SAT-Inspired Eliminations for Superposition

PETAR VUKMIROVIĆ, Vrije Universiteit Amsterdam
JASMIN BLANCHETTE, Vrije Universiteit Amsterdam and Université de Lorraine, CNRS, Inria, LORIA
MARIJN J. H. HEULE, Carnegie Mellon University

Optimized SAT solvers not only preprocess the clause set, they also transform it during solving as inprocessing. Some preprocessing techniques have been generalized to first-order logic with equality. In this article, we port inprocessing techniques to work with superposition, a leading first-order proof calculus, and we strengthen known preprocessing techniques. Specifically, we look into elimination of hidden literals, variables (predicates), and blocked clauses. Our evaluation using the Zipperposition prover confirms that the new techniques usefully supplement the existing superposition machinery.

CCS Concepts: • **Theory of computation → Automated reasoning**; **Logic and verification**; **Constraint and logic programming;**

Additional Key Words and Phrases: Theorem proving, first-order logic, superposition calculus, SAT solving

ACM Reference format:
Petar Vukmirović, Jasmin Blanchette, and Marijn J. H. Heule. 2023. SAT-Inspired Eliminations for Superposition. *ACM Trans. Comput. Logic* 24, 1, Article 7 (January 2023), 25 pages.
https://doi.org/10.1145/3565366

1 INTRODUCTION

Automated reasoning tools have become much more powerful in the last few decades thanks to procedures such as conflict-driven clause learning (CDCL) [35] for propositional logic and superposition [3] for first-order logic with equality. However, the effectiveness of these procedures crucially depends on how the input problem is represented as a clause set. The clause set can be optimized beforehand (*preprocessing*) or during the execution of the procedure (*inprocessing*). In

Vukmirović and Blanchette's research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka). Blanchette's research has also received funding from the Netherlands Organization for Scientific Research (NWO) under the Vidi program (project No. 016.Vidi.189.037, Lean Forward). Heule is supported by the National Science Foundation (NSF) under grant CCF-2015445.
Authors' addresses: P. Vukmirović, NU Building, Vrije Universiteit Amsterdam, Department of Computer Science Section of Theoretical Computer Science, De Boelelaan 1111, 1081 HV Amsterdam, The Netherlands; email: p.vukmirovic@vu.nl; J. Blanchette, NU Building, Vrije Universiteit Amsterdam, Department of Computer Science Section of Theoretical Computer Science, De Boelelaan 1111, 1081 HV Amsterdam, The Netherlands and Université de Lorraine, CNRS, Inria, LORIA, Nancy, Campus Scientifique, 615 rue du Jardin-Botanique, 54506 Vanduvre-lès-Nancy, France; email: j.c.blanchette@vu.nl; M. J. H. Heule, Gates-Hillman Complex 9107, School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue Pittsburgh, PA 15213-3891, USA; email: marijn@cmu.edu.

1529-3785/2023/01-ART7 $15.00
https://doi.org/10.1145/3565366

this article, we lift several preprocessing and inprocessing techniques from propositional logic to clausal first-order logic and demonstrate their usefulness in a superposition prover.

For many years, SAT solvers have used inexpensive clause simplification techniques such as hidden literal elimination (HLE) and hidden tautology elimination (HTE) [27, 28] and failed literal detection [23, Section 1.6]. We generalize these techniques to first-order logic with equality (Section 3). Since the generalization involves reasoning about infinite sets of literals, we propose restrictions to make them usable.

Variable elimination, based on Davis–Putnam resolution [16], has been studied in the context of both propositional logic [14, 45] and quantified Boolean formulas (QBFs) [8]. The basic idea is to resolve all clauses with negative occurrences of a propositional variable (i.e., a nullary predicate symbol) against clauses with positive occurrences and delete the parent clauses. Eén and Biere [19] refined the technique to identify a subset of clauses that effectively define a variable and use it to further optimize the clause set. This latter technique, *variable elimination by substitution*, has been an important preprocessor component in many SAT solvers since its introduction in 2004.

Specializing second-order quantifier elimination [24, 38], Khasidashvili and Korovin [30] adapted variable elimination to preprocess first-order problems, yielding a technique we call *singular predicate elimination* (SPE). We extend their work along two axes (Section 4): We generalize Eén and Biere's refinement to first-order logic, resulting in *defined predicate elimination*, and explain how both types of predicate elimination can be used during the proof search as inprocessing.

The last technique we study is *blocked clause elimination* (BCE) (Section 5). It is used in both SAT [29] and QBF solvers [9]. Its generalization to first-order logic has produced good results when used as a preprocessor, especially on satisfiable problems [32]. We explore more ways to use BCE on satisfiable problems, including using it to establish equisatisfiability with an empty clause set or as an inprocessing rule. Unfortunately, we find that its use as inprocessing can compromise the refutational completeness of superposition.

All techniques are implemented in the Zipperposition prover (Section 6), allowing us to ascertain their usefulness (Section 7). The best configuration solves 160 additional problems on benchmarks consisting of all 13,495 first-order TPTP theorems [46]. The raw experimental data are publicly available.[1]

This article expands on our FMCAD 2021 paper [49]. It includes more examples, lemmas, and proofs as well as a discussion section. It also corrects a mistake in the handling of variables in Definition 5.2.

2 PRELIMINARIES

Our work relies on a clausal version of first-order logic. Some of our techniques and results apply only to the superposition calculus. This section provides a brief introduction to these concepts.

2.1 Clausal First-Order Logic

Our setting is many-sorted, or many-typed, first-order logic [25] with interpreted equality and a distinguished type (or sort) o. Each variable x is assigned a non-Boolean type, and each symbol f is assigned a tuple $(\tau_1, \ldots, \tau_n, \tau)$ where $n \geq 0$, τ_i are non-Boolean types, and τ is the *result type*. We distinguish between *predicate symbols*, with o as the result type, and *function symbols*. Nullary function symbols are called *constants*. Terms are either variables x or applications $\mathrm{f}(t_1, \ldots, t_n)$, of type τ where f is assigned $(\tau_1, \ldots, \tau_n, \tau)$, and each t_i is a term of type τ_i. The parentheses are omitted if $n = 0$. A term is *ground* if it contains no variables. We assume standard definitions and

[1] https://doi.org/10.5281/zenodo.4552499.

notations for positions, subterms, and contexts [2]. We abbreviate a vector $(a_1, \ldots, a_n)$ to $\vec{a}_n$ or $\vec{a}$, and write $\mathsf{f}^i(s)$ for the i-fold application of a unary symbol f (e.g., $\mathsf{f}^3(x) = \mathsf{f}(\mathsf{f}(\mathsf{f}(x)))$).

An atom is an equation $s \approx t$ corresponding to an unordered pair $\{s, t\}$. A literal is an equation $s \approx t$ or a disequation $s \not\approx t$. For every predicate symbol p, the notation $\mathsf{p}(\vec{s})$ abbreviates $\mathsf{p}(\vec{s}) \approx \top$, and $\neg\,\mathsf{p}(\vec{s})$ abbreviates $\mathsf{p}(\vec{s}) \not\approx \top$, where $\top$ is a distinguished constant of type o. We distinguish between *predicate literals* $(\neg)\,\mathsf{p}(\vec{s})$ and *functional literals* $s \approx t$, where s and t are not of type o. Given a literal L, we overload notation and write $\neg L$ to denote its complement. A clause C is a finite multiset of literals, written as $L_1 \vee \cdots \vee L_n$ and interpreted disjunctively. Clauses are often defined as sets of literals, but superposition needs multisets; with multisets, an instance $C\sigma$ always has the same number of literals as C, a most convenient property. Given a clause set N, $N{\downarrow_2}$ denotes the subset of its binary clauses: $N{\downarrow_2} = \{L_1 \vee L_2 \mid L_1 \vee L_2 \in N\}$.

A substitution σ is a well-typed mapping from variables to terms such that the set $\{x \mid \sigma(x) \neq x\}$ is finite. Substitutions are written as $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ or $\{\vec{x} \mapsto \vec{t}\}$. A substitution is a *variable renaming* if it is a bijection from a set of variables to a set of variables.

We assume the natural extensions of domain, valuation, interpretation, and model (as defined by Fitting [22]) from unsorted to many-sorted logic. The models we consider are *normal*—i.e., they interpret $\approx$ as an equality relation. Usual notions of (un)satisfiability and (in)validity are assumed. We write $\mathcal{J} \models_\xi N$ to denote that a model $\mathcal{J}$ satisfies a clause set N, for a variable assignment ξ. If $\mathcal{J}$ is a model of N (i.e., it satisfies it under every variable assignment), we simply write $\mathcal{J} \models N$. Abusing notation, we write $M \models N$ to denote that M *entails* N, i.e., that every model of M is a model of N.

A canonical model $\mathcal{J}$ is a normal model such that for every element d in one of $\mathcal{J}$'s domains there exists a ground term t such that $\mathcal{J}$ interprets t as d. Canonical models generalize Herbrand models to first-order logic with equality: If a clause set is satisfiable, then it is satisfiable in a canonical model.

2.2 Superposition Provers

Superposition [3] is a calculus for clausal first-order logic that extends ordered resolution [4] with equality reasoning. It is refutationally complete: Given a finite, unsatisfiable clause set, it will eventually derive the empty clause. It is parameterized by a *selection function* that influences which of a clause's literals are eligible as the target of inferences. Moreover, it is compatible with the *standard redundancy criterion*, which can be used to delete a clause C while preserving completeness of the calculus.

The redundancy criterion relies on an order $\succ$ that compares terms, literals, clauses, or finite clause sets. The order is used to determine whether clauses can be deleted. If N is ground, C can be deleted if it is entailed by $\prec$-smaller clauses in N. This definition is lifted to nonground sets N. The criterion can be used to delete a clause that is *subsumed* by another clause (e.g., $\mathsf{p}(\mathsf{a}) \vee \mathsf{q}$ by $\mathsf{p}(x)$) or to *simplify* a clause C into C', which amounts to adding C' and then deleting C as redundant with respect to $N \cup \{C'\}$. Subsumption and simplification are the main inprocessing mechanisms available to superposition provers. Some provers also implement clause splitting [20, 41, 48].

Superposition provers saturate the input problem with respect to the calculus's inference rules using the *given clause procedure* [1, 36]. It partitions the proof state into a passive set $\mathcal{P}$ and an active set $\mathcal{A}$. All clauses start in $\mathcal{P}$. At each iteration of the procedure's main loop, the prover chooses a clause C from $\mathcal{P}$, simplifies it, and moves it to $\mathcal{A}$. Then all inferences between C and active clauses are performed. The resulting clauses are again simplified and put in $\mathcal{P}$. The provers differ in which clauses are used for simplification: Otter-loop [36] provers use both active and passive clauses whereas DISCOUNT-loop [1] provers use only active clauses.

3 HIDDEN-LITERAL-BASED ELIMINATION

In propositional logic, binary clauses from a clause set N can be used to efficiently discover literals L, L' for which the implication $L' \longrightarrow L$ is entailed by N's binary clauses—i.e., literals L, L′ such that $N{\downarrow_2} \models L' \longrightarrow L$. Heule et al. [28] introduced the concept of *hidden literals* to capture such implications.

Definition 3.1. Given a propositional literal L and a propositional clause set N, the set of *propositional hidden literals* for L and N is $\mathrm{HL}_{\mathrm{prop}}(L, N) = \{L' \mid L' \hookrightarrow^*_{\mathrm{prop}} L\}\backslash\{L\}$, where $\hookrightarrow_{\mathrm{prop}}$ is defined such that $\neg L_1 \hookrightarrow_{\mathrm{prop}} L_2$ whenever $L_1 \vee L_2 \in N$. Moreover, $\mathrm{HL}_{\mathrm{prop}}(L_1 \vee \cdots \vee L_n, N) = \bigcup_{i=1}^{n} \mathrm{HL}_{\mathrm{prop}}(L_i, N)$.

Heule et al. used a fixpoint computation, but our definition based on the reflexive transitive closure is equivalent. Intuitively, a hidden literal can be added to or removed from a clause without affecting its semantics in models of N. By eliminating hidden literals from C, we simplify it. By adding hidden literals to C, we might get a tautology C' (i.e., a valid clause: $\models C'$), meaning that $N{\downarrow_2} \models C$, thereby making C a candidate for deletion. Note that $\mathrm{HL}_{\mathrm{prop}}(L, N)$ is finite for a finite N.

Definition 3.2. Given $L' \vee L \vee C \in N$, if $L' \in \mathrm{HL}_{\mathrm{prop}}(L, N)$, HLE replaces N by $(N\backslash\{L' \vee L \vee C\}) \cup \{L \vee C\}$. Given $C \in N$, $\{L_1, \ldots, L_n\} = \mathrm{HL}_{\mathrm{prop}}(C, N)$, and $C' = C \vee L_1 \vee \cdots \vee L_n$, if C' is a tautology, HTE replaces N by $N\backslash\{C\}$.

Theorem 3.3. *The result of applying HLE or HTE to a clause set N is equivalent to N.*

Proof. For HLE, assume we have a model of N and $L' \vee L \vee C$ that does not satisfy $L \vee C$. Then the model must satisfy L', but since N implies $\neg L' \vee L$, it must also satisfy $L \vee C$, a contradiction.

For HTE, it can be shown that the sets N and $N \cup \{C'\}\backslash\{C\}$ are equivalent [28, Section 2.1]. Since the clause C' is a tautology, N and $N\backslash\{C\}$ are equivalent. □

We generalize hidden literals to first-order logic with equality by considering substitutivity of variables as well as congruence of equality.

Definition 3.4. Given a literal L and a clause set N, the set of *hidden literals* for L and N is $\mathrm{HL}(L, N) = \{L' \mid L' \hookrightarrow^* L\}\backslash\{L\}$, where $\hookrightarrow$ is defined so that (1) $\neg L'\sigma \hookrightarrow L\sigma$ if $L' \vee L \in N$ and σ is a substitution; (2) $s \approx t \hookrightarrow u[s] \approx u[t]$ for all terms s, t and contexts $u[\ ]$; and (3) $u[s] \not\approx u[t] \hookrightarrow s \not\approx t$ for all terms s, t and contexts $u[\ ]$. Moreover, $\mathrm{HL}(L_1 \vee \cdots \vee L_n, N) = \bigcup_{i=1}^{n} \mathrm{HL}(L_i, N)$.

