Predicate elimination for preprocessing in first-order theorem proving

Zurab Khasidashvili[1] and Konstantin Korovin[2]

[1] Intel Israel Design Center, Haifa 31015, Israel
zurabk@iil.intel.com
[2] The University of Manchester, UK
korovin@cs.man.ac.uk

Abstract. Preprocessing plays a major role in efficient propositional reasoning but has been less studied in first-order theorem proving. In this paper we propose a predicate elimination procedure which can be used as a preprocessing step in first-order theorem proving and is also applicable for simplifying quantified formulas in a general framework of satisfiability modulo theories (SMT). We describe how this procedure is implemented in a first-order theorem prover iProver and show that many problems in the TPTP library can be simplified using this procedure. We also evaluated our preprocessing on the HWMCC'15 hardware verification benchmarks and show that more than 50% of predicates can be eliminated without increasing the problem sizes.

1 Introduction

Preprocessing techniques for Boolean satisfiability apply rewriting and other simplification rules to a propositional formula in conjunctive normal form (CNF) with the aim of making SAT solving easier [3, 6, 10, 14, 27, 37]. Since the development of the SatElite pre-processing technique [14], the usefulness of pre-processing has become evident and it is now an integrated part of SAT solving. We refer to [20] for an overview of recent advances in CNF pre-processing and in-processing techniques where simplification steps are also applied in interleaving the SAT solving.

Variable elimination [6, 12, 14, 37] is an important simplification technique for CNF and QBF formulas. In this paper we aim at generalizing this approach to first-order logic such that it becomes a predicate elimination technique. Predicate elimination is a special case of second-order quantifier elimination which has been investigated starting from the work of Ackermann [1] and more recently in, e.g., [15, 16]. Second-order quantifier elimination in first-order logic has a number of applications ranging from invariant generation and interpolation in program analysis [19, 25] to correspondence theory in modal logics [33] and uniform interpolation in description logics [22].

In this paper we propose to use predicate elimination as a preprocessing technique for first-order reasoning and show that it is also applicable for simplifying quantified formulas in a general framework of satisfiability modulo theories (SMT). The main goal of our algorithm is to eliminate as many predicates as possible without increasing the complexity of the clause set. The second goal is to simplify the original set of clauses as much as possible using newly generated clauses. Let us note that in contrast to variable

elimination in propositional logic, effective predicate elimination is not always possible for first-order formulas with quantifiers. In order obtain a terminating procedure we propose to restrict predicate elimination to a specific case of non-self-referential predicates as defined in the paper. We show that using non-self-referential predicate elimination one can eliminate predicates from 77% of problems in the TPTP library [38] and considerably reduce the number of predicates and clauses in many cases. In the case of HWMCC'15 benchmarks [9] we show that in total more than 50% of predicates can be eliminated without increasing the problem size.

This paper is organized as follows: In the next section we set the terminology and the framework. In Section 3 we introduce predicate elimination and prove its soundness and correctness as a simplification technique for first-order formulas in clausal normal form. In Section 4 we introduce an algorithm for predicate elimination, called NSR-Pred-Elim. In Section 5 we present experimental results which show that a number of previously unsolved problems in the TPTP library for first-order logic problems can be solved after applying predicate elimination.

2 Preliminaries

In this paper we are mainly focused on theorem proving in first-order logic but our considerations are also applicable in a more general setting of satisfiability of quantified first-order formulas modulo theories (SMT) [5, 28, 29, 31], as briefly introduced below. Let Σ be a first-order signature consisting of predicate and function symbols. Let Σ be split into a theory part $\Sigma_{\mathcal{T}}$, containing interpreted theory symbols and $\Sigma_{\mathcal{F}}$ containing uninterpreted symbols. We assume that the equality symbol $\simeq$ is in $\Sigma_{\mathcal{T}}$ and is interpreted as equality. A theory $\mathcal{T}$ is defined by a non-empty class of first-order interpretations in the signature $\Sigma_{\mathcal{T}}$, closed under isomorphisms. We say that a first-order formula F is satisfiable modulo $\mathcal{T}$ if there exists an interpretation I satisfying F such that the reduction of I to the signature $\Sigma_{\mathcal{T}}$ (forgetting symbols in $\Sigma_{\mathcal{F}}$) is in $\mathcal{T}$. In the rest of the paper when we say satisfiability we assume satisfiability modulo some arbitrary but fixed theory $\mathcal{T}$. If $\Sigma_{\mathcal{T}}$ is empty then we have usual first-order logic without equality.

