \wedge \neg((C \vee \overline{A}) \wedge X \wedge Y)$. Then the level-0 BIG contains the implication chain $A \implies B \implies C$, and the level-1 binary clause is equivalent to $A \to C$. Computing the TR of the union of both BIGs shows that the inner binary clause is redundant and can be deleted. As illustrated in Fig. 6, we obtain the equivalent simplified formula $\varphi' = (\overline{A} \vee B) \wedge (\overline{B} \vee C) \wedge (\overline{X} \vee \overline{Y})$.

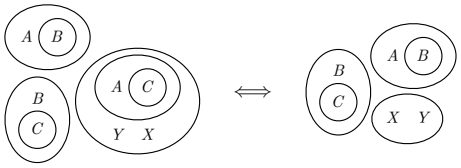


Fig. 6: EGs of $\varphi = (\overline{A} \vee B) \wedge (\overline{B} \vee C) \wedge \neg((C \vee \overline{A}) \wedge X \wedge Y)$ (left), and its equivalent reduction $\varphi' = \mathrm{TRR}(\varphi) = (\overline{A} \vee B) \wedge (\overline{B} \vee C) \wedge (\overline{X} \vee \overline{Y})$ (right).

Opposite singletons implication

Definition 3 (Opposite singletons implication rule (OSIR)). *If a directed path in BIG(φ) contains a literal l and later its negation $\bar{l}$, then all the literals including and after the consequent ($\bar{l}$) evaluate to true, and all the literals in the*

implication path before and including the antecedent (l) evaluate to false. The Boolean variables "collapsing" in opposite chains will be equal by construction, so only one side of the implication path needs to be evaluated. All evaluated literals can be added as singletons and propagated accordingly by the SWR rule.

Note that this preprocessing step can also be seen as a backtrack-free (i.e. non-look-ahead) exhaustive version of *failed-literal probing* (FLP) [4] over all (implicit and explicit) binary clauses, where we do not need to test candidate literals that might not lead to any new knowledge after a whole round of unit propagation. A strategy to make all possible unit-clause inferences from the BIG was proposed in [13], but their approach may add redundant, unnecessary clauses to the formula. As the previous rules, OSIR can also be applied to arbitrary nesting levels, so it does not require a formula to be in CNF (see Example 5).

Example 5. Let $\varphi = (\overline{X} \vee \overline{Z}) \wedge ((B \wedge \overline{C}) \vee (X \wedge Y \wedge \overline{A}) \vee (A \wedge \overline{B}) \vee (P \wedge A \wedge Q) \vee (C \wedge A))$. The BIG of the 1-nested biggest subformula contains the following implication chain: $A \implies B \implies C \implies \overline{A}$, so we can apply the OSIR to it and derive $\overline{A}$, which can be added as a nested singleton and later propagated using the SWR rule (see Fig. 7 below) to obtain the much simpler formula $\varphi' = \text{OSIR}(\varphi) = (\overline{X} \vee \overline{Z}) \wedge ((B \wedge \overline{C}) \vee (X \wedge Y) \vee A)$.

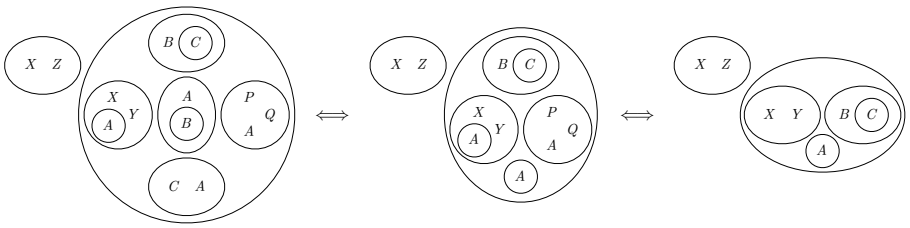


Fig. 7: EGs showing the application of OSIR to $\varphi = (\overline{X} \vee \overline{Z}) \wedge ((B \wedge \overline{C}) \vee (X \wedge Y \wedge \overline{A}) \vee (A \wedge \overline{B}) \vee (P \wedge A \wedge Q) \vee (C \wedge A))$ and the subsequent SWR application where the new singleton $\overline{A}$ is propagated to obtain the equivalent reduced formula $\varphi' = (\overline{X} \vee \overline{Z}) \wedge ((B \wedge \overline{C}) \vee (X \wedge Y) \vee A)$.