The generalized definition also enjoys the key property that $L' \in \mathrm{HL}(L, N)$ implies $N{\downarrow_2} \models L' \longrightarrow L$. However, $\mathrm{HL}(L, N)$ may be infinite even for predicate literals; for example, $\mathrm{p}(\mathrm{f}^i(x)) \in \mathrm{HL}(\mathrm{p}(x), \{\mathrm{p}(x) \vee \neg\mathrm{p}(\mathrm{f}(x))\})$ for every i.

Based on Definition 3.4, we can generalize HLE and support a related technique called *failed literal elimination*:

$$\frac{L' \vee L \vee C}{L \vee C}\,\mathrm{HLE} \quad \text{if } L' \in \mathrm{HL}(L, N) \qquad\qquad \frac{L \vee C}{C}\,\mathrm{FLE} \quad \text{if } L', \neg L' \in \mathrm{HL}(\neg L, N)$$

Double lines denote *simplification rules*: When the premises appear in the clause set, the prover can use the redundancy criterion to replace them by the conclusions. The second rule, failed literal elimination, is inspired by the SAT technique of asserting $\neg L$ if L is a *failed literal* [23]. It is easy to see that rule HLE is sound. From $L' \in \mathrm{HL}(L, N)$ we have $N \models L' \longrightarrow L$ (i.e., $\neg L' \vee L$). Performing subsumption resolution [4] between $L' \vee L \vee C$ and $\neg L' \vee L$ yields the conclusion, which is therefore entailed by N. For FLE, the condition $L', \neg L' \in \mathrm{HL}(\neg L, N)$ means that $N{\downarrow_2} \models \{\neg L' \vee \neg L, L' \vee \neg L\} \models \neg L$.

Example 3.5. Consider the clause set $N = \{\mathrm{p}(x) \vee \neg\mathrm{p}(\mathrm{f}(x)),\ \mathrm{p}(\mathrm{f}(\mathrm{f}(x))) \vee \mathrm{a} \approx \mathrm{b}\}$ and the clause $C = \mathrm{f}(\mathrm{a}) \not\approx \mathrm{f}(\mathrm{b}) \vee \mathrm{p}(x)$. The first clause in N induces $\mathrm{p}(\mathrm{f}(x)) \hookrightarrow \mathrm{p}(x)$, $\mathrm{p}(\mathrm{f}(\mathrm{f}(x))) \hookrightarrow \mathrm{p}(\mathrm{f}(x))$, and hence $\mathrm{p}(\mathrm{f}(\mathrm{f}(x))) \hookrightarrow^* \mathrm{p}(x)$. Together with the second clause in N, it can be used to derive $\mathrm{a} \not\approx \mathrm{b} \hookrightarrow^* \mathrm{p}(x)$. Finally, using rule (3) of Definition 3.4, we derive $\mathrm{f}(\mathrm{a}) \not\approx \mathrm{f}(\mathrm{b}) \hookrightarrow^* \mathrm{p}(x)$—that is, $\mathrm{f}(\mathrm{a}) \not\approx \mathrm{f}(\mathrm{b}) \in \mathrm{HL}(\mathrm{p}(x), N)$. This allows us to remove C's first literal using HLE.

Two special cases of HLE exploit equality congruence as embodied by conditions (2) and (3) of Definition 3.4 without requiring to compute the HL set:

$$\frac{s \approx t \vee u[s] \approx u[t] \vee C}{u[s] \approx u[t] \vee C}\ \textsc{CongHLE}^{+}$$

$$\frac{s \not\approx t \vee u[s] \not\approx u[t] \vee C}{s \not\approx t \vee C}\ \textsc{CongHLE}^{-}$$

Hidden literals can be combined with unit clauses L' to remove more literals:

$$\frac{L' \quad L \vee C}{L' \quad C}\ \textsc{UnitHLE} \quad \text{if } L'\sigma \in \mathrm{HL}(\neg L, N)$$

Given a unit clause $L' \in N$, the rule uses it to discharge $L'\sigma$ in $N \models L'\sigma \longrightarrow \neg L$. As a result, we have $N \models \neg L$, making it possible to remove L from $L \vee C$.

Example 3.6. Consider the clause set $N = \{\mathrm{p}(x) \vee \mathrm{q}(\mathrm{f}(x)),\ \neg\mathrm{q}(\mathrm{f}(\mathrm{a})) \vee \mathrm{f}(\mathrm{b}) \approx \mathrm{g}(\mathrm{c}),\ \mathrm{f}(x) \not\approx \mathrm{g}(y)\}$ and the clause $C = \neg\mathrm{p}(\mathrm{a}) \vee \neg\mathrm{q}(\mathrm{b})$. The first clause in N induces $\neg\mathrm{q}(\mathrm{f}(\mathrm{a})) \hookrightarrow \mathrm{p}(\mathrm{a})$, whereas the second one induces $\mathrm{f}(\mathrm{b}) \not\approx \mathrm{g}(\mathrm{c}) \hookrightarrow \neg\mathrm{q}(\mathrm{f}(\mathrm{a}))$. Thus, we have $\mathrm{f}(\mathrm{b}) \not\approx \mathrm{g}(\mathrm{c}) \hookrightarrow^* \mathrm{p}(\mathrm{a})$—that is, $\mathrm{f}(\mathrm{b}) \not\approx \mathrm{f}(\mathrm{c}) \in \mathrm{HL}(\mathrm{p}(\mathrm{a}), N)$. By applying the substitution $\{x \mapsto \mathrm{b},\ y \mapsto \mathrm{c}\}$ to the third clause in N, we can fulfill the conditions of UnitHLE and remove C's first literal.

Next, we generalize hidden tautologies to first-order logic.

Definition 3.7. A clause C is a *hidden tautology* for a clause set N if there exists a finite set $\{L_1, \ldots, L_n\} \subseteq \mathrm{HL}(C, N)$ such that $C \vee L_1 \vee \cdots \vee L_n$ is a tautology.

In general, hidden tautologies are not redundant and cannot be deleted during saturation, as shown by the following example.

Example 3.8. Consider the unsatisfiable set $N = \{\neg\mathrm{a}, \neg\mathrm{b}, \mathrm{a} \vee \mathrm{c}, \mathrm{b} \vee \neg\mathrm{c}\}$, the order $\mathrm{a} \prec \mathrm{b} \prec \mathrm{c}$, and the empty selection function. The only possible superposition inference from N is between the last two clauses, yielding the clause $\mathrm{a} \vee \mathrm{b}$ (after simplifying away $\top \not\approx \top$). This clause is a hidden tautology: We have $\mathrm{HL}(\mathrm{a} \vee \mathrm{b}, N) = \mathrm{HL}(\mathrm{a}, N) \cup \mathrm{HL}(\mathrm{b}, N) = \{\neg\mathrm{a}, \neg\mathrm{b}\}$, and there exists the set $\{\neg\mathrm{a}\} \subseteq \mathrm{HL}(\mathrm{a} \vee \mathrm{b}, N)$ such that $\neg\mathrm{a} \vee \mathrm{a} \vee \mathrm{b}$ is a tautology. The hidden tautology $\mathrm{a} \vee \mathrm{b}$ is entailed by the larger clauses $\mathrm{a} \vee \mathrm{c}$ and $\mathrm{b} \vee \neg\mathrm{c}$. If the hidden tautology $\mathrm{a} \vee \mathrm{b}$ is removed, the prover could enter an infinite loop, forever generating and deleting the hidden tautology and never getting the opportunity to derive the empty clause.

In practice, most provers use a variant of the given clause procedure. Removing hidden tautologies breaks the invariant of the procedure that all inferences between clauses in $\mathcal{A}$ are redundant. The end result is not that the prover diverges, but that it terminates without deriving the empty clause.

To observe this, assume the setting as in Example 3.8, and let $\mathcal{P} = N$ and $\mathcal{A} = \emptyset$. After moving the first three clauses from $\mathcal{P}$ to $\mathcal{A}$ (leaving us with $\mathcal{P} = \{\mathrm{b} \vee \neg\mathrm{c}\}$ and $\mathcal{A} = \{\neg\mathrm{a}, \neg\mathrm{b}, \mathrm{a} \vee \mathrm{c}\}$), no inferences are possible, and no new clauses are added to $\mathcal{P}$. After the last clause is moved to $\mathcal{A}$, the

hidden tautology a ∨ b is produced. If it is deleted, the prover terminates with the unsatisfiable set $\mathcal{A}$, but does not derive the empty clause.

To delete hidden tautologies during saturation, the prover could check that all the relevant clause instances encountered along the computation of HL are $\prec$-smaller than a given hidden tautology. However, this would be expensive and seldom succeed, given that superposition creates lots of nonredundant hidden tautologies. Instead, we propose to simplify hidden tautologies using the following rules:

$$\frac{L \vee L' \vee C}{L \vee L'}\,\text{HTR} \quad \text{if } \neg L' \in \text{HL}(L, N) \text{ and } C \neq \bot \qquad \frac{L \vee C}{L}\,\text{FLR} \quad \text{if } L', \neg L' \in \text{HL}(L, N) \text{ and } C \neq \bot$$

We call these techniques *hidden tautology reduction* and *failed literal reduction*, respectively. Both rules are sound. As with hidden literals, unit clauses L' can be exploited:

$$\frac{L' \quad L \vee C}{L' \quad L}\,\textsc{UnitHTR} \quad \text{if } L'\sigma \in \text{HL}(L, N) \text{ and } C \neq \bot$$

We give the simplification rules above (for HLE, hidden tautology reduction, failed literal detection, and their variants) the collective name of *hidden-literal-based elimination* (HLBE). Yet another use of hidden literals is for *equivalent literal substitution* [27]: If both $L' \in \text{HL}(L, N)$ and $L \in \text{HL}(L', N)$, we can often simplify $L'\sigma$ to $L\sigma$ in N if $L'\sigma \succ L\sigma$. We want to investigate this further.

Theorem 3.9. *The rules HLE, FLE, CongHLE$^+$, CongHLE$^-$, UnitHLE, HTR, FLR, and UnitHTR are sound simplification rules.*

Proof. It is easy to see that the deleted premises are entailed by the conclusions that replace them and that the conclusions' instances are $\prec$-smaller than the premises' instances, as required by the redundancy criterion. It remains to check soundness.

Case HLE: We have $N, L' \models L$ by the side condition and must show $N, L' \vee L \vee C \models L \vee C$. Let $\mathcal{J} \models N, L' \vee L \vee C$. If $\mathcal{J} \models L'$, then we also have $\mathcal{J} \models L$ thanks to the side condition and hence $\mathcal{J} \models L \vee C$. Otherwise, we have $\mathcal{J} \models L \vee C$, which is exactly what we need to show.

Case FLE: We have $N, L \models L'$ and $N, L \models \neg L'$ by the side condition. If $\mathcal{J} \models N, L$, then both $\mathcal{J} \models L'$ and $\mathcal{J} \models \neg L'$, an absurdity. Otherwise, we have $\mathcal{J} \models C$, as desired.

Case CongHLE$^+$, CongHLE$^-$: Obvious by congruence of equality.

Case UnitHLE: We have $N, L \models \neg L'\sigma$ by the side condition. If $\mathcal{J} \models N, L$, then $N \models \neg L'\sigma$. But since $L' \in N$, this is an absurdity. Otherwise, we have $\mathcal{J} \models C$, as desired.

Case HTR: We have $N, \neg L' \models L$ by the side condition. If either $\mathcal{J} \models L$ or $\mathcal{J} \models L'$, the desired result follows directly. Otherwise, from $\mathcal{J} \models \neg L'$ we also have $\mathcal{J} \models L$ thanks to the side condition, contradicting $\mathcal{J} \models \neg L$.

Case FLR: We have $N, L' \models L$ and $N, \neg L' \models L$ by the side condition. Hence $N \models L$, as desired.

Case UnitHTR: We have $N, L'\sigma \models L$. Since $L' \in N$, we have $N \models L$, as desired. □

4 PREDICATE ELIMINATION

For propositional logic, variable elimination [19] is one of the main preprocessing and inprocessing techniques. Following Gabbay and Ohlbach's ideas [24], Khasidashvili and Korovin [30] generalized variable elimination to first-order logic with equality and demonstrated that it is effective as a preprocessor. We propose an improvement—defined predicate elimination—that can both eliminate more predicates and generate fewer clauses—and show that, with a minor restriction, predicate elimination can be integrated in a superposition prover without compromising its refutational completeness.

Khasidashvili and Korovin's (K&K) preprocessing technique removes singular predicates (which they call "non-self-referential predicates") from the problem using so-called flat resolution.

Definition 4.1. A predicate symbol is called *singular* for a clause set N if it occurs at most once in every clause contained in N.

Definition 4.2. Let $C = \mathrm{p}(\vec{s}_n) \vee C'$ and $D = \neg\mathrm{p}(\vec{t}_n) \vee D'$ be clauses with no variables in common. The clause $s_1 \not\approx t_1 \vee \cdots \vee s_n \not\approx t_n \vee C' \vee D'$ is a *flat resolvent* of C and D on p.

Given two clause sets M, N, predicate elimination iteratively replaces clauses from N containing the symbol p with all flat resolvents against clauses in M. Eventually, it yields a set with no occurrences of p. The sets M and N can be initialized to be the same set, but they can also be different sets; for example, with defined predicate elimination, M will be a so-called definition set, whereas N will be an arbitrary clause set in which p may be nonsingular.

Definition 4.3. Let M, N be clause sets and p be a singular predicate for M. Let $\rightsquigarrow$ be the following relation on clause set pairs and clause sets:

(1) $(M, \{(\neg)\mathrm{p}(\vec{s}) \vee C'\} \uplus N) \rightsquigarrow (M,\ N' \cup N)$ if N' is the set that consists of all clauses (up to variable renaming) that are flat resolvents with $(\neg)\mathrm{p}(\vec{s}) \vee C'$ on p and a clause from M as premises. The premises' variables are renamed apart.

(2) $(M, N) \rightsquigarrow N$ if N has no occurrences of p.

The *resolved set* $M \rtimes_{\mathrm{p}} N$ is the clause set N' such that $(M, N) \rightsquigarrow^* N'$.