Satisfiability of quantified first-order formulas (modulo theories) can be reduced to satisfiability of sets of universally quantified first-order clauses using Skolemization and CNF transformation [18, 30]. In the rest of the paper we will consider the satisfiability of sets of first-order clauses and assume that all variables in clauses are implicitly universally quantified. Given an interpretation I, a *variable assignment* σ is a mapping from variables X into I. We denote by $\sigma_{\bar{x}}$ a restriction of σ onto variables $\bar{x}$. A substitution is a mapping from variables X into the set of terms over X. We will use σ, ρ, γ to denote variable assignments and also substitutions when this does not cause confusion.

3 Predicate elimination

Let S be a set of first-order clauses over the signature Σ. Consider a predicate P of arity n in $\Sigma_{\mathcal{F}}$ which we aim to eliminate from S. A literal which contains P will be called a P-literal. Our predicate elimination procedure will be based on two rules: flattening,

where we abstract all terms from P-literals, and flat resolution, where we resolve the flattened predicates.

A P-literal is flat if it is of the form $P(x_1, \ldots, x_n)$ or $\neg P(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are pairwise distinct variables. A predicate P is flat in a set of clauses S if all occurrences of P-literals in S are flat. We define the *flattening* rule on clauses containing P as follows:

$$\frac{C \vee (\neg)P(t_1, \ldots, t_n)}{C \vee x_1 \not\approx t_1 \vee \ldots \vee x_n \not\approx t_n \vee (\neg)P(x_1, \ldots, x_n)} \text{ (Flattening)}$$

where $x_1, \ldots, x_n$ are fresh variables that do not occur in $C \vee P(t_1, \ldots, t_n)$ and literal $(\neg)P(t_1, \ldots, t_n)$ is not flat. It is clear that flattening is equivalence preserving and therefore we can remove the premise after adding the conclusion of the rule.

We eliminate predicates after flattening using *flat resolution* defined as follows:

$$\frac{C \vee P(x_1, \ldots, x_n) \qquad D \vee \neg P(x_1, \ldots, x_n)}{C \vee D} \text{ (FRes)}$$

We assume that before applying flat resolution, variables in clauses are renamed such that resolved literals are of the form $P(x_1, \ldots, x_n)$ and $\neg P(x_1, \ldots, x_n)$ respectively and all common variables in C and D are in $x_1, \ldots, x_n$.

We call a predicate P *self-referential in a clause* C if the number of occurrences of P in C is greater than 1. A predicate P is *self-referential in a set of clauses* S if P is self-referential in at least one clause in S. A predicate which is not self-referential in a set of clauses S will be called *non-self-referential in* S.

In order to obtain a terminating elimination procedure we restrict ourselves to eliminating non-self-referential predicates. Let S be a set of clauses and let $P \in \Sigma_{\mathcal{F}}$ be a flat and non-self-referential predicate in S. Partition S into three disjoint sets $S_P, S_{\neg P}, S_{\tilde{P}}$ – the set of clauses containing P positively, negatively and not containing P, respectively. Denote by $S_P \bowtie S_{\neg P}$ the set of all resolvents obtained by pairwise flat resolutions between clauses in S_P and $S_{\neg P}$ upon P.

Then, the *non-self-referential predicate elimination transformation* is defined as the following rule on sets of clauses:

$$\frac{S_{\tilde{P}} \cup S_P \cup S_{\neg P}}{S_{\tilde{P}} \cup (S_P \bowtie S_{\neg P})} \text{ (NSR-Pred-Elim)}$$

Let us note that since P is non-self-referential in $S_{\tilde{P}} \cup S_P \cup S_{\neg P}$, P does not occur in the conclusion set of the NSR-Pred-Elim.