Tuple wipe and subflip After understanding the basics of Peirce's EGs rules and binary implication graphs, we can now introduce a rule which, in fact, generalises all the previous rules (except the non-reductive part of EPR).

Definition 4 (Tuple wipe and subflip rule (TWSR)). *Let $C_n, D_m \in \varphi$ be two (sub)clauses of size $n \leq m$. Let c be the nesting level of C_n, and D_m be either in the same area as C_n or in a nested area within c. Let* $\text{des}(l)$ *be the set of descendants of a literal l in* $\text{BIG}(\varphi)$. *Let* $\text{lit}(C_n) = \{p_1, \ldots, p_n\}$ *and* $\text{lit}(D_m) = \{q_1, \ldots, q_m\}$. *If for each $i \in \{1, ..., n-1\}$ either $p_i = q_i$ or $q_i \in \text{des}(p_i)$, and: (1) $p_n = q_n$ or $q_n \in \text{des}(p_n)$, then D_m can be deleted from φ; or (2) $p_n = \overline{q_n}$ or $\overline{q_n} \in \text{des}(p_n)$, then q_n can be deleted from D_m.*

Note that we need to specify the cases $p_n = q_n$ and $p_n = \overline{q_n}$ since it is always the case that $x \to x$ and $\overline{x} \to \overline{x}$, but these tautologies are not added to the BIG in order to keep it redundancy- and tautology-free.

As before, our rule can be applied to arbitrary nesting levels, and so it does not require a formula to be in CNF (see Example 6).

Example 6. Let $\varphi = (A \to E) \wedge (B \to F) \wedge (C \to G) \wedge \neg((\overline{A} \vee \overline{B} \vee G \vee \overline{H}) \wedge (\overline{A} \vee \overline{B} \vee \overline{C} \vee \overline{D}) \wedge (\overline{E} \vee \overline{F} \vee \overline{G}))$. Note that the BIG of the outermost area applies to the biggest 1-nested subformula. As illustrated in Fig. 8, we can apply TWSR to the clauses of sizes 3 and 4 inside the biggest 1-nested area and obtain the simplified equivalent formula $\varphi' = (A \to E) \wedge (B \to F) \wedge (C \to G) \wedge ((A \wedge B \wedge H) \vee (E \wedge F \wedge G))$. In particular, the clauses $P_3 = (\overline{E} \vee \overline{F} \vee \overline{G})$ and $Q_4 = (\overline{A} \vee \overline{B} \vee \overline{C} \vee \overline{D})$ satisfy condition (1) in TWSR's definition, and P_3 together with $R_4 = (\overline{A} \vee \overline{B} \vee G \vee \overline{H})$ satisfy condition (2).

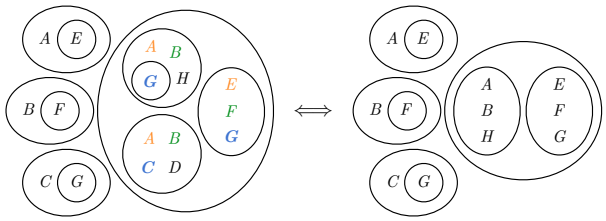
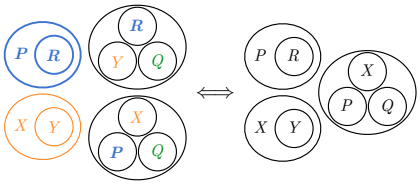


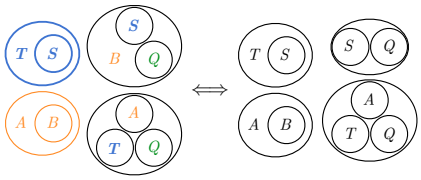
Fig. 8: EG of $\varphi = (A \to E) \wedge (B \to F) \wedge (C \to G) \wedge \neg((\overline{A} \vee \overline{B} \vee G \vee \overline{H}) \wedge (\overline{A} \vee \overline{B} \vee \overline{C} \vee \overline{D}) \wedge (\overline{E} \vee \overline{F} \vee \overline{G}))$ (left), where the literals satisfying TWSR's definition conditions are highlighted in matching colours. The equivalent reduced formula resulting from applying TWSR is shown on the right-hand side.