Lemma 4.4. *Let M, N be clause sets and* p *be a singular predicate for M. The resolved set N' is reached in a finite number of $\rightsquigarrow$ steps, and it is unique up to variable renaming.*

Proof. To show $\rightsquigarrow$ is terminating we use the following ordinal measure on clause sets: $\nu(\{D_1, \ldots, D_n\}) = \omega^{\nu(D_1)} \oplus \cdots \oplus \omega^{\nu(D_n)}$, where $\nu(D)$ is the number of occurrences of p in D, ω is the first infinite ordinal, and $\oplus$ is the Hessenberg, or natural, sum, which is commutative. For every transition $(M, \{C\} \cup N) \rightsquigarrow (M, N' \cup N)$, we have $\nu(\{C\} \cup N) > \nu(N' \cup N)$ because $\omega^{\nu(C)} > \omega^{\nu(C)-1} \cdot |N'|$. Eventually, a state (M, N') with $\nu(N') = \omega^0 \cdot n$ is reached. Then, we apply the second rule of Definition 4.3 to obtain the resolved set N'.

To show that N' is unique, i.e., $\rightsquigarrow$ is confluent, it suffices to show (since $\rightsquigarrow$ is terminating and Newmann's lemma applies [2]) that $\rightsquigarrow$ is locally confluent. In other words, whenever $(M, N) \rightsquigarrow (M, N_1)$ and $(M, N) \rightsquigarrow (M, N_2)$, there exists a set N' such that $(M, N_1) \rightsquigarrow (M, N')$ and $(M, N_2) \rightsquigarrow (M, N')$.

There are two main sources of nondeterminism of $\rightsquigarrow$: The choice of $C \in N$ and the choice of the literal in C to act on. Let us focus on the choice of C in N; the same discussion applies for the choice of literal in C.

Let $N = \{C_1\} \uplus \{C_2\} \uplus N'$, where C_1 and C_2 are clauses with occurrences of p. Then, $(M, \{C_1\} \uplus \{C_2 \cup N'\}) \rightsquigarrow (M, N_1' \cup \{C_2 \cup N'\})$ and $(M, \{C_2\} \uplus \{C_1 \cup N'\}) \rightsquigarrow (M, N_2' \cup \{C_1 \cup N'\})$, where N_1' and

N_2' are sets of corresponding resolvents. Both $\rightsquigarrow$ steps can be joined (up to variable renaming) to $(M, N_1' \cup N_2' \cup N')$, showing that $\rightsquigarrow$ is locally confluent. □

Next, it is useful to partition clause sets into subsets based on the presence and polarity of a singular predicate.

Definition 4.5. Let N be a clause set and p be a singular predicate for N. Let N_p^+ consist of all clauses of the form $\mathsf{p}(\vec{s}) \vee C' \in N$, let N_p^- consist of all clauses of the form $\neg\mathsf{p}(\vec{s}) \vee C' \in N$, let $N_\mathsf{p} = N_\mathsf{p}^+ \cup N_\mathsf{p}^-$, and let $\overline{N}_\mathsf{p} = N \backslash N_\mathsf{p}$.

Definition 4.6. Let N be a clause set and p be a singular predicate for N. SPE of p in N replaces N by $\overline{N}_\mathsf{p} \cup (N_\mathsf{p}^+ \bowtie_\mathsf{p} N_\mathsf{p}^-)$.

The result of SPE is satisfiable if and only if N is satisfiable [30, Theorem 1], justifying SPE's use in a preprocessor. However, eliminating singular predicates aggressively can dramatically increase the number of clauses. To prevent this, Khasidashvili and Korovin suggested to replace N by N' only if $\lambda(N') \leq \lambda(N)$ and $\mu(N') \leq \mu(N)$, where $\lambda(N)$ is the number of literals in N and $\mu(N)$ is the sum for all clauses $C \in N$ of the square of the number of distinct variables in C.

Compared with what modern SAT solvers use, this criterion is fairly restrictive. We relax it to make it possible to eliminate more predicates, within reason. Let $K_\mathrm{tol} \in \mathbb{N}$ be a tolerance parameter. A predicate elimination step from N to N' is allowed if $\lambda(N') < \lambda(N) + K_\mathrm{tol}$ or $\mu(N') < \mu(N)$ or $|N'| < |N| + K_\mathrm{tol}$. A refinement, which we will not study, would be to gradually increment the tolerance K_tol, as is done in some SAT solvers.

SPE is effective, but an important refinement has not yet been adapted to first-order logic: variable elimination by substitution. Eén and Biere [19] discovered that a propositional variable x can be eliminated without computing all resolvents if it is expressible as an equivalence $\mathsf{x} \longleftrightarrow \varphi$, where φ, the "gate," is an arbitrary formula that does not reference x. They extract from a set N of propositional clauses into a definition set G, essentially the clausification of $\mathsf{x} \longleftrightarrow \varphi$, and $R = N_\mathsf{p} \backslash G$, the remaining clauses containing p. To eliminate x from N while preserving satisfiability, it suffices to resolve clauses from G against clauses from R, effectively substituting φ for x in R. Crucially, we do not need to resolve pairs of clauses from G or pairs of clauses from R. We generalize this idea to first-order logic.

Definition 4.7. Let G be a clause set and p be a predicate symbol. The set G is a *definition set* for p if (1) p is singular for G; (2) G consists of clauses of the form $(\neg)\mathsf{p}(\vec{x}) \vee C'$ (up to variable renaming), where the variables $\vec{x}$ are distinct; (3) the variables in C' are all among p's arguments $\vec{x}$; (4) all clauses in $G_\mathsf{p}^+ \bowtie_\mathsf{p} G_\mathsf{p}^-$ are tautologies; and (5) $E(\vec{\mathsf{c}})$ is unsatisfiable, where the *environment* $E(\vec{x})$ consists of all subclauses C' of any $(\neg)\mathsf{p}(\vec{x}) \vee C' \in G$ and $\vec{\mathsf{c}}$ is a tuple of distinct fresh constants substituted in for $\vec{x}$.

A definition set G corresponds intuitively to a definition by cases in mathematics—e.g.,

$$\mathsf{p}(\vec{x}) = \begin{cases} \top & \text{if } \varphi(\vec{x}) \\ \bot & \text{if } \psi(\vec{x}) \end{cases}$$

Part (4) states that the case conditions are mutually exclusive (e.g., $\neg\varphi(\vec{x}) \vee \neg\psi(\vec{x})$), and part (5) states that they are exhaustive (e.g., $\nexists\vec{\mathsf{c}}.\ \neg\varphi(\vec{\mathsf{c}}) \wedge \neg\psi(\vec{\mathsf{c}})$). Given a quantifier-free formula $\mathsf{p}(\vec{x}) \longleftrightarrow \varphi(\vec{x})$ with distinct variables $\vec{x}$ such that $\varphi(\vec{x})$ does not contain p, any reasonable clausification algorithm would produce a definition set for p.

Example 4.8. Given the formula $\mathsf{p}(x) \longleftrightarrow \mathsf{q}(x) \wedge (\mathsf{r}(x) \vee \mathsf{s}(x))$, a standard clausification algorithm [37] produces $\{\neg\mathsf{p}(x) \vee \mathsf{q}(x), \neg\mathsf{p}(x) \vee \mathsf{r}(x) \vee \mathsf{s}(x), \mathsf{p}(x) \vee \neg\mathsf{q}(x) \vee \neg\mathsf{r}(x), \mathsf{p}(x) \vee \neg\mathsf{q}(x) \vee \neg\mathsf{s}(x)\}$, which qualifies as a definition set for p.

Definition sets generalize Eén and Biere's gates. They can be recognized syntactically for formulas such as $\mathrm{p}(\vec{x}) \leftrightarrow \bigvee_i \mathrm{q}_i(\vec{s}_i)$ or $\mathrm{p}(\vec{x}) \leftrightarrow \bigwedge_i \mathrm{q}_i(\vec{s}_i)$, or semantically: Condition (4) can be checked using the congruence closure algorithm, and condition (5) amounts to a propositional unsatisfiability check.

The key result about propositional gates carries over to definition sets.

Definition 4.9. Let N be a clause set, p be a predicate symbol, $G \subseteq N$ be a definition set for p, and $R = N_{\mathrm{p}} \backslash G$. *Defined predicate elimination* (DPE) of p in N replaces N by $\overline{N}_{\mathrm{p}} \cup (G \rtimes_{\mathrm{p}} R)$.

Lemma 4.10. *Let $N(\vec{x})$ be a clause set such that the variables of all clauses in it are among the argument n-tuple $\vec{x}$, and let $\vec{\mathrm{c}}$ be an n-tuple of distinct fresh constants. If $N(\vec{\mathrm{c}})$ (i.e., $N(\vec{x})\{\vec{x} \mapsto \vec{\mathrm{c}}\}$) is unsatisfiable, then for every interpretation $\mathcal{J}$ and valuation ξ, $\mathcal{J} \not\models_\xi N$.*

Proof. We show the contrapositive. Assume that for some $\mathcal{J}$ and ξ, $\mathcal{J} \models_\xi N(\vec{x})$. Then let $\mathcal{J}'$ be a model that assigns each c_i the interpretation of x_i under $\mathcal{J}$ and ξ, and otherwise coincides with $\mathcal{J}$. We obtain $\mathcal{J}' \models N(\vec{\mathrm{c}})$. □

Lemma 4.11. *Let G be a definition set for* p *and N be an arbitrary clause set. If $(G, N) \rightsquigarrow (G, N')$ for some N', then $G \cup N$ and $G \cup N'$ are equivalent.*

Proof. Since flat resolution is sound, the nontrivial direction is to show that a model $\mathcal{J}$ of the set $G \cup N'$ is also a model of $G \cup N$. Since the only clause in $N \backslash N'$ is the clause $C = (\neg)\mathrm{p}(\vec{s}_n) \vee C'$ on which the $\rightsquigarrow$ step is performed, it suffices to show $\mathcal{J} \models C$.

Without loss of generality, we assume that the leading literal of C is positive. Toward a contradiction, assume ξ is a valuation such that $\mathcal{J} \not\models_\xi C$. Then, $\mathcal{J} \not\models_\xi \mathrm{p}(\vec{s}_n)$. Consider an arbitrary clause $D = \mathrm{p}(\vec{x}_n) \vee D' \in G^+_{\mathrm{p}}$ and a valuation ξ' that assigns each x_i the interpretation of s_i under $\mathcal{J}$ and ξ. As $\mathcal{J} \not\models_{\xi'} \mathrm{p}(\vec{x}_n)$ and $\mathcal{J} \models G$, then $\mathcal{J} \models_{\xi'} D'$ for every such clause D. However, by part (5) of Definition 4.7 and by Lemma 4.10, $\mathcal{J} \not\models_{\xi'} E(\vec{x}_n)$, where $E(\vec{x}_n)$ is the environment associated with the definition set G. Therefore, there must exist a clause $D = \neg\mathrm{p}(\vec{x}_n) \vee D'$ in G^-_{p} such that $\mathcal{J} \not\models_\xi D'$.

Now consider the flat resolvent of C and D on p: $R = x_1 \not\approx s_1 \vee \cdots \vee x_n \not\approx s_n \vee C' \vee D'$. Let ζ be a valuation coinciding with ξ on variables of C and with ξ' on $\vec{x}_n$. Clearly, $\mathcal{J} \not\models_\zeta R$. Yet, $R \in N'$, and as $\mathcal{J} \models N'$, we reach a contradiction. □

Lemma 4.12. *Let G be a definition set for* p *and N be a clause set with no occurrences of* p. *Then $G \cup N$ is satisfiable if and only if N is satisfiable.*

Proof. The nontrivial direction is to show that if N is satisfiable, $G \cup N$ is as well. Let $\mathcal{J}$ be a model of N. We construct a model $\mathcal{J}'$ of G over the same universe as $\mathcal{J}$. For any atom A such that p does not occur in A and for every ξ, we set $\mathcal{J}' \models_\xi A$ if and only if $\mathcal{J} \models_\xi A$. For any clause $\mathrm{p}(\vec{x}_n) \vee C' \in G$ and any assignment ξ such that $\mathcal{J} \not\models_\xi C'$, we define $\mathcal{J}'$ so that $\mathcal{J}' \models_\xi \mathrm{p}(\vec{x}_n)$. By construction, $\mathcal{J}' \models G^+_{\mathrm{p}} \cup N$. It remains to show that $\mathcal{J}' \models G^-$.

Let $C = \neg\mathrm{p}(\vec{x}_n) \vee C' \in G$ and let ξ be an arbitrary assignment. Toward a contradiction, assume $\mathcal{J}' \not\models_\xi C$, and consequently $\mathcal{J}' \models_\xi \mathrm{p}(\vec{x}_n)$. By construction of $\mathcal{J}'$, there exists a clause $\mathrm{p}(\vec{y}_n) \vee D' \in G$ and an assignment ξ' that assigns each y_i value of $\xi(x_i)$ such that $\mathcal{J} \not\models_{\xi'} D'$. The resolvent $R = x_1 \not\approx y_1 \vee \cdots \vee x_n \not\approx y_n \vee C' \vee D'$ is a tautology, according to condition (4) of Definition 4.7. However, for a valuation that behaves like ξ on $\vec{x}$ and ξ' on $\vec{y}$, $\mathcal{J}'$ does not satisfy $R \in N$, contradicting our assumption. □

Theorem 4.13. *The result of applying DPE to a clause set N is satisfiable if and only if N is satisfiable.*

Proof. Let p be a predicate symbol and $G \subseteq N$ be the definition set used by DPE, and let $R = N_p \backslash G$.

Using Lemma 4.4, we get that there is a derivation $(G, R) \rightsquigarrow^n (G, R') \rightsquigarrow R'$. Applying Lemma 4.11 n times, we get that $G \cup R$ is equivalent to $G \cup R'$. Finally, Lemma 4.12 gives us the desired result. □

Since there will typically be at most only a few defined predicates in the problem, it makes sense to fall back on SPE when no definition is found.

Definition 4.14. Let N be a clause set and p be a predicate symbol. If there exists a definition set $G \subseteq N$ for p, *portfolio predicate elimination* (PPE) on p in N replaces N with $\overline{N}_p \cup (G \rtimes_p R)$, where $R = N_p \backslash G$. Otherwise, if p is singular in N, it results in $\overline{N}_p \cup (N_p^+ \rtimes_p N_p^-)$. In all other cases, it is not applicable.