The following theorem essentially follows from results in [5, 16]. Here we give a simple proof and would like to emphasise that predicate elimination is a satisfiability preserving transformation in a general setting of satisfiability modulo theories.

Theorem 1. *The non-self-referential predicate elimination transformation is satisfiability preserving.*

Proof. Since flat resolution is a sound rule, satisfiability of the set of clauses in the premise implies satisfiability of the set of clauses in the conclusion. Let us show that

the reverse direction also holds. Let us note that P does not occur in $S_{\tilde{P}} \cup (S_P \bowtie S_{\neg P})$. Assume that $S_{\tilde{P}} \cup (S_P \bowtie S_{\neg P})$ is satisfiable and let I be a model of $S_{\tilde{P}} \cup (S_P \bowtie S_{\neg P})$ in the signature $\Sigma \setminus \{P\}$. We construct an interpretation I_P over the signature Σ which will be an expansion of I (i.e., coincide with I on all predicates other than P) and satisfy all clauses in $S_{\tilde{P}} \cup S_P \cup S_{\neg P}$.

We define the predicate P in I_P by defining the truth value of $P(\bar{x})\sigma$ for each variable assignment σ in I. Consider a variable assignment σ in I. We define $P(\bar{x})\sigma$ to be true in I_P whenever there is a clause $D \vee P(\bar{x}) \in S_P$ such that $D\sigma$ is false in I. Likewise, define $P(\bar{x})\sigma$ to be false in I_P if there is a clause $C \vee \neg P(\bar{x}) \in S_{\neg P}$ such that $C\sigma$ is false in I. In all other cases we define the truth value of $P(\bar{x})\sigma$ arbitrarily. Let us show that this definition is well-defined, i.e., we never assign true to $P(\bar{x})\sigma$ and false to $P(\bar{x})\rho$ under assignments σ and ρ such that $\sigma_{\bar{x}} = \rho_{\bar{x}}$. Assume otherwise. Let σ and ρ be such that $\sigma_{\bar{x}} = \rho_{\bar{x}}$ and for some clauses $D \vee P(\bar{x}) \in S_P$ and $C \vee \neg P(\bar{x}) \in S_{\neg P}$, both $D\sigma$ and $C\rho$ are false in I. Then $D \vee C$ is in $S_P \bowtie S_{\neg P}$ and since all common variables in D and C are in $\bar{x}$ we have $(D\sigma) \vee (C\rho) = (D \vee C)\gamma$ for an assignment γ that coincides with σ on variables of D and with ρ on variables of C. We have $I \models (D \vee C)\gamma$, hence $I \models (D\sigma) \vee (C\rho)$ which contradicts to the assumption that $D\sigma$ and $C\rho$ are false in I. By construction, I_p satisfies all clauses in $S_P \cup S_{\neg P}$ and since I_P is an expansion of I we also have I_P satisfies all clauses in $S_{\tilde{P}}$.

Predicate elimination allows us to eliminate a non-self-referential predicate from a set of clauses by first applying flattening and then NSR-Pred-Elim. After application of NSR-Pred-Elim we can apply the equality substitution rule to eliminate negative equalities which were introduced during flattening. Equality substitution is defined as follows:

$$\frac{C \vee x \not\simeq t}{C[t/x]} \text{ (Eq-Subst)}$$

where x does not occur in t. Equality substitution is an equivalence preserving rule so we can remove the premise when we add the conclusion to a set of clauses.

Let us note that in the case of first-order logic without theories (including equality) we do not need to apply flattening and instead we can use most general unifiers when applying resolution in the elimination. The proof of Theorem 1 carries over by considering Herbrand interpretations rather than arbitrary interpretations. Already in the presence of equality flattening is essential.

