On Incremental Pre-processing for SMT

Nikolaj Bjørner[1](✉) and Katalin Fazekas[2]

[1] Microsoft Research, Redmond, USA
nbjorner@microsoft.com
[2] TU Wien, Vienna, Austria
katalin.fazekas@tuwien.ac.at

Abstract. We introduce a calculus for incremental pre-processing for SMT and instantiate it in the context of z3. It identifies when powerful formula simplifications can be retained when adding new constraints. Use cases that could not be solved in incremental mode can now be solved incrementally thanks to the availability of pre-processing. Our approach admits a class of transformations that preserve satisfiability, but not equivalence. We establish a taxonomy of pre-processing techniques that distinguishes cases where new constraints are modified or constraints previously added have to be replayed. We then justify the soundness of the proposed incremental pre-processing calculus.

1 Introduction

Pre-processing is a central ingredient for scaling automated deduction. These techniques apply targeted global simplification steps that can drastically reduce the complexity of problems before search techniques that use mainly local inference steps are invoked. They are used across several solver domains, spanning SAT, to SMT, first-order automated theorem proving, constraint programming, and integer programming. With the exception of SAT solvers, prior techniques do not combine well when new constraints are added incrementally to a pre-processed state. Solvers have the option to restart pre-processing from scratch. This model is viable if the overall number of solver calls is small compared to time spent solving, but is not practical for scenarios where many minor variations of a set of main constraints are queried. Such scenarios may be found in applications of dynamic symbolic execution or symbolic model checking.

A procedure to incorporate pre- and in-processing techniques [27] into incremental SAT solvers was introduced in [18], where such incremental in-processing allowed a dramatic improvement in the performance of bounded model checking applications. In the case of SAT, the effect of a simplification step is recorded in a *reconstruction stack*. Each eliminated clause is saved on that stack together with a partial assignment, called its *witness*, that is used to show the redundancy of the eliminated clause. For example, the redundancy of blocked clauses are witnessed by their blocked literal, a literal that upon all resolvents are tautological [26, 32]. The reconstruction stack has two very important roles in SAT solvers. First of all, it has all the information that is necessary for model reconstruction [25]. When the elimination of a clause is not model-preserving, its

witness on the stack tells how to modify or extend any found solution of the simplified formula such that it then satisfies the removed clause as well. Beyond that, the reconstruction stack allows to recognize all those previous simplification steps that are potentially invalidated by an incrementally added new constraint. For example, literals that were blocked in the global state of the previous clauses might not be blocked any more in the presence of some new constraints. Finding these clauses and their cone of influence on the reconstruction stack allows to *undo* only the problematic previous simplification steps, thereby allows pre- and in-processing to be incremental [18].

Motivated by incremental in-processing SAT solvers, our goal here is to pave a path towards a similar mechanism in the context of SMT solvers. However, SMT problems extend propositional SAT formulas in several dimensions: the base theory of SMT is the theory of equality over uninterpreted functions and predicates, SMT formulas may contain quantifiers, and constants and functions that have interpretations over theories. Concrete cases of incremental SMT pre-processing was considered in [19]. While most of the formula simplification techniques of SAT solvers are captured by well studied redundancy properties [23], such a unified understanding and description of SMT pre-processing techniques is not yet introduced. Though some redundancy notions of SAT solvers can be directly embedded or generalized to SMT [30], a notion that appears to capture simplifications in SMT in many cases is that of a *substitution*: an uninterpreted constant or function is defined into a solved form and the constraints are simplified based on the solution. When new constraints, containing the solved function symbols, are added after pre-processing, our method distinguishes between simplifications that allow applying the substitution to the new formula or removing the substitution and re-adding the old constraints that were simplified. We have found it useful to characterize pre-processing simplifications by the following categories.

Equivalence Preserving Simplifications Many simplification methods are based on equivalence preserving simplifications. For example $x > x - y + 1$ simplifies to $y > 1$. They are automatically incremental by virtue of not changing the set of models. Developing equivalence preserving simplifications is a significant area of research and engineering by itself. A good example is using and-inverter graphs (AIGs) for simplifying propositional and first-order formulas [45, 24]. The main challenge with developing equivalence preserving simplifications in an incremental setting is to make them efficient.

Rigid Constrained Simplifications An important class of simplifications are based on eliminating variables by finding solutions to them. In the formula $x \leq y + 1 \wedge x \geq y + 1 \wedge \varphi[x, y]$ we can solve for x (or y) by setting $x \simeq y + 1$ and then substituting in the solution for x into φ. The simplified formula is $\varphi[y+1, y]$. The set of models of the original formula must all satisfy the equality $x \simeq y + 1$. This property allows to reuse the simplification when later adding a formula $\psi[x, y]$. It can be added by applying the solution for x: $\psi[y+1, y]$. A model of $\varphi[y+1, y] \wedge \psi[y+1, y]$ must conversely correspond to a model of the original formulas $\varphi[x, y]$ and $\psi[x, y]$. The equality $x \mapsto y + 1$ is used in a *model converter* to establish the

original model. Some pre-processing techniques translate constraints from one domain to another. For example, formulas over bounded integers can be solved by translation into bit-vectors. This translation can be described with a set of equalities where bounded integers are solved for their bit-vector representation (see later an example in Table 1).