Hidden-literal-based techniques fit within the traditional framework of saturation, because they delete or reduce a clause based on the *presence* of other clauses. In contrast, predicate elimination relies on the *absence* of clauses from the proof state. We can still integrate it with superposition as follows: At every kth iteration of the given clause procedure, perform predicate elimination on $\mathcal{A} \cup \mathcal{P}$, and add all new clauses to $\mathcal{P}$.

One may wonder whether such an approach preserves the refutational completeness of the calculus. The answer is no.

To see why, consider the following *binary splitting* rule based on Riazanov and Voronkov [41]:

$$\frac{C \vee D}{\mathrm{p} \vee C \quad D \vee \neg \mathrm{p}}\,\mathrm{BS}$$

Provisos: C and D have no free variables in common, p is fresh, and p is $\prec$-smaller than C and D. Since the conclusions are smaller than the premise, the rule can be applied aggressively as a simplification. But notice that the effect of splitting can be undone by SPE, possibly giving rise to loops BS, SPE, BS, SPE, ...

Under which conditions is predicate elimination refutationally complete? To answer this question, we employ the saturation framework of Waldmann, Tourret, Robillard, and Blanchette [50, 51]. Let (*FInf*, *Red*) be the base calculus without predicate elimination—e.g., resolution or superposition inferences together with the standard redundancy criterion [4, Section 4.2]. The inference system *FInf* is a set of inferences $(C_n, \ldots, C_1, C_0)$, for $n \geq 1$, where $C_n, \ldots, C_1$ are the premises and C_0 is the conclusion. C_1 is called the main premise. The redundancy criterion is a pair $Red = (Red_I, Red_F)$ where Red_I determines which inferences can be omitted and Red_F is used to remove clauses.

Next, consider an abstract proving process working on a single clause set. Let $\triangleright_{Red}$ denote the transition relation that supports (1) adding arbitrary clauses and (2) removing clauses deemed useless by Red_F. Typically, the added clauses are the result of performing inferences and are entailed by the premises, but other clauses can be added as well. A $\triangleright_{Red}$*-derivation* is a finite or infinite sequence of clause sets $N_0 \triangleright_{Red} N_1 \triangleright_{Red} \cdots$.

We fix a finite set $\mathbf{P}$ of predicate symbols that may be subjected to predicate elimination. These might include all the predicate symbols occurring in the input problem, but exclude any symbols introduced by splitting or other rules. Given a clause or clause set N, we write $\mathbf{P}(N)$ to denote the set of all predicate symbols from $\mathbf{P}$ occurring in N. Let $\triangleright_{\mathrm{p}}$ denote the elimination of a singular or defined predicate symbol from $\mathbf{P}$. A *mixed derivation* consists of transitions either of the form $N \triangleright_{\mathrm{p}} N'$ or of the form $N \triangleright_{Red} N'$ where $\mathbf{P}(N) \supseteq \mathbf{P}(N')$. Because $\mathbf{P}$ is finite, any mixed derivation consists of at most finitely many $\triangleright_{\mathrm{p}}$ transitions. Hence, in any derivation, there exists an index k from which all transitions are standard $\triangleright_{Red}$-transitions.

This suggests the following path to completeness: Pretend that the transitions between N_0 and N_k are merely preprocessing and start the actual derivation at N_k. This works at the abstract level of derivations on single clause sets. It fails, however, for an actual saturation prover that distinguishes between passive and active clauses.

Example 4.15. The counterexample below is based on the given clause prover GC from the saturation framework. It shows how predicate elimination can break GC's key invariant, which states that all inferences between active clauses are redundant. Breaking the invariant means that the limit might be unsaturated, breaking the refutational completeness proof.

We use superposition with the order $a \prec b \prec c \prec d$ and without selection. Assume $a \in \mathbf{P}$ and suppose we start with the satisfiable clause set

$$\neg a \vee \boxed{d} \qquad \neg a \vee \boxed{\neg d} \qquad a \vee b \vee \boxed{c} \qquad c \vee \boxed{d} \qquad b \vee \boxed{\neg d}$$

where gray boxes mark maximal literals. Suppose the prover makes $c \vee d$ and $b \vee \neg d$ active. From these two clauses, a superposition inference ι could derive the conclusion $b \vee c$. However, the three passive clauses are $\prec$-smaller than ι's main premise $b \vee \neg d$ and collectively entail ι's conclusion. This means that ι is redundant and can be ignored.

If the prover now eliminates the predicate a using SPE, the passive set is reduced to $\{b \vee c \vee d,\ b \vee c \vee \neg d\}$. Either clause is subsumed by an active clause, so the prover deletes it. It stops with the active set $\{c \vee d,\ b \vee \neg d\}$, which is unsaturated because ι is no longer redundant. The invariant is broken.

Example 4.16. In Example 4.15, the initial clause set was satisfiable. If it is unsatisfiable, we can even lose refutational completeness. To see why, we add the unit clauses $\neg b$ and $\neg c$ to the initial clause set of Example 4.15 to make it unsatisfiable. We repeat the same steps as above, including the subsumptions at the end, yielding the passive set $\{\boxed{\neg b}, \boxed{\neg c}\}$ and the active set $\{c \vee \boxed{d},\ b \vee \boxed{\neg d}\}$. Then, making $\neg b$ and $\neg c$ active triggers no inferences. The prover stops with an unsatisfiable four-clause active set that does not contain the empty clause.

A solution could be to move all active clauses to the passive set at step k or later, but this would be costly, since it would force the prover to redo inferences whose conclusions might then have to be simplified or subsumed again. Instead, we salvage the existing completeness proof for GC, by resolving the issues concerning splitting and the GC invariant. Our approach is to weaken the redundancy criterion slightly, enough both to disable splitting on **P**-predicates and to ensure that inferences such as ι in Examples 4.15 and 4.16 are performed. The required weakening is so mild that it does not invalidate any practical simplification or subsumption techniques we are aware of, except of course splitting.

In accordance with the saturation framework, let **F** be the set of first-order Σ-clauses, let **G** be its ground subset, and let $\mathcal{G}$ be a function that maps an **F**-clause to the set of its **G**-clause instances and analogously for **F**-inferences. We define an extension $\mathbf{G}^\flat$ of **G** for $\Sigma^\flat$-clauses in an ad hoc nonclassical logic reminiscent of paraconsistent logics [13]. The signature $\Sigma^\flat$ extends Σ with a distinguished predicate symbol $\bot$ that is interpreted differently from $\top$. For $\Sigma^\flat$, the Boolean type o may be interpreted as any set of cardinality at least 2.

The objective is to disallow the entailment that makes splitting and Examples 4.15 and 4.16 possible. The weaker entailment will be used to define a weaker redundancy criterion, which will curtail splitting. Properties such as soundness of the calculus can still be established using the standard entailment notion.

Definition 4.17. The operator $^\flat$ translates Σ-literals to $\Sigma^\flat$-literals as follows, where $\mathrm{p} \in \mathbf{P}$, $\mathrm{q} \notin \mathbf{P}$, and s, t are non-Boolean terms:

$$\begin{array}{lll} \mathrm{p}(\vec{t})^\flat = \mathrm{p}(\vec{t}) \not\approx \bot & \mathrm{q}(\vec{t})^\flat = \mathrm{q}(\vec{t}) \approx \top & (s \approx t)^\flat = s \approx t \\ \neg\, \mathrm{p}(\vec{t})^\flat = \mathrm{p}(\vec{t}) \not\approx \top & \neg\, \mathrm{q}(\vec{t})^\flat = \mathrm{q}(\vec{t}) \not\approx \top & (s \not\approx t)^\flat = s \not\approx t \end{array}$$

The operator is lifted elementwise to **G**-clauses and **G**-clause sets. The *weak entailment* $\models^\flat$ over **G**-clause sets are defined via an encoding into $\Sigma^\flat$-clauses: $M \models^\flat N$ if and only if $M^\flat \models N^\flat$. The lifting to **F**-clauses and **F**-clause set is achieved in the standard way via grounding.

The following property of weak entailment will allow us to eliminate **P**-predicates without losing completeness:

Lemma 4.18. *Let C be a clause that contains the predicate symbol $\mathrm{p} \in \mathbf{P}$ and D be a clause that does not contain* p. *If $N \cup \{C\} \models^\flat \{D\}$, then $N \models^\flat \{D\}$.*

Proof. Suppose $\mathcal{J} \models N^\flat$. We will define $\mathcal{J}'$ so that $\mathcal{J}' \models N^\flat \cup \{C^\flat\}$, retrieve $\mathcal{J}' \models D^\flat$, and then argue that $\mathcal{J} \models D^\flat$. We take $\mathcal{J}'$ to coincide with $\mathcal{J}$ except that we extend the domain for o with one fresh value and use this value as the interpretation of $\mathrm{p}(\vec{t})$ for all argument tuples $\vec{t}$. This modification makes any p literal of $C^\flat$ true, and it preserves the truth of $N^\flat$. By the hypothesis, $\mathcal{J}' \models D^\flat$. And since p does not occur in D, we have $\mathcal{J} \models D^\flat$. □

Note that the above lemma does not hold for classical entailment $\models$; indeed, $\{\mathrm{p} \vee \mathrm{q},\ \neg \mathrm{p} \vee \mathrm{q}\} \models \{\mathrm{q}\}$ yet $\{\mathrm{p} \vee \mathrm{q}\} \not\models \{\mathrm{q}\}$. On the other hand, the law of excluded middle does hold for weak entailment: $\models^\flat \mathrm{p} \vee \neg\, \mathrm{p}$. In fact, all classical clausal tautologies are tautologies for $\models^\flat$.

The standard redundancy criterion *Red* is obtained by lifting a criterion on **G**-clauses to **F**-clauses. The same construction can be replicated using $\models^\flat$ instead of $\models$, yielding the *weak redundancy criterion* $Red^\flat$. It is easy to check that the usual simplification techniques implemented in superposition provers can be justified using $Red^\flat$. Specifically, this concerns the following rules described by Schulz [42, Sections 2.3.1 and 2.3.2]: deletion of duplicated literals, deletion of resolved literals, syntactic tautology deletion, semantic tautology deletion, rewriting of negative literals, positive simplify-reflect, negative simplify-reflect, clause subsumption, and equality subsumption. Moreover, rewriting of positive literals is possible if the rewriting clause is smaller than the rewritten clause (a condition that is also needed with $\models$ but omitted by Schulz). Finally, destructive equality resolution cannot be justified with $\models$, let alone $\models^\flat$.

We instantiate the saturation framework with $(FInf, Red^\flat)$ to obtain a given clause prover GC. The prover operates on sets of labeled clauses (C, l), where C is a standard clause and $l \in \mathbf{L}$ is a label. The active label identifies active clauses; all other clauses are passive. The prover takes the form of two rules, Process and Infer, restricted to prevent the introduction of **P**-predicates. We extend it with a third rule, PredElim, for predicate elimination, and call the extended prover GCP. The rules are as follows, using again the framework notations:

Process $\quad \mathcal{N} \cup \mathcal{M} \Longrightarrow_{\mathrm{GCP}} \mathcal{N} \cup \mathcal{M}'$
where $\mathcal{M} \subseteq LRed_{\mathrm{F}}^{\flat,\neg}(\mathcal{N} \cup \mathcal{M}')$, $\mathcal{M}'{\downarrow_{\mathrm{active}}} = \emptyset$, and
$\mathbf{P}(\lfloor \mathcal{M}' \rfloor) \subseteq \mathbf{P}(\lfloor \mathcal{N} \cup \mathcal{M} \rfloor)$

Infer $\quad \mathcal{N} \cup \{(C, l)\} \Longrightarrow_{\mathrm{GCP}} \mathcal{N} \cup \{(C, \mathrm{active})\} \cup \mathcal{M}$
where $l \neq \mathrm{active}$, $\mathcal{M}{\downarrow_{\mathrm{active}}} = \emptyset$,
$FInf(\lfloor \mathcal{N}{\downarrow_{\mathrm{active}}} \rfloor, \{C\}) \subseteq Red_{\mathrm{I}}^{\flat \cap \mathcal{G}}(\lfloor \mathcal{N} \rfloor \cup \{C\} \cup \lfloor \mathcal{M} \rfloor)$, and
$\mathbf{P}(\lfloor \mathcal{M} \rfloor) \subseteq \mathbf{P}(\lfloor \mathcal{N} \rfloor \cup \{C\})$

PredElim $\quad \mathcal{N} \cup \mathcal{M} \Longrightarrow_{\mathrm{GCP}} \mathcal{N} \cup \mathcal{M}'$
where $\mathcal{N} \cup \mathcal{M} \rhd_{\mathrm{P}} \mathcal{N} \cup \mathcal{M}'$ and $\mathcal{M}'{\downarrow_{\mathrm{active}}} = \emptyset$

Here is a summary of the main framework notations:

- The letters $\mathcal{M}, \mathcal{N}$ range over sets of labeled clauses. $\mathcal{M}{\downarrow_l}$ denotes the subset of clauses in $\mathcal{M}$ labeled with l. The operator $\lfloor\ \rfloor$ erases all labels in a labeled clause or clause set.
- $\mathit{FInf}(N)$ denotes the set of all base calculus inferences with premises in N, and $\mathit{FInf}(N, M) = \mathit{FInf}(N \cup M) \backslash \mathit{FInf}(N \backslash M)$. The same notations are also available for the straightforward extension FLInf of FInf with labels.
- $\mathit{LRed}^{\flat,\sqsupset}$ is the extension of the standard redundancy criterion $\mathit{Red}^{\flat}$ defined using $\models^{\flat}$ to nonground labeled clauses with subsumption ($\sqsubset$).
- Given a sequence $(\mathcal{N}_i)_i$, its *limit* (*inferior*) is $\mathcal{N}_\infty = \bigcup_i \bigcap_{j \geq i} \mathcal{N}_j$.

The completeness proof follows the invariance-based argument found in the article by Waldmann et al. [51] about the saturation framework.