Example 1. Consider the following set of unit clauses:

$$P(a), \neg P(b), a \simeq b.$$

This set is unsatisfiable, but we can not eliminate P based on unification. When flattening is applied to P-literals we obtain:

$$P(x_1) \vee x_1 \not\simeq a, \neg P(x_1) \vee x_1 \not\simeq b, a \simeq b.$$

After applying flat resolution and equality substitution we obtain an unsatisfiable set $a \simeq b, a \not\simeq b$ where P is eliminated.

4 Predicate elimination for preprocessing

Let us note that if we start with a finite set of clauses then after eliminating a non-self-referential predicate we obtain a clause set that is at most quadratic in the number of clauses compared to the original set. As we will see in Section 5, in practice this set is usually much smaller after eliminating redundant clauses. Let us also note that after eliminating a non-self-referential predicate some previously non-self-referential predicates can become self-referential and conversely some self-referential predicates can become non-self-referential due to removal of clauses during elimination and simplification steps. Thus, the order of elimination can affect which predicates can be eliminated in the process.

Example 2. Consider the following set of clauses S, where predicates P, Q, R are uninterpreted and function symbols f, g can be either interpreted or uninterpreted.

$$\begin{array}{ll} 1. & P(f(u), u) \vee f(f(u)) \simeq g(u) \\ 2. & \neg P(v, g(u)) \vee Q(f(g(u))) \vee \neg Q(v) \\ 3. & Q(v) \vee R(v) \\ 4. & \neg Q(f(v)) \vee \neg R(f(v)) \vee R(f(f(v))) \end{array}$$

In this set of clauses predicate P is non-self-referential while Q and R are self-referential. In order to eliminate P we first flatten P-literals in clauses 1 and 2 and rename variables, obtaining:

$$\begin{array}{ll} 1_{flat}. & P(x_1, x_2) \vee x_1 \not\simeq f(u_1) \vee x_2 \not\simeq u_1 \vee f(f(u_1)) \simeq g(u_1) \\ 2_{flat}. & \neg P(x_1, x_2) \vee x_1 \not\simeq v_2 \vee x_2 \not\simeq g(u_2) \vee Q(f(g(u_2))) \vee \neg Q(v_2) \end{array}$$

After applying flat resolution and repeatedly applying equality substitution we obtain:

$$5.\ f(f(g(u_2))) \simeq g(g(u_2)) \vee Q(f(g(u_2))) \vee \neg Q(f(g(u_2)))$$

Clause 5 is a tautology and therefore we can eliminate P by simply removing clauses 1 and 2. After eliminating P, Q becomes non-self-referential and therefore can also be eliminated. After eliminating Q we obtain the empty set of clauses, hence the original set of clauses is satisfiable.

Our main goal is to use predicate elimination for preprocessing to simplify sets of first-order clauses. For this we consider the non-self-referential predicate elimination (NSR-Pred-Elim), presented below. Clause simplifications play a central role in the NSR-Pred-Elim algorithm. Let us first briefly describe simplifications that were used in our implementation.

Tautology elimination. Clauses of the form $P(\bar{t}) \vee \neg P(\bar{t}) \vee C$ are *tautologies* and can be eliminated. In the presence of equality we also eliminate equational tautologies of the form $t \simeq t \vee C$. We refer to [26] for a more general notion of equational tautologies.

Subsumption. A clause C *subsumes* a clause D if $C\sigma \subseteq D$ for some substitution σ, considering clauses as literal multi-sets. Subsumed clause D can be removed in the presence of clause C.

Subsumption resolution. Subsumption resolution of two clauses can be seen as an application of resolution to these clauses followed by subsumption of one of its premises by the conclusion [4]. In this case, the subsumed premise is replaced by the conclusion.

Global subsumption. A set of clauses S *globally subsumes* a clause D if there is a clause C such that $S \models_{gr} C$ and $C\sigma \subsetneq D$ for a substitution σ, where $\models_{gr}$ is a propositional approximation of $\models$. In this case C is called a *witness for global subsumption* and we can replace subsumed clause D by C. A global subsumption witness C can be obtained from S and D by adjoining negations of subclauses of D to S and applying propositional reasoning. We refer to [24] for details.