Under Constrained Simplifications The rigid constrained simplifications already cover a significant class of pre-processing methods. Allowing incrementally solving for variables has a profound practical effect on using z3 incrementally in user scenarios. There is however a larger class of simplifications that also allow eliminating variables but do not preserve solutions to the eliminated variable. These simplifications have the same or more solutions for symbols in the original formula and we call them *under-constrained*. For example, the formula $((x \simeq y \wedge y < z + u) \vee y \geq z \cdot u)$ contains x in only one position. It can be replaced by the formula $((b \wedge y < z + u) \vee y \geq z \cdot u)$ where b is fresh. Similarly introducing definitions of fresh symbols does not eliminate solutions to symbols in the original formula. Lastly, when removing redundant clauses, the new formula may have more solutions. Tseitin transformation introduces definitions that allow removing redundant, non-CNF, formulas.

Over Constrained Simplifications Symmetry reduction [38, 14] and strengthening using propagation redundancy criteria [37] are prominent examples of simplifications that apply strengthening to reduce the search space. These transformations are not covered by the classes covered by our main result. We leave it to future work to examine whether or how to incorporate strengthening: one avenue is to leverage assumption literals [16] to temporarily enable strengthenings either as part of pre-processing or during search [39].

Table 1 summarizes the main categories of pre-processing techniques discussed so far. This paper develops a calculus of incremental pre-processing for rigid constrained, under-constrained, clause elimination, and introduction of definitions. However, it does not discuss further over-constrained simplifications.

In this paper we introduce the concept of *simplification modulo substitutions* and show that the main SMT pre-processing methods maintain such a property. Based on that, we show how to apply or revert the effect of previous pre-processing steps when new formulas are added after simplification.

2 Preliminaries

We assume the usual notions of first-order logic with equality, satisfiability, logical consequence and theory, as described e.g. in [17]. An interpretation $\mathcal{M}$ for a signature Σ (or Σ-model) consists of a non-empty set $\mathcal{U}_\mathcal{M}$ called the universe of the model, and a mapping $(\_)^\mathcal{M}$ assigning to each variable and constant symbol an element of $\mathcal{U}_\mathcal{M}$, to each n-ary function symbol f in Σ an n-ary function $f^\mathcal{M}$ from $\mathcal{U}_\mathcal{M}^n$ to $\mathcal{U}_\mathcal{M}$, and to each n-ary predicate symbol p in Σ an n-ary function

category	example input	example output	model converter
equivalence	$x > x - y + 1$	$y > 1$	ε
rigid	$x \simeq t, \varphi$	$\varphi[t/x]$	$x \mapsto t$
	$0 \leq x \leq 1 \wedge (x \simeq 1 \vee y > 0)$	$x_b \vee y > 0$	$x \mapsto ite(x_b, 1, 0)$
	$1 \leq x \leq 4 \wedge (x \simeq 1 \vee y > 0)$	$b_{[2]} \simeq 0 \vee y > 0$	$x \mapsto 1 + bv2int(b_{[2]})$
under	$F, ((x \simeq t \wedge \varphi) \vee \psi)$ $x \notin FV(\psi), FV(F)$	$F, (\varphi[t/x] \vee \psi)$	$x \mapsto t$
	$F, x \leq y, x \leq z, y \leq u$ $x, y \notin FV(F)$	F	$x \mapsto \min(y, z), y \mapsto u$
def-intro	$(a \wedge b) \vee c$	$\neg x_b \vee a, \neg x_b \vee b, x_b \vee c$	ε
redundant	$F, \neg p \vee \neg q, p \vee q$ p is positive in F	$F, \neg p \vee \neg q$	$p \mapsto p \vee \neg q$
over	$p(x), p(y), p(z)$	$x \leq y \leq z$ $p(x), p(y), p(z)$	ε

Table 1. Main categories of pre-processing techniques found in SMT solvers. Function *ite* is an abbreviation for *if-then-else* and *bv2int* is a function that maps a bit-vector to an integer value.

from the set $\mathcal{U}^n_{\mathcal{M}}$ to distinguished values representing true and false. Note that to keep the presentation simple, we only consider a single universe in the models. Interpretations extend to terms by composition.

We use the terminology *symbols* referring to uninterpreted symbols (variables) and function symbols. Given a model $\mathcal{M}$ and a symbol x, the model $\mathcal{M}[x \mapsto a]$ is exactly the same as $\mathcal{M}$, except that $x^{\mathcal{M}} = a$ where $a \in \mathcal{U}_{\mathcal{M}}$ for 0-ary symbols and a is a function over $\mathcal{U}_{\mathcal{M}}$ for n-ary function or predicate symbols.