It is easy to see that TWSR with $n = 1$ is equivalent to SWR. TWSR restricted to binary clauses only (i.e. with $n = m = 2$) is equivalent to the clause deletion part of EPR and computing the TR of BIG(φ) when condition (1) is satisfied; and to the literal deletion part of EPR together with OSIR when condition (2) applies. However, TWSR cannot fully emulate EPR since it cannot perform the non-reductive substitution step (i.e. substitute a literal by its class representative) nor keep equivalence classes information. Thus, we recommend applying EPR before applying any form of TWSR (including SWR) in order to maximise the preservation of structural problem information. Note that preserving the structural information of the problem is of key importance not only in preprocessing but also for symmetry detection and general solving [1]. Fig. 9 shows two examples of TWSR applications for $n = m = 3$ satisfying, respectively, conditions (1) and (2).

Our TWSR is a generalisation of subsumption and self-subsuming resolution since it removes in one pass not only all the explicitly (self)subsumed redundant clauses but also all the *implicitly* (self)subsumed ones. It also generalises unit propagation, transitive reduction and, since it can be applied to clauses of arbitrary sizes, it is strictly more powerful than binary resolution combined with



(i) Let $\varphi = (P \to R) \wedge (X \to Y) \wedge (X \vee Q \vee P) \wedge (R \vee Y \vee Q)$. Let $C_3 = (X \vee Q \vee P)$ be the penultimate clause and $D_3 = (R \vee Y \vee Q)$ be the last clause in φ. These satisfy condition (1) in TWSR's definition, so D_3 can be deleted.



(ii) Let $\varphi = (T \to S) \wedge (A \to B) \wedge (S \vee \overline{B} \vee Q) \wedge (A \vee Q \vee T)$. Let $C_3 = (A \vee Q \vee T)$ and $D_3 = (S \vee \overline{B} \vee Q)$. They satisfy condition (2) in TWSR's definition, so the literal $\overline{B}$ can be deleted from D_3.

Fig. 9: EGs of two applications of the TWSR, where colours highlight related literals (equal –in green– or in the same implication chain –in orange or blue).

transitive reduction and unit propagation, while guaranteeing a non-increasing number of variables, literals and clauses (i.e. redundant variables, literals or clauses are never added). TWSR can also be seen as a backtrack-free (non-look-ahead, i.e., with no arbitrary assignments) more efficient version of FLP over all literals in the BIG of the formula.

Moreover, TWSR strictly generalises the combination of *hidden subsumption elimination* (HSE) [15], *hidden tautology elimination* (HTE) [15] and *hidden literal elimination* (HLE) [16] in a manner that is guaranteed to never increase the size of the formula. Note that this is not necessarily the case for the aforementioned rules. For example, in order to deduce that clauses are redundant, HTE adds literals instead of removing them and tests if this leads to a tautology. Note in particular that neither of these rules nor their combination could achieve the reduction shown in Example 6, even if all clauses were at the same nesting level. For formulas in CNF, HTE combined with HSE, computing the transitive closure of BIG(φ) and adding all new binary clauses to φ can achieve the same reduction as TWSR with condition (1) only, but clearly at a much higher cost and space complexity.

3.2 Rules properties

Visual proofs using the EG calculus are much easier to follow than symbolic ones, and EGs inference rules have been used to establish the results for the two theorems presented below. However, because of space constraints, we only provide symbolic and 'verbal'-EGs skeleton proofs here. Moreover, since the TWSR rule generalises all the others except the substitution part of EPR, we need only provide proofs for these two rules.

Theorem 1. *SWR, EPR, TRR, OSIR, and TWSR are all reversible and so equivalence-preserving.*