Let us note that we are not restricted to using only these simplifications, the method allows one to use any other collection of sound simplifications.

The NSR-Pred-Elim algorithm. The input of the NSR-Pred-Elim (Algorithm 1) is a set of first-order clauses and a predicate elimination queue. The main goal is to eliminate as many predicates as possible from the elimination queue without increasing the complexity of the clause set. The second goal is to simplify the original set of clauses as much as possible using newly generated clauses. There are different possible complexity measures, we use one of the most restrictive and require that after each predicate elimination the number of literals should not increase and in addition that the *normalised variable complexity* of the set of clauses should also not increase. We define normalised variable complexity of a set of clauses as a sum of squares of the number of variables in each clause. The idea behind the last restriction is to give a greater weight to clauses with higher number of variables since reasoning with such clauses is generally more expensive.

During a run of the algorithm we maintain a set $global\text{-}clauses$ which is equisatisfiable to the input set and a set $local\text{-}clauses$ which is generated during elimination of a predicate. Clause simplifications play a central role in the NSR-Pred-Elim algorithm. Each newly generated clause is first *self-simplified* by simplifications such as equality substitution and tautology elimination (line 14). Then the $forward\text{-}simplify$ procedure (line 15) is applied, where the clause is simplified by local clauses and global clauses using simplifications such as subsumption, subsumption resolution and global subsumption. If the clause is eliminated by, e.g., tautology elimination or subsumption then we proceed to the next generated clause, otherwise we apply the $backward\text{-}simplify$ procedure which uses (a simplification of) this clause to simplify both local and global clauses using simplifications such as subsumption and subsumption resolution (lines 17, 18). After processing all clauses in the resolvent set $S_P \bowtie S_{\neg P}$ we check whether to keep the result of the elimination or to discard it based on the complexity of the generated clause set (line 22). Even when we discard the result of the predicate elimination we still benefit from the simplifications of the global set by the clauses generated in the process. For this, in the case when we simplify a clause in the global set we keep $simp\text{-}witnesses$ of this simplification (line 26): the set of clauses which are used to simplify the clause and imply the simplified clause. A simplification witness usually consists of a clause derived by flat resolution and equality substitution during the elimination or by simplifications such as subsumption resolution or global subsumption.

The algorithm maintains a map ($pred\text{-}map$) which maps predicates to sets of clauses where the predicate occurs positively and negatively and a set of suspended predicates ($suspended\text{-}pred\text{-}set$) for which the elimination was suspended either due to self-referential occurrences in clauses (line 32) or to complexity of the clause set after elimination (line 27). After updating the set of global clauses we also update $pred\text{-}map$ and $suspended\text{-}pred\text{-}set$. During this update some predicates from $suspended\text{-}pred\text{-}set$ can be moved back to the elimination queue due to clauses eliminated from the global set, which contain predicates from $suspended\text{-}pred\text{-}set$.

Algorithm 1 NSR-Pred-Elim

```
 1: input: S – input clause set
 2: input: elim-queue – predicate elimination priority queue
 3: output: a simplified set of clauses equi-satisfiable with S
 4: global-clauses ← S
 5: eliminated-pred-set ← ∅
 6: suspended-pred-set ← ∅
 7: pred-map.create(global-clauses)
 8: while elim-queue ≠ ∅ do
 9:     P ← pop(elim-queue)
10:     if non-self-referential(P) then
11:         (S_P, S_¬P) ← pred-map.find(P)
12:         local-clauses ← ∅
13:         for C ∈ S_P ⋈ S_¬P do
14:             self-simplify(C)
15:             C ← forward-simplify(C, local-clauses ∪ global-clauses)
16:             if ¬is-eliminated(C) then
17:                 local-clauses ← backward-simplify(C, local-clauses)
18:                 global-clauses ← backward-simplify(C, global-clauses)
19:                 local-clauses ← local-clauses ∪ {C}
20:             end if
21:         end for
22:         if keep-elim(local-clauses) then
23:             global-clauses ← (global-clauses \ (S_P ∪ S_¬P)) ∪ local-clauses
24:             eliminated-pred-set ← eliminated-pred-set ∪ {P}
25:         else
26:             global-clauses ← global-clauses ∪ simp-witnesses(global-clauses)
27:             suspended-pred-set ← suspended-pred-set ∪ {P}
28:         end if
29:         pred-map.update(global-clauses)
30:         suspended-pred-set.update(global-clauses)
31:     else
32:         suspended-pred-set ← suspended-pred-set ∪ {P}
33:     end if
34: end while
35: return global-clauses
```