Lemma 4.19. *Every $\Longrightarrow_{\mathrm{GCP}}$-derivation is a mixed derivation.*

Proof. The cases for Process and Infer are almost as in Waldmann et al., with adjustments to show that P-predicates cannot appear from nowhere. The case for ElimPred is trivial. □

Let $\mathit{Inv}^{\flat}_{\mathcal{N}}(k)$ denote the condition $\mathit{FLInf}(\mathcal{N}_k{\downarrow_{\mathrm{active}}}) \subseteq \mathit{LRed}^{\flat}_{\mathrm{I}}(\mathcal{N}_k)$. Notice the difference with the definition of the key invariant $\mathit{Inv}_{\mathcal{N}}$ in the saturation framework, whose right-hand side is $\bigcup_{i=0}^{k} \mathit{Red}^{\mathbf{L}}_{\mathrm{I}}(\mathcal{N}_i)$. We cannot use the big union $\bigcup$ starting at $i = 0$ because we will need to truncate a sequence prefix of an a priori unknown length. The argument will still work thanks to monotonicity properties of redundancy criteria.

Lemma 4.20. *Let $(\mathcal{N}_i)_i$ be a $\Longrightarrow_{\mathrm{GCP}}$-derivation. If $\mathcal{N}_0{\downarrow_{\mathrm{active}}} = \emptyset$, then $\mathit{Inv}^{\flat}_{\mathcal{N}}(k)$ holds for all indices k.*

Proof. The base case is as in Waldmann et al.

For Process and Infer, the proof is essentially as in Waldmann et al., except that we also need to show that $\mathit{LRed}^{\flat}_{\mathrm{I}}(\mathcal{N}_k) \subseteq \mathit{LRed}^{\flat}_{\mathrm{I}}(\mathcal{N}_{k+1})$. This is a consequence of $\mathcal{N}_k \rhd_{\mathit{LRed}^{\flat,\sqsupset}} \mathcal{N}_{k+1}$ and of properties (R2) and (R3) of redundancy criteria.

A new case to consider is that of a PredElim transition $\mathcal{N}_k \Longrightarrow_{\mathrm{GCP}} \mathcal{N}_{k+1}$. Let $\mathcal{N}_k = \mathcal{N} \cup \mathcal{M} \Longrightarrow_{\mathrm{GCP}} \mathcal{N} \cup \mathcal{M}' = \mathcal{N}_{k+1}$, where $\mathcal{N} \cup \mathcal{M} \rhd_{\mathrm{p}} \mathcal{N} \cap \mathcal{M}'$ and $\mathcal{M}'{\downarrow_{\mathrm{active}}} = \emptyset$. We assume without loss of generality that $\mathcal{M} \cap \mathcal{M}' = \emptyset$. Let p be the eliminated predicate. Note that p occurs in every clause in $\mathcal{M}$ but in none of the clauses in $\mathcal{N}$ or $\mathcal{M}'$. We must show $\mathit{FLInf}(\mathcal{N}_{k+1}{\downarrow_{\mathrm{active}}}) \subseteq \mathit{LRed}^{\flat}_{\mathrm{I}}(\mathcal{N}_{k+1})$. As for Process, we have the inclusion $\mathit{FLInf}(\mathcal{N}_{k+1}{\downarrow_{\mathrm{active}}}) \subseteq \mathit{FLInf}(\mathcal{N}_k{\downarrow_{\mathrm{active}}})$, by the side condition that $\mathcal{M}'{\downarrow_{\mathrm{active}}} = \emptyset$. Moreover, by the induction hypothesis, $\mathit{FLInf}(\mathcal{N}_k{\downarrow_{\mathrm{active}}}) \subseteq \mathit{LRed}^{\flat}_{\mathrm{I}}(\mathcal{N}_k)$. Thus, $\mathit{FLInf}(\mathcal{N}_k{\downarrow_{\mathrm{active}}}) \subseteq \mathit{LRed}^{\flat}_{\mathrm{I}}(\mathcal{N} \cup \mathcal{M})$.

Let $\iota \in \mathit{FLInf}(\mathcal{N}{\downarrow_{\mathrm{active}}})$. By the argument above, we have $\iota \in \mathit{LRed}^{\flat}_{\mathrm{I}}(\mathcal{N} \cup \mathcal{M})$. We must show $\iota \in \mathit{LRed}^{\flat}_{\mathrm{I}}(\mathcal{N} \cup \mathcal{M}')$. By definition of $\mathit{LRed}^{\flat}_{\mathrm{I}}$, it suffices to show that for every ground instance $(C_n, \ldots, C_1, C_0)$ of ι, there exists a finite clause set $\mathcal{D} \subseteq \mathcal{G}(\mathcal{N}) \cup \mathcal{G}(\mathcal{M}')$ that is $\prec$-smaller than C_1 and such that $\{C_n, \ldots, C_2\} \cup \mathcal{D} \models^{\flat} \{C_0\}$. Without loss of generality, we assume that $\mathcal{D}$ is the $\prec$-smallest such set; such a set exists because $\prec$ is well founded.

By definition of $\mathcal{N}$, p cannot occur in C_0. By Lemma 4.18, if p occurred in $D \in \mathcal{D}$, we could remove D, but this would mean $\mathcal{D}$ is not minimal. As a result, $\mathcal{D}$ cannot contain clauses from $\mathcal{G}(\mathcal{M})$ and hence $\mathcal{D} \subseteq \mathcal{G}(\mathcal{N})$. Thus, $\iota \in \mathit{LRed}^{\flat}_{\mathrm{I}}(\mathcal{N})$. By property (R2) of redundancy criteria, we have the desired result: $\iota \in \mathit{LRed}^{\flat}_{\mathrm{I}}(\mathcal{N} \cup \mathcal{M}')$. □

Lemma 4.21. *Let $(\mathcal{N}_i)_i$ be a $\triangleright_{LRed^{\flat,\sqsupset}}$-derivation. If $Inv^{\flat}_{\mathcal{N}}(i)$ holds for all indices i, then $FLInf(\mathcal{N}_\infty{\downarrow_{\text{active}}}) \subseteq \bigcup_i LRed^{\flat}_{\text{I}}(\mathcal{N}_i)$ holds.*

Proof. We assume $\iota \in FLInf(\mathcal{N}_\infty{\downarrow_{\text{active}}})$ and show $\iota \in \bigcup_i LRed^{\flat}_{\text{I}}(\mathcal{N}_i)$ for some arbitrary ι. For ι to belong to $FLInf(\mathcal{N}_\infty{\downarrow_{\text{active}}})$, all of its finitely many premises must be in $\mathcal{N}_\infty{\downarrow_{\text{active}}}$. Therefore, there must exist an index k such that $\mathcal{N}_k{\downarrow_{\text{active}}}$ contains all of them, and therefore $\iota \in FLInf(\mathcal{N}_k{\downarrow_{\text{active}}})$. Since $Inv_{\mathcal{N}}(k)$ holds, $\iota \in LRed^{\flat}_{\text{I}}(\mathcal{N}_k)$. Hence, $\iota \in \bigcup_i LRed^{\flat}_{\text{I}}(\mathcal{N}_i)$. □

Lemma 4.22. *Let $(\mathcal{N}_i)_i$ be a $\Longrightarrow_{\text{GCP}}$-derivation. If $\mathcal{N}_0{\downarrow_{\text{active}}} = \emptyset$ and $\mathcal{N}_\infty{\downarrow_l} = \emptyset$ for every label $l \neq$ active, then there exists an index k such that $(\mathcal{N}_{k+i})_i$ is a fair $\triangleright_{LRed^{\flat,\sqsupset}}$-derivation.*

Proof. By Lemma 4.19, there must exist an index k such that the sequence $(\mathcal{N}_{k+i})_i$ is a pure $\triangleright_{LRed^{\flat,\sqsupset}}$-derivation. By Lemma 4.20, $Inv^{\flat}_{\mathcal{N}}(k+i)$ holds for all indices i. Hence, by Lemma 4.21, $FLInf(\mathcal{N}_\infty{\downarrow_{\text{active}}}) \subseteq \bigcup_i LRed^{\flat}_{\text{I}}(\mathcal{N}_{k+i})$. By the second hypothesis, this inclusion simplifies to $FLInf(\mathcal{N}_\infty) \subseteq \bigcup_i LRed^{\flat}_{\text{I}}(\mathcal{N}_{k+i})$. □

Theorem 4.23. *Let $(\mathcal{N}_i)_i$ be a $\Longrightarrow_{\text{GCP}}$-derivation with $\mathcal{N}_0{\downarrow_{\text{active}}} = \emptyset$ and $\mathcal{N}_\infty{\downarrow_l} = \emptyset$ for every label $l \neq$ active. If $\lfloor \mathcal{N}_0 \rfloor$ is unsatisfiable, then some $\mathcal{N}_i$ contains the empty clause with some arbitrary label.*

Proof. By Lemma 4.22, we know that there exists an index k such that $(\mathcal{N}_{k+i})_i$ is a fair $\triangleright_{LRed^{\flat,\sqsupset}}$-derivation. Moreover, since $\triangleright_{LRed^{\flat,\sqsupset}}$ and $\triangleright_{\text{P}}$ preserve unsatisfiability (by (R1) of redundancy criteria, K&K Theorem 1, and our Theorem 4.13), we have that $\lfloor \mathcal{N}_k \rfloor$ is unsatisfiable. Since the base calculus $(FLInf, LRed)$ is assumed to be statically refutationally complete with respect to $\models$, the calculus $(FLInf, LRed^{\flat})$ with a weaker redundancy criterion is also statically complete with respect to $\models$, and by the saturation framework, $(FLInf, LRed^{\flat,\sqsupset})$ preserves this. Exploiting the equivalence of static and dynamic completeness (Lemmas 10 and 11 in Waldmann et al. [51]), we conclude that some $\mathcal{N}_{k+i}$ must contain a labeled empty clause. □

To summarize, we have shown that predicate elimination can be used as inprocessing in a superposition prover based on the given clause procedure, with one minor restriction: To avoid loops, predicate symbols arising from clause splitting should not be eliminated.

5 SATISFIABILITY BY CLAUSE ELIMINATION

The main approaches to show satisfiability of a first-order problem are to produce either a finite Herbrand model or a saturated clause set. Saturations rarely occur except for very small problems or within decidable fragments. In this section, we explore an alternative approach that establishes satisfiability by iteratively removing clauses while preserving unsatisfiability, until the clause set has been transformed into the empty set. So far, this technique has been studied only for QBF [26]. We show that BCE can be used for this purpose. It can efficiently solve some problems for which the saturated set would be infinite. However, it can break the refutational completeness of a saturation prover. We conclude with a procedure that transforms a finite Herbrand model into a sequence of clause elimination steps ending in the empty clause set, thereby demonstrating the theoretical power of clause elimination.

Kiesl et al. [32] generalized BCE to first-order logic. Their generalization uses flat L-resolvents, an extension of flat resolvents that resolves a single literal L against m literals of the other clause.

Definition 5.1. Let $C = L \vee C'$ and $D = L_1 \vee \cdots \vee L_m \vee D'$, where (1) $m \geq 1$, (2) the literals L_i are of opposite polarity to L, (3) L's atom is $\text{p}(\vec{s}_n)$, (4) L_i's atom is $\text{p}(\vec{t}_i)$ for each i, and (5) C and D have no variables in common. The clause $\left(\bigvee_{i=1}^{m} \bigvee_{j=1}^{n} s_j \not\approx t_{ij}\right) \vee C' \vee D'$ is a *flat L-resolvent* of C and D.

Definition 5.2. Let $C = L \vee C'$ be a clause and N be a clause set. Let N' consist of all clauses from $N \backslash \{C\}$ with their variables renamed so that they share no variables with C. The clause C is (*equality-*)*blocked* by L in the set N if all flat L-resolvents between C and clauses in N' are tautologies.

Removing a blocked clause from a set preserves unsatisfiability [32]. Kiesl et al. evaluated the effect of removing all blocked clauses as a preprocessing step and found that it increases the prover's success rate.

In fact, there exist satisfiable problems that cannot be saturated in finitely many steps regardless of the calculus's parameters but that can be reduced to an empty, vacuously satisfiable problem through BCE.

Example 5.3. Consider the clause set N consisting of $C = \mathrm{p}(x, x)$ and $D = \neg\mathrm{p}(y_1, y_3) \vee \mathrm{p}(y_1, y_2) \vee \mathrm{p}(y_2, y_3)$. Note that if no literal is selected, all literals are eligible for superposition. In particular, the superposition of $\mathrm{p}(x, x)$ into D's negative literal eventually needs to be performed regardless of the chosen selection function or term order, with the conclusion $E_1 = \mathrm{p}(z_1, z_2) \vee \mathrm{p}(z_2, z_1)$. Then, superposition of E_1 into D yields $E_2 = \mathrm{p}(z_1, z_2) \vee \mathrm{p}(z_2, z_3) \vee \mathrm{p}(z_3, z_1)$. Repeating this process yields infinitely many clauses $E_i = \mathrm{p}(z_1, z_2) \vee \cdots \vee \mathrm{p}(z_i, z_{i+1}) \vee \mathrm{p}(z_{i+1}, z_1)$ that cannot be eliminated using standard redundancy-based techniques.

In the example above, the clause D is blocked by its second or third literal. If we delete D, C becomes blocked in turn. Deleting C leaves us with the empty set, which is vacuously satisfiable. The example suggests that using BCE during saturation might help focus the proof search. Indeed, Kiesl et al. ended their investigations by asking whether BCE can be used as an inprocessing technique in a saturation prover. Unfortunately, in general the answer is no:

Example 5.4. Consider the unsatisfiable set $N = \{C_1, \ldots, C_6\}$, where

$$C_1 = \neg\mathrm{c} \vee \mathrm{e} \vee \boxed{\neg\mathrm{a}} \qquad C_2 = \neg\mathrm{c} \vee \boxed{\neg\mathrm{e}} \qquad C_3 = \mathrm{b} \vee \boxed{\mathrm{c}}$$
$$C_4 = \neg\mathrm{b} \vee \boxed{\neg\mathrm{c}} \qquad C_5 = \mathrm{a} \vee \boxed{\mathrm{b}} \qquad C_6 = \mathrm{c} \vee \boxed{\neg\mathrm{b}}$$

Assume the simplification ordering $\mathrm{a} \prec \mathrm{b} \prec \mathrm{c} \prec \mathrm{d} \prec \mathrm{e}$ and the selection function that chooses the last negative literal of a clause as presented. Gray boxes indicate literals that can take part in superposition inferences. Only two superposition inferences are possible: from C_3 into C_4, yielding the tautology $C_7 = \mathrm{b} \vee \boxed{\neg\mathrm{b}}$, and from C_5 into C_6, yielding $C_8 = \mathrm{a} \vee \boxed{\mathrm{c}}$. Clause C_7 is clearly redundant, whereas C_8 is blocked by its first literal. If we allow removing blocked clauses, the prover enters a loop: C_8 is repeatedly generated and deleted. Thus, the prover will never generate the empty clause for this unsatisfiable set.