Proof (TWSR). Let $C_n, D_m \in \varphi$ be two (sub)clauses of size $n \leq m$ as in TWSR's definition, with $\mathrm{lit}(C_n) = \{p_1, ..., p_n\}$ and $\mathrm{lit}(D_m) = \{q_1, ..., q_m\}$. (1) Let $p_i = q_i$ or $q_i \in \mathrm{des}(p_i)$ for each $i \in \{1, ..., n\}$. We can iterate C_n inside D_m, and all literals in the inner copy satisfying $p_i = q_i$ can be deleted by deiterating their copies in D_m. All literals in the inner copy of C_n satisfying $q_i \in \mathrm{des}(p_i)$ can also be deleted as follows: since $q_i \in \mathrm{des}(p_i)$, it means that we have the binary clauses $(\overline{p_i} \vee x_1) \wedge \cdots \wedge (\overline{x_j} \vee q_i)$, which we can iterate inside D_m, and successively deiterate all the consequents until we obtain $D_m \vee p_i \vee x_1 \vee \cdots \vee x_j \vee \overline{C_n}$. Thus, D_m can be expanded with all p_i's, which can then be deleted from the inner copy of C_n, leaving an empty cut which is either in an even or in an odd area. If the empty cut is in an even area, then we have $\bot$ in a conjunction, which evaluates to `False`*. This conjunction is contained in an odd area, and so it can be deleted from the disjunction. If the empty cut is in an odd area, then we have $\top$ in a disjunction, and the whole clause evaluates to* `True`*, which can be safely deleted from its implicit surrounding conjunction. (2) Let $p_i = q_i$ or $q_i \in \mathrm{des}(p_i)$ for each $i \in \{1, ..., n-1\}$ and $p_n = \overline{q_n}$ or $\overline{q_n} \in \mathrm{des}(p_n)$. We can iterate C_n inside the nesting area of q_n in D_m (with the insertion of a double cut around it if required). If $p_n = \overline{q_n}$, then $\overline{q_n}$ inside the cut can be deiterated from the inner copy of C_n, and all the remaining $n-1$ literals can be deleted as in (1). If $\overline{q_n} \in \mathrm{des}(p_n)$, we can deiterate the $n-1$ literals from the inner copy as in (1) and are left with $(\overline{p_n} \vee q_n)$. Since we know that $\overline{q_n} \in \mathrm{des}(p_n)$, we can derive $(\overline{p_n} \vee \overline{q_n})$ from the implication chain as in (1), iterate it inside q_n's cut and deiterate its inner q_n and its inner p_n to obtain an empty cut, which results in deleting q_n from D_m, but we then have a $\overline{p}$ in its place. This extra $\overline{p}$ can be deleted by inserting a double cut around it, iterating the $n-1$ implications $q_i \in \mathrm{des}(p_i)$ inside and deiteraiting their $q_1, \ldots, q_{n-1}$. This results in $D_m \vee \overline{C_n}$ from which we can deiterate C_n to obtain $D_m \setminus \{q_n\}$. Since we have only used Peirce's iteration (2i), deiteration (2e) and double cut (3i and 3e) rules, which are all reversible and equivalence-preserving, we know that our rule is too.*

Proof (EPR). Trivial, since we retain the information on equivalence classes. But, in more detail, the equivalence preservation of the reductive part of EPR follows from TWSR's proof above, and the equivalence of the substitution part can be proved as follows: Let x and y be in the same strongly connected component of a BIG. Then, we have or can easily deduce the following two clauses: $\overline{x} \vee y$ and $\overline{y} \vee x$. For any clause C containing y we can iterate $\overline{x} \vee y$ within C, and deiterate y from it to obtain $C \vee x$. We can then iterate $\overline{y} \vee x$ within the nested area of y (potentially adding a double cut), and deiterate y from it to obtain $y \wedge x$, which can be deleted by an iteration of the copy of x in expanded C, to obtain C with y replaced by x.

Given that all of our rules can never add any variables, literals or clauses, it is also straightforward to prove the following theorem.

Theorem 2. *The applications of SWR, EPR, TRR, OSIR, and TWSR are guaranteed to be monotonically non-increasing in the number of variables, in the number of clauses and in the number of literals of a propositional formula.*

Proof (EPR). Trivial, since either (i) the BIG has no strongly connected components and so the formula stays the same, or (ii) at least a strongly connected component is found, and for each component, at least the binary clauses corresponding to all the edges in the component can be deleted.

Proof (TWSR). Trivial, since either (i) TWSR does not apply so the formula stays the same, (ii) condition (1) is satisfied and so the number of clauses is reduced by one, or (iii) condition (2) applies and the number of literals is reduced by one.