5 Implementation and evaluation

We implemented the NSR-Pred-Elim algorithm in iProver[3] – a theorem prover for first-order logic [23, 24]. iProver is a general purpose theorem prover for first-order logic which incorporates SAT solvers at its core, currently MiniSAT [13] and optionally PicoSAT [7]. One of the main challenges in efficient implementation of NSR-Pred-Elim are efficient local and global simplifications. In contrast to propositional subsumption, clause-to-clause first-order subsumption is an NP-complete problem. In order to deal with subsumption and subsumption resolution efficiently we employed the compressed feature vector index (CFVI) which is an extension of the feature vector index proposed in [34]. This allowed us to successfully apply NSR-Pred-Elim to problems containing hundreds of thousands of clauses. One of the important parameters of the algorithm is a criterion deciding when to keep the result of a predicate elimination.

For our experiments we used machines with Intel Xeon L5410 2.33 GHz CPU, 4 cores, 12Gb. Each problem was run on a single core with time limit 300s and memory limit 3.5Gb. We evaluated predicate elimination over first-order problems in FOF and CNF formats over the TPTP-v6.1.0 library. iProver accepts first-order problems in CNF form, for problems in general first-order form we used Vampire's [26] clausifier to transform them into CNF. TPTP-v6.1.0 contains 15897 problems in FOF and CNF formats, 13708 (86%) of problems contain predicates other than equality. The left-hand table

eliminated predicates	problems
$1-10$	4759
$11-100$	3682
$101-1000$	1473
$1001-10000$	607
$10001-100000$	82
$100001-1000000$	14
total	10617

reduction in clauses	problems
$0.01\%-20\%$	8042
$20\%-40\%$	930
$40\%-60\%$	515
$60\%-80\%$	399
$80\%-100\%$	188
total	10074

Fig. 1: The number of TPTP problems with the number of eliminated predicates in the specified range (left) and the specified reduction in the number of clauses (right).

in Figure 1 shows that our predicate elimination procedure is able to eliminate predicates in 10617 problems in TPTP, which is 77% of all first-order problems in TPTP containing non-equality predicates. In particular, from this it follows that at least 77% of problems contain non-self-referential predicates which can be eliminated without increasing the size of the problem.

The right-hand table in Figure 1 shows the percentage of reduction in the number of clauses due to the predicate elimination. Let us note that even when there is no reduction in the number of clauses the set of clauses may change after the predicate elimination process. This is the case when the number of removed clauses due to simplifications is the same as the number of generated clauses due to predicate elimination and the addition of simplification witnesses. Table 1 compares the performance of iProver with

[3] iProver is available at `http://www.cs.man.ac.uk/~korovink/iprover`

		pred elim		w/o pred elim	
Ratings	Total	Solved	%	Solved	%
0.0 - 0.1	3495	3123	89	3110	89
0.1 - 0.2	1744	1512	87	1498	86
0.2 - 0.3	1302	931	71	916	70
0.3 - 0.4	1092	602	55	587	54
0.4 - 0.5	999	512	51	506	51
0.5 - 0.6	830	321	39	294	35
0.6 - 0.7	913	268	29	241	26
0.7 - 0.8	607	141	23	129	21
0.8 - 0.9	1115	109	10	110	10
0.9 - 1.0	993	55	6	44	4
1.0 - 1.0	1892	20	1	18	1
total	14982	7594	51	7453	50