As with hidden tautologies, removing blocked clauses breaks the invariant of the given clause procedure that all inferences between clauses in $\mathcal{A}$ are redundant. To see this, assume the setting of Example 5.4, and let $\mathcal{P} = N$ and $\mathcal{A} = \emptyset$. Assume C_1, C_2, C_3 are moved to the active set. As there are no possible inferences between them, the proof state becomes $\mathcal{A} = \{C_1, C_2, C_3\}$ and $\mathcal{P} = \{C_4, C_5, C_6\}$. After C_4 is moved to $\mathcal{A}$, the conclusion C_7 is computed, but it is not added to $\mathcal{P}$ as it is redundant. Moving C_5 to $\mathcal{A}$ produces no new conclusions, but after C_6 is moved, C_8 is produced. However, if we allow eliminating blocked clauses, it will not be added to $\mathcal{P}$ as it is blocked. The prover then terminates with $\mathcal{A} = N$ and $\mathcal{P} = \emptyset$, even though the original set N is unsatisfiable.

Although using BCE as inprocessing breaks the completeness of superposition in general, it is conceivable that a well-behaved fragment of BCE might exist. This could be investigated further.

Not only can BCE prevent infinite saturation (Example 5.3), but it can also be used to convert a finite Herbrand model into a certificate of clause set satisfiability. The certificate uses only BCE

and addition, in conjunction with a transformation to reduce the clause set to an empty set. This theoretical result explores the relationship between Herbrand models and satisfiability certificates based on clause elimination and addition. It is conceivable that it can form the basis of an efficient way to certify Herbrand models.

In propositional logic, *asymmetric literals* can be added to or removed from clauses, retaining the equivalence of the resulting clause set with the original one. Kiesl and Suda [31] described an extension of this technique to first-order logic. Their definition of asymmetric literals can be relaxed to allow the addition of more literals, but the resulting set is then only equisatisfiable to the original one, not equivalent. This, in turn, allows us to show that a problem is satisfiable by reducing it to an empty problem, as is done in some SAT solvers.

For the rest of this section, we work with clausal first-order logic without equality. We use Herbrand models as canonical representatives of first-order models, recalling that every satisfiable set has a Herbrand model [22, Section 5.4].

Definition 5.5. A literal L is a *global asymmetric literal* (GAL) for a clause C and a clause set N if for every ground instance $C\sigma$ of C, there exists a ground instance $D\varrho \vee L'\varrho$ of $D \vee L' \in N \backslash \{C\}$ such that $D\varrho \subseteq C\sigma$ and $\neg L'\varrho = L\sigma$.

Every asymmetric literal is GAL, but the converse does not hold:

Example 5.6. Consider a clause $C = \mathrm{p}(x, y)$ and a clause set $N = \{\mathrm{q} \vee \mathrm{p}(\mathrm{a}, \mathrm{a})\}$. Then, $\neg\mathrm{q}$ is not an asymmetric literal for C and N, but it is a GAL for C and N.

Adding and removing GALs preserves and reflects satisfiability:

Theorem 5.7. *If L is a GAL for the clause C and the clause set N, then the set $(N \backslash \{C\}) \cup \{C \vee L\}$ is satisfiable if and only if N is satisfiable.*

Proof. Let $N' = N \backslash \{C\} \cup \{C \vee L\}$. The nontrivial direction is to prove that if N' has a model, so does N. If N' has a model, it has an Herbrand model $\mathcal{J}$. Clearly, $\mathcal{J}$ satisfies every clause in N, with the possible exception of C. Assume there exists a ground instance $C\sigma$ falsified by $\mathcal{J}$. Since L is a GAL for C and N, then there exists a ground instance $D\tau \vee L'\tau$ of $D \vee L' \in N'$ and $\neg L'\tau = L\sigma$. If $\mathcal{J} \models L'\tau$, for $C\sigma \vee L\sigma$ to be satisfied by $\mathcal{J}$, some literal in $C\sigma$ must be satisfied by $\mathcal{J}$, contradicting that $C\sigma$ is falsified by $\mathcal{J}$. If $\mathcal{J} \not\models L'\tau$ then some literal in $D\tau$ must be satisfied. Since $D\tau \subseteq C\sigma$, we get the same contradiction. Therefore, $\mathcal{J}$ satisfies N, as needed. □

For first-order logic without equality, a clause $L \vee C$ is blocked if all its L-resolvents are tautologies [32]. The L-resolvent between $L \vee C$ and $\neg L_1 \vee \cdots \vee \neg L_n \vee D$ is $(C \vee D)\sigma$, where σ is the most general unifier of the literals $L, L_1, \ldots, L_n$ [4]. Given a Herbrand model $\mathcal{J}$ of a problem, the following procedure removes all clauses while preserving satisfiability:

(1) Let q be a fresh predicate symbol. For each atom $\mathrm{p}(\vec{s})$ in the Herbrand universe: If $\mathcal{J} \models \mathrm{p}(\vec{s})$, add the clause $\mathrm{q} \vee \mathrm{p}(\vec{s})$; otherwise, add $\mathrm{q} \vee \neg\mathrm{p}(\vec{s})$. Adding either clause preserves satisfiability as both are blocked by q.

(2) Since $\mathcal{J}$ is a model, for each ground instance $C\sigma$, there exists a clause $\mathrm{q} \vee L$ with $L \in C\sigma$. We can transform $C \in N$ into $C \vee \neg\mathrm{q}$, since $\neg\mathrm{q}$ is a GAL for C and N.

(3) Consider the clause $\mathrm{q} \vee L$ added by step 1. Since L is ground and no clause $\mathrm{q} \vee \neg L$ was added (since $\mathcal{J}$ is a model), the only L-resolvents are against clauses added by step 2. Since all of those clauses contain $\neg\mathrm{q}$, the resolvents are tautologies. Thus, each $\mathrm{q} \vee L$ is blocked and can be removed in turn.

(4) The remaining clauses all contain the literal $\neg\mathrm{q}$. They can be removed by BCE as well.

The procedure is limited to first-order logic without equality, since step 3 is justified only if L is a predicate literal. (Otherwise, L cannot block clause $\mathsf{q} \vee L$ [32].) The procedure also terminates only for finite Herbrand models.

Example 5.8. Consider the satisfiable clause set $N = \{\mathsf{r}(x) \vee \mathsf{s}(x), \neg\mathsf{r}(\mathsf{a}), \neg\mathsf{s}(\mathsf{b})\}$ and a Herbrand model $\mathcal{J}$ over $\{\mathsf{a}, \mathsf{b}, \mathsf{r}, \mathsf{s}\}$ such that $\mathsf{r}(\mathsf{b})$ and $\mathsf{s}(\mathsf{a})$ are the only true atoms in $\mathcal{J}$. We show how to remove all clauses in N using $\mathcal{J}$ by following the procedure above.

Let $N_\mathcal{J} = \{\mathsf{q} \vee \neg\mathsf{r}(\mathsf{a}), \mathsf{q} \vee \mathsf{r}(\mathsf{b}), \mathsf{q} \vee \mathsf{s}(\mathsf{a}), \mathsf{q} \vee \neg\mathsf{s}(\mathsf{b})\}$. We set $N \leftarrow N \cup N_\mathcal{J}$. This preserves satisfiability since all clauses in $N_\mathcal{J}$ are blocked. It is easy to check that $\neg\mathsf{q}$ is GAL for every clause in $N \backslash N_\mathcal{J}$. The only substitutions that need to be considered are $\{x \mapsto \mathsf{a}\}$ and $\{x \mapsto \mathsf{b}\}$ for $\mathsf{r}(x) \vee \mathsf{s}(x)$. So we set $N \leftarrow \{\neg\mathsf{q} \vee \mathsf{r}(x) \vee \mathsf{s}(x), \neg\mathsf{q} \vee \neg\mathsf{r}(\mathsf{a}), \neg\mathsf{q} \vee \neg\mathsf{s}(\mathsf{b})\} \cup N_\mathcal{J}$. Clearly, all clauses in $N_\mathcal{J}$ are blocked, so we set $N \leftarrow N \backslash N_\mathcal{J}$. All clauses remaining in N have a literal $\neg\mathsf{q}$ and can be removed, leaving N empty as desired.

6 IMPLEMENTATION

HLBE, predicate elimination, and BCE all admit efficient implementations in a superposition prover. In this section, we describe how to implement the first two sets of techniques. For BCE, we refer to Kiesl et al. [32]. All techniques are implemented in the Zipperposition prover [15]. Zipperposition is designed for fast prototyping of improvements to superposition, but it implements many of the most successful heuristics from the E prover [43] and has recently become competitive [47].

6.1 Hidden-Literal-Based Elimination

For HLBE, an efficient representation of $\mathrm{HL}(L, N)$ is crucial. Because this set may be infinite, we underapproximate it by restricting the length of the transitive chains via a parameter K_len. Given the current clause set N, the finite map $\mathit{Imp}[L']$ associates with each literal L' a set of pairs (L, M) such that $L' \hookrightarrow^k L$, where $k \leq K_\mathrm{len}$ and M is the multiset of clauses used to derive $L' \hookrightarrow^k L$. Moreover, we consider only transitions of type (1) (as per Definition 3.4). The following algorithm maintains Imp dynamically, updating it as the prover derives and deletes clauses. It depends on the global variable Imp and the parameters K_len and K_imp.

procedure AddImplication(L_a, L_c, C)
  if $\mathit{Imp}[L_\mathrm{a}\sigma] \neq \emptyset$ for some renaming σ **then**
    $(L_\mathrm{a}, L_\mathrm{c}) \leftarrow (L_\mathrm{a}\sigma, L_\mathrm{c}\sigma)$
  if there are no L, L', M, σ such that $(L', M) \in \mathit{Imp}[L]$,
    $L\sigma = L_\mathrm{a}$, and $L'\sigma = L_\mathrm{c}$ **then**
    for all (σ, M) such that $(L_\mathrm{c}\sigma, M) \in \mathit{Imp}[L_\mathrm{a}\sigma]$ **do**
      erase all (L', M') such that $M \subseteq M'$ from $\mathit{Imp}[L_\mathrm{a}\sigma]$
    for all L such that $(L', M) \in \mathit{Imp}[L]$
        and $L_\mathrm{a}\sigma = L'$ for some σ **do**
      if $|M| < K_\mathrm{len}$ **then**
        $\mathit{Imp}[L] \leftarrow \mathit{Imp}[L] \cup \{(L_\mathrm{c}\sigma, M \uplus \{C\})\}$
    for all L such that $\mathit{Imp}[L] \neq \emptyset$
        and $L\sigma = L_\mathrm{c}$ for some σ **do**
      $\mathit{Concl} \leftarrow \{(L'\sigma, M \uplus \{C\}) \mid$
          $(L', M) \in \mathit{Imp}[L], |M| < K_\mathrm{len}\}$
      $\mathit{Imp}[L_\mathrm{a}] \leftarrow \mathit{Imp}[L_\mathrm{a}] \cup \mathit{Concl}$
    $\mathit{Congr} \leftarrow \{(s \not\approx t, \{C\}) \mid \exists u.\, L_\mathrm{c} = u[s] \not\approx u[t]\}$
    $\mathit{Imp}[L_\mathrm{a}] \leftarrow \mathit{Imp}[L_\mathrm{a}] \cup \{(L_\mathrm{c}, \{C\})\} \cup \mathit{Congr}$

procedure TrackClause(C)
20 **if** $C = L_1 \vee L_2$ **then**
AddImplication($\neg L_1, L_2, C$)
AddImplication($\neg L_2, L_1, C$)
if $L_2 = \neg L_1\sigma$ for some nonidempotent σ **then**
for all $i \leftarrow 1$ to K_{imp} **do**
25 $L_2 \leftarrow L_2\sigma$
AddImplication($\neg L_1, L_2, C$)

procedure UntrackClause(C)
for all $L_\text{a}, L_\text{c}, M$ such that $(L_\text{c}, M) \in \mathit{Imp}[L_\text{a}]$ **do**
if $C \in M$ **then**
30 erase (L_c, M) from $\mathit{Imp}[L_\text{a}]$

The algorithm views a clause $L \vee L'$ as two implications $\neg L \longrightarrow L'$ and $\neg L' \longrightarrow L$. It stores only one entry for all literals equal up to variable renaming (line 2). Each implication $L_\text{a} \longrightarrow L_\text{c}$ represented by the clause is stored only if its generalization is not present in *Imp* (line 4). Conversely, all instances of the implication are removed (line 6).

Next, the algorithm finds each implication stored in *Imp* that can be linked to $L_\text{a} \longrightarrow L_\text{c}$: Either L_c becomes the new consequent (line 9) or L_a becomes the new antecedent (line 13). If L_c can be decomposed into $u[s] \not\approx u[t]$, rule (3) of Definition 3.4 allows us to store $s \not\approx t$ in $\mathit{Imp}[L_\text{a}]$ (line 18). This is an exception to the idea that transitive chains should use only rule (1). The application of rule (3) does not count toward the bound K_{len}. If L_a is of the form $u[s] \approx u[t]$, then *Imp* could be extended so that $\mathit{Imp}[s \approx t] = \mathit{Imp}[L_\text{a}]$, but this would substantially increase *Imp*'s memory footprint.

In first-order logic, different instances of the same clause can be used along a transitive chain. For example, the clause $C = \neg \mathrm{p}(x) \vee \mathrm{p}(\mathrm{f}(x))$ induces $\mathrm{p}(x) \hookrightarrow^i \mathrm{p}(\mathrm{f}^i(x))$ for all i. The algorithm discovers such self-implications (line 23): For each clause C of the form $\neg L \vee L\sigma$, where σ is some nonidempotent substitution, the pairs $(L\sigma^2, \{C\}), \ldots, (L\sigma^{K_{\text{imp}}+1}, \{C\})$ are added to $\mathit{Imp}[L]$, where K_{imp} is a parameter.