4 Discussion and future work

Our systematic reduction procedure applies EPR whenever new binary clauses are present, and then prioritises the propagation (by TWSR) of the smallest unprocessed clauses, since these have greater reduction potential than bigger clauses. Our approach might appear to be quadratic in nature, but this is easily avoidable by using existing efficient graph algorithms. For example, finding strongly connected components for the EPR step can be linear [28,32,10] in the number of edges and nodes of the BIG (which is less than $2e + 2n$ with e being the number of binary clauses and n the number of variables present in binary clauses). Additional applications of EPR only need to search for strongly connected components containing any new BIG edges, which is even faster. The substitution step can be done efficiently using occurrence lists [5], or if we condense the hypergraph of the formula instead of searching for equivalent literals to be replaced in every clause. The reachability queries performed in TWSR can be answered in as low as $\mathcal{O}(1)$ time with the right data structure and a preprocessing step of $\mathcal{O}((2n)^3)$ time in the worst case [8], where n is again the number of variables present in binary clauses. In order to avoid a quadratic number of comparisons in TWSR, we can sort clauses and literals, or use known tactics such as the one-watched [35] or two-watched [34,23] literal schemes. Traditional literal occurrence lists would not be as helpful in this case since they do not capture the 'hidden' occurrences. However, other kinds of lists would be helpful, such as lists of clauses containing a given number of BIG nodes (since a clause C_n can only potentially reduce another clause D_m if D_m has at least as many nodes in the BIG as C_n). TWSR could also be guided by the hypergraph of the formula, which can give a first approximation to the (undirected) connectedness of literals and significantly reduce the search for redundant clauses. Finally, processing independent components of the BIG and the hypergraph of the formula would obviously speed up our simplification approach, but parallelisation of preprocessing is still in its infancy [7]. Nonetheless, using any of these known techniques can certainly improve the performance of our method as well as increase its scalability.

Another key advantage of our unified approach is that it eliminates the need to choose between different techniques or their order of application, which can have a major impact on the final level of reduction (e.g. some preprocessing techniques add redundant clauses by resolution, whilst others delete them, which

could lead to an infinite loop). It can also be the case that some rules are not applicable at first, but they can provide further reduction if applied after other techniques. This is a nontrivial problem usually resolved by heuristics (e.g. restarts) or by limiting the application and behaviour of the rules according to arbitrary parameters (e.g. apply preprocessing to learned or bigger clauses only). Example 7 illustrates these problems and how our solution addresses them instead. Moreover, note that our approach is guaranteed to terminate since it never increases the size of the problem.

Example 7. Let $\varphi = (A \vee B \vee C) \wedge (A \vee B \vee D) \wedge (B \vee \overline{C}) \wedge (\overline{A} \vee B \vee \overline{C})$. If we apply TWSR to it, our approach first propagates the smallest clause $(B \vee \overline{C})$ and reduces the formula to $\text{TWSR}(\varphi) = \varphi' = (A \vee B) \wedge (A \vee B \vee D) \wedge (B \vee \overline{C})$. TWSR next propagates the smallest unprocessed clause $(A \vee B)$, and returns the equivalent formula $\varphi'' = (A \vee B) \wedge (B \vee \overline{C})$. This same result could also be obtained by using other existing techniques combined in many different ways. For example, applying a round of subsumption, then a round of self-subsumption, and then another round of subsumption would achieve the same reduction in this case. We can also obtain φ'' by applying HTE, HSE and self-subsumption in any order. Adding all the possible redundant clauses obtained by resolution and then applying two rounds of subsumption would also lead to φ''. Note, however, that the choice of techniques or their application orders would of course be problem-dependent and not obvious beforehand, so many techniques would probably be unsuccessfully applied before reaching (or not) the desired result.

Some of the existing computationally most expensive preprocessing techniques (namely FLP, *hyper binary resolution* (HBR) [3], *asymmetric subsumption elimination* [15], *asymmetric literal elimination* [7] and *asymmetric tautology elimination* (ATE) [15]) can achieve reductions on CNF formula that TWSR is not able to attain. However, TWSR can reach the same level of reduction if applied to the original nested formula (see Example 8).