Table 1: iProver performance on TPTP problems with and without predicate elimination

predicate elimination and without. Ratings of problems range from 0 – easy problems to 1 – problems that cannot be solved by any solver so far (including previous versions of iProver). We can see from this table that iProver with predicate elimination considerably outperforms iProver without predicate elimination on problems of all ratings except for one case where it loses one problem. iProver can also be used as a preprocessing tool and combined with other theorem provers. We evaluated the effect of the NSR-Pred-Elim preprocessing on top-of-the-range first-order theorem provers Vampire [26] and E [35], which also have their own advanced preprocessors. In both cases the NSR-Pred-Elim preprocessing considerably increased the number of solved problems over TPTP: 130 in the case of E and 32 in the case of Vampire.

Our second set of experiments focuses on the model checking problems in the Hardware Model Checking Competition, HWMCC15 [9]. In [21], the authors proposed two new model checking algorithms based on an encoding of the model-checking problem into the effectively propositional fragment of first-order logic (EPR). The first one is a bounded model checking [8] algorithm called UCM-BMC1, and the second one is a k-induction algorithm [36] called UCM-k-ind. These algorithms employ an unsat core and model-based (UCM) abstraction refinement scheme inspired by [2, 11, 17], based on approximations of the transition relation by first-order predicates. These algorithms are implemented in iProver and the implementation uses the iProver theorem prover to solve the BMC and induction formulas of UCM-BMC1 and UCM-k-ind.

In these experiments, we apply predicate elimination to the BMC and induction formulas of UCM-BMC1 and UCM-k-ind as part of preprocessing and compare them with UCM-BMC1 and UCM-k-ind where predicate elimination is not performed in iProver. With predicate elimination iProver solves more problems and reaches deeper bounds: on the 547 available problems, the total number of bounds reached with predicate elimination is 24244, without predicate elimination it is 21820. Furthermore, predicate elimination eliminated 27 million predicates from the total of 54 million, and eliminated 57 million clauses from the total of 160 million.

References

1. Ackermann, W.: Untersuchungen über das Eliminationsproblem der matheraatischen Logik. Mathematische Annalen 110, 390–413 (1935)
2. Amla, N., McMillan, K.L.: Combining abstraction refinement and sat-based model checking. In: Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS. pp. 405–419 (2007)
3. Bacchus, F., Winter, J.: Effective preprocessing with hyper-resolution and equality reduction. In: Giunchiglia, E., Tacchella, A. (eds.) Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. LNCS, vol. 2919, pp. 341–355. Springer (2003)
4. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson and Voronkov [32], pp. 19–99
5. Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational theorem proving for hierarchic first-order theories. Appl. Algebra Eng. Commun. Comput. 5, 193–212 (1994)
6. Biere, A.: Resolve and expand. In: SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing (2004)
7. Biere, A.: Picosat essentials. JSAT 4(2-4), 75–97 (2008)
8. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without bdds. In: Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS'99. pp. 193–207 (1999)
9. Biere, A., Heljanko, K.: Hardware model checking competition report. `http://fmv.jku.at/hwmcc15/Biere-HWMCC15-talk.pdf` (2015)
10. Brafman, R.I.: A simplifier for propositional formulas with many binary clauses. IEEE Trans. Systems, Man, and Cybernetics, Part B 34(1), 52–59 (2004)
11. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
12. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM 7(3), 201–215 (1960)
13. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proc. of the 6th International Conference SAT'03. LNCS, vol. 2919, pp. 502–518. Springer (2004)
14. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing, 8th International Conference, SAT. LNCS, vol. 3569, pp. 61–75. Springer (2005)
15. Gabbay, D.M., Schmidt, R.A., Szałas, A.: Second-Order Quantifier Elimination: Foundations, Computational Aspects and Applications, Studies in Logic: Mathematical Logic and Foundations, vol. 12. College Publications (2008)
16. Gabbay, D.M., Ohlbach, H.J.: Quantifier elimination in second-order predicate logic. In: Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). pp. 425–435 (1992)
17. Gupta, A., Ganai, M.K., Yang, Z., Ashar, P.: Iterative abstraction using sat-based BMC with proof analysis. In: International Conference on Computer-Aided Design, ICCAD. pp. 416–423 (2003)
18. Hoder, K., Khasidashvili, Z., Korovin, K., Voronkov, A.: Preprocessing techniques for first-order clausification. In: Cabodi, G., Singh, S. (eds.) Formal Methods in Computer-Aided Design, FMCAD. pp. 44–51. IEEE (2012)
19. Hoder, K., Kovács, L., Voronkov, A.: Interpolation and symbol elimination in Vampire. In: Automated Reasoning, 5th International Joint Conference, IJCAR 2010. pp. 188–195 (2010)
20. Järvisalo, M., Heule, M., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Automated Reasoning - 6th International Joint Conference, IJCAR. LNCS, vol. 7364, pp. 355–370. Springer (2012)