To track and untrack clauses efficiently, we implement the mapping *Imp* as a nonperfect discrimination tree [39]. Given a query literal L, this indexing data structure efficiently finds all literals L' such that for some σ, $L'\sigma = L$ and $\mathit{Imp}[L'] \neq \emptyset$. We can use it to optimize all lookups except the one on line 9. For this remaining lookup, we add an index Imp^{-1} that inverts *Imp*, i.e., $\mathit{Imp}^{-1}[L] = \{L' \mid \mathit{Imp}[L'] = (L, M) \text{ for some } M\}$. To avoid sequentially going through all entries in *Imp* when the prover deletes them, for each clause C we keep track of each literal L such that C appears in $\mathit{Imp}[L]$. Finally, we limit the number of entries stored in $\mathit{Imp}[L]$—by default, up to 48 pairs in each $\mathit{Imp}[L]$ are stored.

To implement the HLE rule, we use $\mathit{Imp}[L]$ as follows: Given a clause $C = L \vee L' \vee C'$, if there are two literals L_1, L_2 and a substitution σ such that $(L_2, M) \in \mathit{Imp}[L_1]$, $C \notin M$, $L_1\sigma = L$, and $L_2\sigma = L'$, we remove L from C. Literal L can also be removed if $L_1\sigma = \neg L'$ and $L_2\sigma = \neg L$. Rule HTR is implemented analogously.

The UnitHTR rule relies on maintaining the index *Unit*, which is built as follows. Whenever the prover derives a unit clause $C = \{L\}$, we find all entries L_a in *Imp* such that L_a and L are unifiable with the most general unifier σ. Then, we set $\mathit{Unit} \leftarrow \mathit{Unit} \cup \{(L_\text{c}\sigma, M \cup \{C\}) \mid (L_\text{c}, M) \in \mathit{Imp}[L_\text{a}]\}$. Given a clause $L \vee C'$ we apply UnitHLE by looking for $(L', M) \in \mathit{Unit}$ such that $L'\sigma = \neg L$, for some substitution σ; we apply UnitHTR by looking for L' such that $L'\sigma = L$. The sets stored together with literals in *Unit* are used for building the proof object and to remove literals from *Unit* once a clause from the given set becomes redundant.

The same data structure is used for supporting FLE and FLR. When (L', M) is added to $Imp[L]$, we check whether $(\neg L', M') \in Imp[L]$ for some M'. If so, $(\neg L, M \cup M')$ is added to *Unit*.

In propositional logic, the conventional approach constructs the *binary implication graph* for the clause set N [28], with edges $(\neg L, L')$ and $(\neg L', L)$ whenever $L \vee L' \in N$. To avoid traversing the graph repeatedly, solvers rely on timestamps to discover connections between literals. This relies on syntactic literal comparisons, which is very fast in propositional logic but not in first-order logic, because of substitutions and congruence.

6.2 Predicate Elimination

To implement portfolio predicate elimination, we maintain a record for each predicate symbol p occurring in the problem with the following fields: set of definition clauses for p, set of nondefinition clauses in which p occurs once, and set of clauses in which p occurs more than once. These records are kept in a priority queue, prioritized by properties such as presence of definition sets and number of estimated resolutions. If p is the highest-priority symbol that is eligible for SPE or DPE, we eliminate it by removing all the clauses stored in p's record from the proof state and by adding flat resolvents to the passive set. Eliminating a symbol might make another symbol eligible.

As an optimization, predicate elimination keeps track only of symbols that appear at most K_{occ} times in the clause set. For inprocessing, we use signals that the prover emits whenever a clause is added to or removed from the proof state and update the records. At the beginning of the 1st, $(K_{\text{iter}} + 1)$st, $(2K_{\text{iter}} + 1)$st, ... iteration of the given clause procedure's loop body, predicate elimination is systematically applied to the entire proof state. The first application of inprocessing amounts to preprocessing. By default, $K_{\text{occ}} = 512$ and $K_{\text{iter}} = 10$. The same ideas and limits apply for BCE.

The most important novel aspect of our predicate elimination implementation recognizes the definition clauses for a symbol p in a clause set N. It is performed as follows:

(1) Let $G = \{C \mid C = (\neg)\mathrm{p}(\vec{x}) \vee C', C \in N, \vec{x}$ are distinct, and the variables of C' are all among $\vec{x}\}$. If G is empty, report failure; otherwise, continue.

(2) Rename all clauses in G so that their only variables are $\vec{x}$.

(3) Let $\lfloor A \rfloor$ be an injective function that assigns a propositional variable to each atom A. This function is lifted to literals by assigning $\lfloor \neg A \rfloor = \neg x$, if $\lfloor a \rfloor = x$, and to clauses elementwise. Furthermore, let $E = \{\lfloor C' \rfloor \mid (\neg)\mathrm{p}(\vec{x}) \vee C' \in G\}$. If E is satisfiable, report failure. Otherwise, let E' be an unsatisfiable core of E and G' be the set of corresponding first-order clauses, and continue.

(4) If all resolvents in $G'_{\mathrm{p}} \bowtie_{\mathrm{p}} G'_{\neg \mathrm{p}}$ are tautologies, then G' is a definition set for symbol p. Otherwise, report failure.

The invalidity of set E in step 3 is checked using a SAT solver, which is already integrated in Zipperposition. As modern theorem provers (including E and Vampire) also use SAT solvers, the method can easily be implemented.

During experimentation, we noticed that recognizing definitions of symbols that occur in the conjecture often harms performance. Thus, Zipperposition recognizes definitions only for the remaining symbols.

In contrast to SAT practice, we do not combine predicate elimination and subsumption when predicate elimination is used as preprocessing.

	Tracked clauses			
	250	500	1000	∞
Active	−14	−16	−8	−12
Passive	+7	+10	+5	−35
Both	**+12**	+10	+7	−45

Fig. 1. Impact of the number and kinds of tracked clauses on HLBE performance, when only predicate literals are tracked.

	Tracked clauses			
	250	500	1000	∞
Active	−10	−14	−8	−18
Passive	−5	−5	−14	−71
Both	**+2**	−1	−8	−79

Fig. 2. Impact of the number and kinds of tracked clauses on HLBE performance, when all literals are tracked.

7 EVALUATION

We measure the impact of our elimination techniques for various values of their parameters. As a baseline, we use Zipperposition's first-order portfolio mode, which runs the prover in 13 configurations of heuristic parameters in consecutive time slices. None of these configurations use our new techniques. To evaluate a given parameter value, we fix it across all 13 configurations and compare the results with the baseline.

The benchmark set consists of all 13 495 CNF and FOF TPTP 7.3.0 theorems [46]. The experiments were carried out on StarExec servers [44] equipped with Intel Xeon E5-2609 CPUs clocked at 2.40 GHz. The portfolio mode uses a single CPU core with a CPU time limit of 180 s. The base configuration solves 7897 problems. The values in the tables indicate the number of problems solved minus 7897. Thus, positive numbers indicate gains over the baseline. The best result is shown in bold.

7.1 Hidden-Literal-Based Elimination

The first experiments use all implemented HLBE rules. To avoid overburdening Zipperposition, we can enable an option to limit the number of tracked clauses for hidden literals. Once the limit has been reached, any request for tracking a clause will be rejected until a tracked clause is deleted. We can choose which kind of clauses are tracked: only clauses from the active set $\mathcal{A}$, only clauses from the passive set $\mathcal{P}$, or both. We also vary the maximal implication chain length K_{len} and the number of computed self-implications K_{imp}.

In Zipperposition, every lookup for instances or generalizations of $s \approx t$ must be done once for each orientation of the equation. To avoid this inefficiency, and also because the implementation of hidden literals does not fully exploit congruence, we can disable tracking clauses with at least one functional literal. Clauses containing functional literals can then still be simplified.

Figures 1 and 2 show the results, without and with functional literal tracking enabled, for $K_{\text{len}} = 2$ and $K_{\text{imp}} = 0$. The columns specify different limits on the number of tracked clauses, with ∞ denoting that no limit is imposed. The rows represent different kinds of tracked clauses. The results suggest that tracking functional literals is not worth the effort but that tracking predicate literals is. The best improvement is observed when both active and passive clauses are tracked. Normally DISCOUNT-loop provers [1], such as Zipperposition do not simplify active clauses using passive clauses, but here we see that this can be effective. Figure 3 shows the impact of varying

	Chain length K_{len}			
	1	2	4	8
$K_{\text{imp}} = 0$	+9	+10	+7	+5
$K_{\text{imp}} = 1$	+5	**+11**	+7	+4
$K_{\text{imp}} = 2$	+6	**+11**	+8	+8

Fig. 3. Impact of the parameters K_{len} and K_{imp} on HLBE performance.

		Relaxed with K_{tol}				
	K&K	0	25	50	100	200
SPE preprocessing	+70	+117	+154	+160	+154	+158
PPE preprocessing	+71	+124	+160	+164	**+165**	+162

Fig. 4. Impact of the choice of criterion on predicate elimination performance.

K_{len} and K_{imp}, when 500 clauses from the entire proof state are tracked. These results suggest that computing long implication chains is counterproductive.

7.2 Predicate and Blocked Clause Elimination

For defined predicate elimination, the number of resolvents grows exponentially with the number of occurrences of p. To avoid this expensive computation, we limit the applicability of PPE to proof states for which p is singular. According to our informal experiments, full PPE, without this restriction, generally performs less well.

Predicate elimination can be done using the K&K criterion or using our relaxed criterion with different values of K_{tol}. Figure 4 shows the results for SPE and PPE used as preprocessors. Our numbers corroborate K&K findings: SPE with K&K proves 70 more problems than the base, a 0.9% increase, comparable to the 1.8% they observe when they combine SPE with additional preprocessing. Remarkably, the number of additional proved problems more than doubles when we use our criterion with $K_{\text{tol}} > 0$, for both SPE and PPE.

Although this is not evident in Figure 4, varying K_{tol} substantially changes the set of problems solved. For example, when $K_{\text{tol}} = 0$, SPE proves 60 theorems not proved using $K_{\text{tol}} = 50$. The effect weakens as K_{tol} grows. When $K_{\text{tol}} = 100$, SPE proves only 13 problems not found when $K_{\text{tol}} = 200$. Similarly, the set of problems proved by SPE and PPE differs: When $K_{\text{tol}} = 25$, 14 problems are proved by PPE but missed by SPE. Recognizing definition sets is useful: PPE outperforms SPE regardless of the criterion.

Performing BCE and variable elimination until fixpoint increases the performance of SAT solvers [29]. We can check whether the same holds for superposition provers. In this experiment, we use the relaxed criterion with $K_{\text{tol}} = 25$ and HLBE, which tracks up to 500 clauses from any clause set, $K_{\text{len}} = 2$, and $K_{\text{imp}} = 0$. We use each technique as preprocessing and inprocessing.

The results are summarized in Figure 5, where the + sign denotes the combination of techniques. We confirm the results obtained by Kiesl et al. about the performance of BCE as preprocessing: It helps prove 30 more problems from our benchmark set, increasing the success rate by roughly 0.4%. The same percentage increase was obtained Kiesl et al. Using BCE as inprocessing, however, hurts performance, presumably because of its incompatibility with the redundancy criterion.

For preprocessing, the combinations SPE+BCE and PPE+BCE performed roughly on a par with SPE and PPE, respectively. This stands in contrast to the situation with SAT solvers, where such a combination usually helps. It is also worth noting that the inprocessing techniques never

	BCE	SPE	SPE +BCE	PPE	PPE +BCE	HLBE +PPE +BCE
Preprocessing	+30	+154	+159	+160	**+166**	+162
Inprocessing	−48	+140	+127	+146	+131	+127

Fig. 5. Performance of predicate elimination and BCE.

	BCE	SPE	SPE +BCE	PPE	PPE +BCE	HLBE +PPE +BCE
Preprocessing	+29	+46	**+60**	+47	+59	+55

Fig. 6. Performance of predicate elimination and BCE for establishing satisfiability.

outperform their preprocessing counterparts. The last column shows that combining HLBE with other elimination techniques overburdens the prover.

7.3 Satisfiability by Blocked Clause Elimination

Kiesl et al. found that BCE is especially effective on satisfiable problems. To corroborate their results and ascertain whether a combination of predicate elimination and BCE increases the success rate, we evaluate BCE on all 2 273 satisfiable or TPTP FOF and CNF problems. The hardware and CPU time limits are the same as in the experiments above. Figure 6 presents the results.

The baseline establishes the satisfiability of 856 problems. We consider only preprocessing techniques, since BCE compromises refutational completeness—a saturation does not guarantee that the original problem was satisfiable. We note that recognizing definition sets makes almost no difference on satisfiable problems. The sets of problems solved by BCE and PPE differ—30 problems are solved by BCE and not by PPE.

8 DISCUSSION AND RELATED WORK

We briefly surveyed related work in Section 1. In this section, we give a more detailed overview and further discuss connections with the related work.

The research presented in this article is two-pronged. For SAT elimination techniques already generalized to preprocess first-order problems, we looked for ways to interleave them with the given clause procedure of a superposition prover, as inprocessing. For techniques that had not yet been ported to first-order logic, we looked for generalizations that allow both preprocessing and inprocessing.

HTE was first described by Heule et al. [27]. Better implementation that also supports HLE was later described by the same group of authors [28]. We generalized the underlying theoretical concepts to first-order logic, and provided an efficient way to deal with the infinite number of hidden literals that arise with this generalization. More efficient graph-based techniques are yet to be explored.

Variable elimination, based on Davis–Putnam resolution [16], has been studied in the context of both propositional logic [14, 45] and QBF [8]. It was generalized to first-order logic (as a preprocessor) by Khasidashvili and Korovin [30] yielding a technique called predicate elimination. An improvement of variable elimination that uses formula definition information has been popularized as a preprocessing and inprocessing technique for CDCL solvers by Eén and Biere [19]. We generalized this improvement to first-order logic and combined it with the K&K approach. With

reasonable restrictions, this extension can be used as an inprocessing technique. In SAT and QBF, it was observed that allowing variable elimination to slightly increase in the clause set size improves performance [9]. We implemented a similar approach, achieving the double the number of additional proofs found compared to more restrictive approaches.

BCE is used in both SAT [29] and QBF solvers [9]. Its generalization to first-order logic [32] has showed positive effects when used as a preprocessor. We showed that blocked clauses cannot be removed during saturation, but that they can be effectively used to show satisfiability of the clause set. A combination of BCE and variable elimination performs well in propositional logic [29], but we observed no comparable improvement when their generalizations are combined.