Example 8. Let $\varphi = (\overline{C} \vee \neg(\overline{A} \vee \overline{B} \vee \overline{D})) \wedge (\overline{A} \vee \overline{B} \vee \overline{D})$. Let $CNF(\varphi) = (\overline{C} \vee A) \wedge (\overline{C} \vee B) \wedge (\overline{C} \vee D) \wedge (\overline{A} \vee \overline{B} \vee \overline{D})$ be φ expressed in conjunctive normal form. Both FLP and HBR can reduce $CNF(\varphi)$ to $\varphi' = \overline{C} \wedge (\overline{A} \vee \overline{B} \vee \overline{D})$, but TWSR would not be able to do so unless we apply first a nontrivial EG factorisation step to recover its nested form. However, if we apply TWSR directly to φ, we can obtain φ' much more efficiently (see Fig. 10). Note that none of the existing preprocessing methods can be applied directly to φ since it is not in CNF.

In fact, by using EGs and BIGs, we have realised that TWSR and these advanced SAT preprocessing techniques can actually be seen as n-ary versions of TRR and OSIR , where the nodes of the implication graph can be clauses instead of singletons. That is, instead of finding a redundant edge between two singletons, we remove redundant edges between implied clauses (see Example 9), and instead of finding a singleton implying its negation, we uncover an n-ary clause implying its negation (see Example 10). Thus, we are currently working on an extension of TWSR guided by the *n-ary implication hypergraph* of the

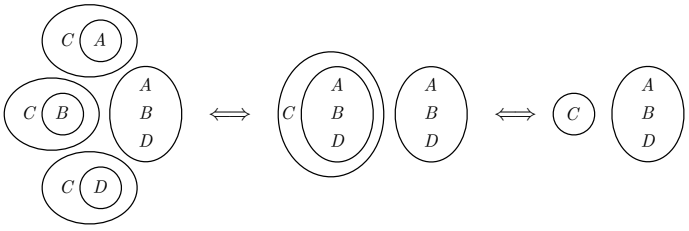


Fig. 10: EGs of CNF(φ) = $(\overline{C} \vee A) \wedge (\overline{C} \vee B) \wedge (\overline{C} \vee D) \wedge (\overline{A} \vee \overline{B} \vee \overline{D})$ (left), its equivalent factorised form $\varphi = (\overline{C} \vee \neg(\overline{A} \vee \overline{B} \vee \overline{D})) \wedge (\overline{A} \vee \overline{B} \vee \overline{D})$ (middle), and its equivalent reduced form TWSR(φ) = $\varphi' = \overline{C} \wedge (\overline{A} \vee \overline{B} \vee \overline{D})$ (right).

formula, which we claim will be able to generalise all the aforementioned rules while still guaranteeing a never-increasing problem size.

Example 9. Let $\varphi = (\overline{A} \vee \overline{B} \vee C) \wedge (A \vee \overline{B} \vee \overline{C}) \wedge (\overline{A} \vee B \vee D) \wedge (A \vee B \vee \overline{D}) \wedge (A \vee \overline{C} \vee \overline{D}) \wedge (\overline{A} \vee\ C \vee D)$. These clauses define the following three implication chains (amongst many others): $B \wedge C \implies A \implies \overline{B} \vee C$, $\overline{B} \wedge D \implies A \implies B \vee D$, and $C \wedge D \implies A \implies C \vee D$. These can be combined to form the following equivalent implication chain: $(B \wedge C) \vee (\overline{B} \wedge D) \vee (C \wedge D) \implies A \implies (\overline{B} \vee C) \wedge (B \vee D) \wedge (C \vee D)$, which easily reduces to: $(B \wedge C) \vee (\overline{B} \wedge D) \implies A \implies (\overline{B} \vee C) \wedge (B \vee D)$, meaning that $(A \vee \overline{C} \vee \overline{D})$ and $(\overline{A} \vee\ C \vee D)$ are redundant clauses in φ. Our current version of TWSR cannot find these redundancies, nor can FLP or HBR. From the rules we have mentioned in this paper, only ATE would be able to detect this redundancy.