21. Khasidashvili, Z., Korovin, K., Tsarkov, D.: EPR-based k-induction with counterexample guided abstraction refinement. In: Gottlob, G., Sutcliffe, G., Voronkov, A. (eds.) GCAI 2015. Global Conference on Artificial Intelligence. EPiC Series in Computing, vol. 36, pp. 137–150. EasyChair (2015)
22. Koopmann, P., Schmidt, R.A.: Uniform interpolation and forgetting for $\mathcal{ALC}$ ontologies with aboxes. In: Bonet, B., Koenig, S. (eds.) Proc. AAAI-2015. pp. 175–181. AAAI Press (2015)
23. Korovin, K.: iProver - an instantiation-based theorem prover for first-order logic (system description). In: the 4th International Joint Conference on Automated Reasoning. LNCS, vol. 5195, pp. 292–298. Springer (2008)
24. Korovin, K.: Inst-Gen - a modular approach to instantiation-based automated reasoning. In: Voronkov, A., Weidenbach, C. (eds.) Programming Logics. LNCS, vol. 7797, pp. 239–270. Springer (2013)
25. Kovács, L., Voronkov, A.: Interpolation and symbol elimination. In: Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction. pp. 199–213 (2009)
26. Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV. LNCS, vol. 8044, pp. 1–35. Springer (2013)
27. Lynce, I., Silva, J.P.M.: Probing-based preprocessing techniques for propositional satisfiability. In: 15th IEEE International Conference on Tools with Artificial Intelligence ICTAI. p. 105. IEEE Computer Society (2003)
28. de Moura, L.M., Bjørner, N.: Efficient e-matching for SMT solvers. In: Pfenning, F. (ed.) Automated Deduction - CADE-21, 21st International Conference on Automated Deduction. LNCS, vol. 4603, pp. 183–198. Springer (2007)
29. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). J. ACM 53(6), 937–977 (2006)
30. Nonnengart, A., Weidenbach, C.: Computing small clause normal forms. In: Robinson and Voronkov [32], pp. 335–367
31. Reynolds, A., Tinelli, C., de Moura, L.M.: Finding conflicting instances of quantified formulas in SMT. In: Formal Methods in Computer-Aided Design, FMCAD. pp. 195–202. IEEE (2014)
32. Robinson, J.A., Voronkov, A. (eds.): Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press (2001)
33. Schmidt, R.A.: The Ackermann approach for modal logic, correspondence theory and second-order reduction. Journal of Applied Logic 10(1), 52–74 (2012)
34. Schulz, S.: Simple and efficient clause subsumption with feature vector indexing. In: Automated Reasoning and Mathematics - Essays in Memory of William W. McCune. pp. 45–67 (2013)
35. Schulz, S.: System description: E 1.8. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8312, pp. 735–743. Springer (2013)
36. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a sat-solver. In: Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000. pp. 108–125 (2000)
37. Subbarayan, S., Pradhan, D.K.: NiVER: non increasing variable elimination resolution for preprocessing SAT instances. In: SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing (2004)
38. Sutcliffe, G.: The TPTP world - infrastructure for automated reasoning. In: Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Revised Selected Papers. pp. 1–12 (2010)