Our general approach is one of many ways to combine ideas from SAT solving and first-order proving. Other noteworthy architectures that either incorporate a SAT solver or that generalize the CDCL calculus include DPLL(T) with quantifier instantiation [5, 18, 40], DPLL($\Gamma + T$) [11], labeled splitting [20], AVATAR [48], MCSAT [17], CDSAT [10], SGGS [12], and SCL [21].

9 CONCLUSION

We adapted several preprocessing and inprocessing elimination techniques implemented in modern SAT solvers so that they work in a superposition prover. This involved lifting the techniques to first-order logic with equality but also tailoring them to work in tandem with superposition and its redundancy criterion. Although SAT solvers and superposition provers embody radically different philosophies, we found that the lifted SAT techniques provide valuable optimizations.

We see several avenues for future work. First, the implementation of hidden literals could be extended to exploit equality congruence. Second, although BCE is generally incomplete as an inprocessing technique, we hope to achieve refutational completeness for a substantial fragment of it. Third, predicate elimination and BCE, which thrives on the absence of clauses from the proof state, could be enhanced by tagging and ignoring generated clauses that have not yet been used to subsume or simplify untagged clauses. Fourth, predicate elimination and BCE could be extended to work with functional literals. Fifth, more SAT techniques could be adapted, including bounded variable addition [34] and blocked clause addition [33]. Sixth, the techniques we covered could be adapted to work with other first-order calculi, or generalized further to work with higher-order calculi such as combinatory superposition [7] and λ-superposition [6].

ACKNOWLEDGMENTS

We are grateful to the maintainers of StarExec for letting us use their service. Uwe Waldmann participated in the search for a counterexample to completeness of BCE as inprocessing and confirmed that Example 5.4 is correct. He also suggested major simplifications and helped us debug the proofs of the claims about predicate elimination. Anne Baanen helped us define the nonclassical logic used to disallow splitting. Ahmed Bhayat, Armin Biere, Mathias Fleury, Wan Fokkink, Benjamin Kiesl, and the anonymous reviewers made some useful comments on our manuscript, and Mark Summerfield suggested many textual improvements. We thank them all.

REFERENCES

[1] Jürgen Avenhaus, Jörg Denzinger, and Matthias Fuchs. 1995. DISCOUNT: A system for distributed equational deduction. In *International Conference on Rewriting Techniques and Applications RTA-95 (LNCS)*, Jieh Hsiang (Ed.), Vol. 914. Springer, 397–402.

[2] Franz Baader and Tobias Nipkow. 1998. *Term Rewriting and All That*. Cambridge University Press.

[3] Leo Bachmair and Harald Ganzinger. 1994. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation* 4, 3 (1994), 217–247.

[4] Leo Bachmair and Harald Ganzinger. 2001. Resolution theorem proving. In *Handbook of Automated Reasoning*. John Alan Robinson and Andrei Voronkov (Eds.). Vol. I. Elsevier and MIT Press, 19–99.

[5] Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds. 2017. Congruence closure with free variables. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS 2017, Part II (LNCS)*. Axel Legay and Tiziana Margaria (Eds.), Vol. 10206. 214–230.

[6] Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, Petar Vukmirović, and Uwe Waldmann. 2021. Superposition with lambdas. *Journal of Automated Reasoning* 65, 7 (2021), 893–940.

[7] Ahmed Bhayat and Giles Reger. 2020. A combinator-based superposition calculus for higher-order logic. In *International Joint Conference on Automated Reasoning IJCAR 2020, Part I (LNCS)*. Nicolas Peltier and Viorica Sofronie-Stokkermans (Eds.), Vol. 12166. Springer, 278–296.

[8] Armin Biere. 2004. Resolve and expand. In *International Conference on Theory and Applications of Satisfiability Testing SAT 2004 (LNCS)*. Holger H. Hoos and David G. Mitchell (Eds.), Vol. 3542. Springer, 59–70.

[9] Armin Biere, Florian Lonsing, and Martina Seidl. 2011. Blocked clause elimination for QBF. In *International Conference on Automated Deduction CADE-23 (LNCS)*. Nikolaj Bjørner and Viorica Sofronie-Stokkermans (Eds.), Vol. 6803. Springer, 101–115.

[10] Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar. 2017. Satisfiability modulo theories and assignments. In *International Conference on Automated Deduction CADE-26 (LNCS)*. Leonardo de Moura (Ed.), Vol. 10395. Springer, 42–59.

[11] Maria Paola Bonacina, Christopher Lynch, and Leonardo de Moura. 2009. On deciding satisfiability by DPLL($\Gamma + T$) and unsound theorem proving. In *International Conference on Automated Deduction CADE-22 (LNCS)*. Renate A. Schmidt (Ed.), Vol. 5663. Springer, 35–50.

[12] Maria Paola Bonacina and David A. Plaisted. 2014. SGGS theorem proving: An exposition. In *PAAR-2014 (EPiC Series in Computing)*, Stephan Schulz, Leonardo de Moura, and Boris Konev (Eds.), Vol. 31. EasyChair, 25–38.

[13] Walter Carnielli, Marcelo E. Coniglio, and João Marcos. 2007. Logics of formal inconsistency. In *Handbook of Philosophical Logic (Handbook of Philosophical Logic)*. Dov M. Gabbay and Franz Guenthner (Eds.), Vol. 14. Springer, 1–93.

[14] Philippe Chatalic and Laurent Simon. 2000. ZRES: The old Davis–Putnam procedure meets ZBDD. In *International Conference on Automated Deduction CADE-18 (LNCS)*. David A. McAllester (Ed.), Vol. 1831. Springer, 449–454.

[15] Simon Cruanes. 2017. Superposition with structural induction. In *International Symposium on Frontiers of Combining Systems FroCoS 2017 (LNCS)*. Clare Dixon and Marcelo Finger (Eds.), Vol. 10483. Springer, 172–188.

[16] Martin Davis and Hilary Putnam. 1960. A computing procedure for quantification theory. *Journal of the ACM* 7, 3 (1960), 201–215.

[17] Leonardo de Moura and Dejan Jovanović. 2013. A model-constructing satisfiability calculus. In *International Workshop on Verification, Model Checking, and Abstract Interpretation VMCAI 2013 (LNCS)*. Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni (Eds.), Vol. 7737. Springer, 1–12.

[18] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2007. Efficient E-matching for SMT solvers. In *International Conference on Automated Deduction CADE-21 (LNCS)*, Frank Pfenning (Ed.), Vol. 4603. Springer, 183–198.

[19] Niklas Eén and Armin Biere. 2005. Effective preprocessing in SAT through variable and clause elimination. In *International Conference on Theory and Applications of Satisfiability Testing SAT 2005 (LNCS)*. Fahiem Bacchus and Toby Walsh (Eds.), Vol. 3569. Springer, 61–75.

[20] Arnaud Fietzke and Christoph Weidenbach. 2009. Labelled splitting. *Annals of Mathematics and Artificial Intelligence* 55, 1–2 (2009), 3–34.

[21] Alberto Fiori and Christoph Weidenbach. 2019. SCL clause learning from simple models. In *International Conference on Automated Deduction CADE-27 (LNCS)*. Pascal Fontaine (Ed.), Vol. 11716. Springer, 233–249.

[22] Melvin Fitting. 1996. *First-Order Logic and Automated Theorem Proving* (2nd ed.). Springer.

[23] Jon William Freeman. 1995. *Improvements to Propositional Satisfiability Search Algorithms*. Ph.D. Dissertation. University of Pennsylvania.

[24] Dov M. Gabbay and Hans Jürgen Ohlbach. 1992. Quantifier elimination in second-order predicate logic. In *International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, Bernhard Nebel, Charles Rich, and William R. Swartout (Eds.). Morgan Kaufmann, 425–435.

[25] Jean H. Gallier. 1987. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Wiley.

[26] Marijn Heule, Martina Seidl, and Armin Biere. 2014. A unified proof system for QBF preprocessing. In *International Joint Conference on Automated Reasoning 2014 (LNCS)*. Stéphane Demri, Deepak Kapur, and Christoph Weidenbach (Eds.), Vol. 8562. Springer, 91–106.

[27] Marijn J. H. Heule, Matti Järvisalo, and Armin Biere. 2010. Clause elimination procedures for CNF formulas. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning LPAR-17 (LNCS)*. Christian G. Fermüller and Andrei Voronkov (Eds.), Vol. 6397. Springer, 357–371.

[28] Marijn J. H. Heule, Matti Järvisalo, and Armin Biere. 2011. Efficient CNF simplification based on binary implication graphs. In *International Conference on Theory and Applications of Satisfiability Testing SAT 2011 (LNCS)*, Karem A. Sakallah and Laurent Simon (Eds.). Vol. 6695. Springer, 201–215.

[29] Matti Järvisalo, Armin Biere, and Marijn Heule. 2010. Blocked clause elimination. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS 2010 (LNCS)*. Javier Esparza and Rupak Majumdar (Eds.), Vol. 6015. Springer, 129–144.

[30] Zurab Khasidashvili and Konstantin Korovin. 2016. Predicate elimination for preprocessing in first-order theorem proving. In *International Conference on Theory and Applications of Satisfiability Testing SAT 2016 (LNCS)*. Nadia Creignou and Daniel Le Berre (Eds.), Vol. 9710. Springer, 361–372.

[31] Benjamin Kiesl and Martin Suda. 2017. A unifying principle for clause elimination in first-order logic. In *International Conference on Automated Deduction CADE-26 (LNCS)*, Leonardo de Moura (Ed.), Vol. 10395. Springer, 274–290.

[32] Benjamin Kiesl, Martin Suda, Martina Seidl, Hans Tompits, and Armin Biere. 2017. Blocked clauses in first-order logic. In *LPAR-21 (EPiC Series in Computing)*. Thomas Eiter and David Sands (Eds.), Vol. 46. EasyChair, 31–48.

[33] Oliver Kullmann. 1999. On a generalization of extended resolution. *Discrete Applied Mathematics* 96–97 (1999), 149–176. https://www.sciencedirect.com/science/article/pii/S0166218X99000372?via%3Dihub.

[34] Norbert Manthey, Marijn Heule, and Armin Biere. 2012. Automated reencoding of Boolean formulas. In *Haifa Verification Conference HVC 2012 (LNCS)*. Armin Biere, Amir Nahir, and Tanja E. J. Vos (Eds.), Vol. 7857. Springer, 102–117.

[35] Joao P. Marques-Silva, Ines Lynce, and Sharad Malik. 2009. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*. Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh (Eds.). Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press, 131–153.

[36] William McCune and Larry Wos. 1997. Otter—The CADE-13 competition incarnations. *Journal of Automated Reasoning* 18, 2 (1997), 211–220.

[37] Andreas Nonnengart and Christoph Weidenbach. 2001. Computing small clause normal forms. In *Handbook of Automated Reasoning*. John Alan Robinson and Andrei Voronkov (Eds.). Vol. I. Elsevier and MIT Press, 335–367.

[38] Hans Jürgen Ohlbach. 1996. SCAN—Elimination of predicate quantifiers. In *International Conference on Automated Deduction CADE-13 (LNCS)*. Michael A. McRobbie and John K. Slaney (Eds.), Vol. 1104. Springer, 161–165.

[39] I. V. Ramakrishnan, R. C. Sekar, and Andrei Voronkov. 2001. Term indexing. In *Handbook of Automated Reasoning*. Vol. II. Elsevier and MIT Press, 1853–1964.

[40] Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. 2018. Revisiting enumerative instantiation. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS 2018, Part II (LNCS)*. Dirk Beyer and Marieke Huisman (Eds.), Vol. 10806. Springer, 112–131.

[41] Alexandre Riazanov and Andrei Voronkov. 2001. Splitting without backtracking. In *International Joint Conference on Artificial Intelligence 2001*. Bernhard Nebel (Ed.). Morgan Kaufmann, 611–617.

[42] Stephan Schulz. 2002. E—A brainiac theorem prover. *AI Communications* 15, 2–3 (2002), 111–126.

[43] Stephan Schulz, Simon Cruanes, and Petar Vukmirović. 2019. Faster, higher, stronger: E 2.3. In *International Conference on Automated Deduction CADE-27 (LNCS)*, Pascal Fontaine (Ed.), Vol. 11716. Springer, 495–507.

[44] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. 2014. StarExec: A cross-community infrastructure for logic solving. In *International Joint Conference on Automated Reasoning (IJCAR'14), (LNCS)*. Stéphane Demri, Deepak Kapur, and Christoph Weidenbach (Eds.), Vol. 8562. Springer, 367–373.

[45] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. 2004. NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In *International Conference on Theory and Applications of Satisfiability Testing SAT 2004 (LNCS)*. Holger H. Hoos and David G. Mitchell (Eds.), Vol. 3542. Springer, 276–291.

[46] Geoff Sutcliffe. 2017. The TPTP problem library and associated infrastructure—from CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning* 59, 4 (2017), 483–502.

[47] G. Sutcliffe. 2020. The CADE-27 automated theorem proving system competition—CASC-27. *AI Communications* 32, 5–6 (2020), 373–389.

[48] Andrei Voronkov. 2014. AVATAR: The architecture for first-order theorem provers. In *International Conference on Computer Aided Verification CAV 2014 (LNCS)*. Armin Biere and Roderick Bloem (Eds.), Vol. 8559. Springer, 696–710.

[49] Petar Vukmirović, Jasmin Blanchette, and Marijn J. H. Heule. 2021. SAT-inspired eliminations for superposition. In *Proceedings of the Formal Methods in Computer-Aided Design FMCAD 2021*. IEEE, 231–240.

[50] Uwe Waldmann, Sophie Tourret, Simon Robillard, and Jasmin Blanchette. 2020. A comprehensive framework for saturation theorem proving. In *International Joint Conference on Automated Reasoning 2020, Part I (LNCS)*. Nicolas Peltier and Viorica Sofronie-Stokkermans (Eds.), Vol. 12166. Springer, 316–334.

[51] Uwe Waldmann, Sophie Tourret, Simon Robillard, and Jasmin Blanchette. A comprehensive framework for saturation theorem proving. *Journal of Automated Reasoning* (accepted). Retrieved from https://matryoshka-project.github.io/pubs/saturate_article.pdf.

Received 28 March 2022; revised 15 July 2022; accepted 22 July 2022