Example 10. Let $\varphi = (\overline{X} \vee Y) \wedge (\overline{Y} \vee \overline{Z}) \wedge (X \vee \overline{P} \vee \overline{Q}) \wedge (\overline{Y} \vee \overline{P} \vee \overline{Q})$. Remember that we can read $(\overline{X} \vee Y)$ as $(X \rightarrow Y)$ or, equivalently, $(\overline{Y} \rightarrow \overline{X})$. The clause $(X \vee \overline{P} \vee \overline{Q})$ can be interpreted as $(P \wedge Q \rightarrow X)$ or $(\overline{X} \rightarrow \overline{(P \wedge Q)})$, amongst many other equivalent readings. Notice that the reduction obtained from the application of TWSR, $\varphi' = (\overline{X} \vee Y) \wedge (\overline{Y} \vee \overline{Z}) \wedge (\overline{P} \vee \overline{Q})$, can be seen as applying an n-ary version of OSIR to the following implication path: $P \wedge Q \implies \overline{Y} \implies \overline{X} \implies \overline{(P \wedge Q)}$.

5 Conclusion

Reasoning with EGs allowed us to independently rediscover many existing equivalence preserving SAT preprocessing techniques, gain a better understanding of their underlying relationships, and more easily prove and establish many of their properties. EGs also offer a fresh viewpoint which led to a novel approach that generalises many of these techniques with added advantages: it is more efficient, avoids look-ahead backtracking, guarantees a monotonically decreasing number of variables, literals and clauses (and so termination), it is structure-preserving, can be applied to nested formulae, and does not require CNF.

With our approach, it becomes clear why some simplification techniques based on adding redundant clauses or literals sometimes help reduce the problem but other times do not, leading to wasted preprocessing effort and the need for bespoke or contrived heuristics. Since our proposed method generalises a significant set of previously-thought independent techniques, the high complexity and effort associated with finding a suitable application order are drastically reduced. Our rules can also decrease the space complexity of the problem since they may be applied at a higher level before formulas are flattened (e.g. converted to CNF). This can greatly minimise the search space and even prevent potential explosions of the formula size during translation. Our reductions also allow for further problem insight and understanding, which can lead to better modelling, better solving strategies, search heuristics and translations, further symmetry breaking, provide model counting bounds and aid #SAT. Moreover, our techniques are solver-, problem-, and form-agnostic, and apply to propositional logic formula in general, so they can be of use in other fields such as SMT, automated reasoning and theorem proving.

Future work includes extensions of the TWSR rule informed by a novel n-ary implication hypergraph, refining our current implementation and a formal complexity analysis of our working algorithm.

References

1. Anders, M.: SAT preprocessors and symmetry. In: Meel, K.S., Strichman, O. (eds.) 25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel. LIPIcs, vol. 236, pp. 1:1–1:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022), https://doi.org/10.4230/LIPIcs.SAT.2022.1
2. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Inf. Process. Lett. **8**(3), 121–123 (1979), https://doi.org/10.1016/0020-0190(79)90002-4
3. Bacchus, F.: Enhancing davis putnam with extended binary clause reasoning. In: Dechter, R., Kearns, M.J., Sutton, R.S. (eds.) Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada. pp. 613–619. AAAI Press / The MIT Press (2002), http://www.aaai.org/Library/AAAI/2002/aaai02-092.php
4. Berre, D.L.: Exploiting the real power of unit propagation lookahead. Electron. Notes Discret. Math. **9**, 59–80 (2001), https://doi.org/10.1016/S1571-0653(04)00314-2
5. Biere, A.: Resolve and expand. In: SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings (2004), http://www.satisfiability.org/SAT04/programme/93.pdf
6. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)

7. Biere, A., Järvisalo, M., Kiesl, B.: Preprocessing in SAT solving. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 391–435. IOS Press (2021), https://doi.org/10.3233/FAIA200992
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Transitive closure of a directed graph. Introduction to Algorithms pp. 632–634 (2001)
9. Dau, F.: Mathematical logic with diagrams – based on the existential graphs of peirce. Habilitation thesis. TU Darmstadt, Germany (2008), http://www.dr-dau.net/Papers/habil.pdf
10. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976), https://www.worldcat.org/oclc/01958445
11. Eén, N., Mishchenko, A., Sörensson, N.: Applying logic synthesis for speeding up SAT. In: Marques-Silva, J., Sakallah, K.A. (eds.) Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4501, pp. 272–286. Springer (2007), https://doi.org/10.1007/978-3-540-72788-0_26
12. Fazekas, K., Biere, A., Scholl, C.: Incremental inprocessing in SAT solving. In: Janota, M., Lynce, I. (eds.) Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11628, pp. 136–154. Springer (2019), https://doi.org/10.1007/978-3-030-24258-9_9
13. Gelder, A.V.: Toward leaner binary-clause reasoning in a satisfiability solver. Ann. Math. Artif. Intell. **43**(1), 239–253 (2005), https://doi.org/10.1007/s10472-005-0433-5
14. Gelder, A.V., Tsuji, Y.K.: Satisfiability testing with more reasoning and less guessing. In: Johnson, D.S., Trick, M.A. (eds.) Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, pp. 559–586. DIMACS/AMS (1993), https://doi.org/10.1090/dimacs/026/27
15. Heule, M., Järvisalo, M., Biere, A.: Clause elimination procedures for CNF formulas. In: Fermüller, C.G., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6397, pp. 357–371. Springer (2010), https://doi.org/10.1007/978-3-642-16242-8_26
16. Heule, M., Järvisalo, M., Biere, A.: Efficient CNF simplification based on binary implication graphs. In: Sakallah, K.A., Simon, L. (eds.) Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6695, pp. 201–215. Springer (2011), https://doi.org/10.1007/978-3-642-21581-0_17
17. Järvisalo, M., Heule, M., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7364, pp. 355–370. Springer (2012), https://doi.org/10.1007/978-3-642-31365-3_28
18. Karnaugh, M.: The map method for synthesis of combinational logic circuits. Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics **72**(5), 593–599 (1953)

19. Li, C.M.: Integrating equivalency reasoning into davis-putnam procedure. In: Kautz, H.A., Porter, B.W. (eds.) Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA. pp. 291–296. AAAI Press / The MIT Press (2000), http://www.aaai.org/Library/AAAI/2000/aaai00-045.php
20. Li, C., Xiao, F., Luo, M., Manyà, F., Lü, Z., Li, Y.: Clause vivification by unit propagation in CDCL SAT solvers. Artif. Intell. **279** (2020), https://doi.org/10.1016/j.artint.2019.103197
21. Luo, M., Li, C., Xiao, F., Manyà, F., Lü, Z.: An effective learnt clause minimization approach for CDCL SAT solvers. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 703–711. ijcai.org (2017), https://doi.org/10.24963/ijcai.2017/98
22. McCluskey, E.J.: Minimization of boolean functions. The Bell System Technical Journal **35**(6), 1417–1444 (1956)
23. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001. pp. 530–535. ACM (2001), https://doi.org/10.1145/378239.379017
24. Peirce, C.: Existential graphs: manuscript 514, with commentary by jf sowa. Self-published by JF Sowa.[online] URL: http://www. jfsowa. com/peirce/ms514. htm (1909)
25. Roberts, D.D.: The existential graphs of Charles S. Peirce. Ph.D. thesis, University of Illinois at Urbana-Champaign (1963)
26. Roberts, D.D.: The existential graphs of Charles S. Peirce. Mouton (1973)
27. Rudell, R.L.: Logic synthesis for VLSI design. Ph.D. thesis, University of California, Berkeley (1989)
28. Sharir, M.: A strong-connectivity algorithm and its applications in data flow analysis. Computers & Mathematics with Applications **7**(1), 67–72 (1981)
29. Shin, S.: Reconstituting beta graphs into an efficacious system. J. Log. Lang. Inf. **8**(3), 273–295 (1999), https://doi.org/10.1023/A:1008303204427
30. Sörensson, N., Biere, A.: Minimizing learned clauses. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5584, pp. 237–243. Springer (2009), https://doi.org/10.1007/978-3-642-02777-2_23
31. Sowa, J.F.: Peirce's tutorial on existential graphs. Semiotica **2011**(186), 347–394 (2011)
32. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972), https://doi.org/10.1137/0201010
33. Zeman, J.J.: The graphical logic of C. S. Peirce. Ph.D. thesis, The University of Chicago (1964)
34. Zhang, H., Stickel, M.E.: Implementing the davis-putnam method. J. Autom. Reason. **24**(1/2), 277–296 (2000), https://doi.org/10.1023/A:1006351428454
35. Zhang, L.: On subsumption removal and on-the-fly CNF simplification. In: Bacchus, F., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3569, pp. 482–489. Springer (2005), https://doi.org/10.1007/11499107_42