Lemma 1 (Translation Lemma [41]). *If F is a formula and t is a term s.t. no variable in t occurs bound in F, then $\mathcal{M} \models F[t/x]$ iff $\mathcal{M}[x \mapsto t^{\mathcal{M}}] \models F$.*

Note that we may use λ terms to represent updates to function and predicate symbols. The interpretation of a λ term is a function.

We denote *Skolem symbols* for n-ary functions (where $n = 0$ is possible) that cannot occur in input formulas. Only pre-processing methods may introduce the Skolem symbols as a guarantee that they are fresh.

Convention 1 (Variable non-capture) *Throughout this paper we assume that free and bound variables are disjoint, such that when we substitute a term t for a variable x in formula F, none of the variables in t are captured.*

Definition 1 (Labeled substitution). $\langle x \leftarrow t; \Psi \rangle^{\mathbb{B}}$ *represents a substitution of x by t, justified by the formula Ψ. The label $\mathbb{B}$ is either $\top$ or $\bot$ and it indicates whether the map $x \mapsto t$ may be used as an equal replacement of Ψ.*

Example 1. The labeled substitution $\langle x \leftarrow y + 1; x \simeq y + 1 \rangle^{\bot}$ represents the substitution of x by $y + 1$ justified by the formula $x \simeq y + 1$. The label $\bot$ of the substitution indicates that applying the substitution on a formula F where $x \simeq y + 1$ is present does not change the set of models of the formula.

Definition 2. *Given* $\theta = \langle x_1 \leftarrow t_1; \Psi_1 \rangle^{\mathbb{B}_1} \langle x_2 \leftarrow t_2; \Psi_2 \rangle^{\mathbb{B}_2} \ldots \langle x_n \leftarrow t_n; \Psi_n \rangle^{\mathbb{B}_n}$ *and an interpretation* $\mathcal{M}$*, we define the interpretation* $\mathcal{M}\theta$ *as follows:*

$$\mathcal{M}\varepsilon = \mathcal{M}$$
$$\mathcal{M}\theta\langle x \leftarrow t; \Psi \rangle^{\mathbb{B}} = (\mathcal{M}[x \mapsto t^{\mathcal{M}}])\theta$$

Definition 3. *Given* $\theta = \langle x_1 \leftarrow t_1; \Psi_1 \rangle^{\mathbb{B}_1} \langle x_2 \leftarrow t_2; \Psi_2 \rangle^{\mathbb{B}_2} \ldots \langle x_n \leftarrow t_n; \Psi_n \rangle^{\mathbb{B}_n}$ *and a formula* F*, we define the formula* $F\theta$ *as follows:*

$$F\varepsilon = F$$
$$F\langle x \leftarrow t; \Psi \rangle^{\mathbb{B}}\theta = (F[t/x])\theta$$

Informally, a sequence of substitutions θ is applied to interpretations from right to left (i.e. backwards), while to formulas from left to right (i.e. forward). Further, note that the translation lemma generalizes in a straight-forward way to substitutions.

3 Incremental Pre-processing

In this section we introduce a calculus to describe incremental pre-processing for SMT based on the following notion.

Definition 4 (Simplification modulo θ). *We say that the formula* F *simplifies to* F' *modulo* θ*, denoted* $F \succeq_\theta F'$ *if*

- *If* $\mathcal{M} \models F$ *then there is a model* $\mathcal{M}'$ *such that,* $\mathcal{M}' \models F'$ *and* $\mathcal{M}'$ *agrees with* $\mathcal{M}$ *on all symbols that are in* F *or in background theories or not in* F'*.*
- *If* $\mathcal{M}' \models F'$ *then* $\mathcal{M}'\theta \models F$*.*

It follows that simplification allows transitive chaining assuming that symbols are not recycled.

Lemma 2 (Transitivity of simplification). *Let* $F \succeq_\theta F'$ *and* $F' \succeq_{\theta'} F''$ *such that every symbol that is both in* F *and* F'' *also occurs in* F' *(i.e. old symbols are not re-introduced). Then* $F \succeq_{\theta\theta'} F''$*.*

3.1 Simplification rules

There are several possible situations where the concept of simplification modulo substitutions can be used to capture potential simplification steps. For example, a useful special case for simplification modulo θ is when a formula F implies an equality $x \simeq t$ that can then be turned into a substitution to simplify F.

Example 2. The formula $\mathit{isCons}(x) \wedge F[x]$ implies $\exists h, t \,.\, x \simeq \mathit{cons}(h, t)$, where h, t are fresh variables (corresponding to the head and tail of a cons list). We may substitute x by $\mathit{cons}(h, t)$ in $F[x]$ to eliminate x. The literal $\mathit{isCons}(\mathit{cons}(h, t))$ is equivalent true and $F[\mathit{cons}(h, t)]$ is a model simplification of the original formula modulo $x \simeq \mathit{cons}(h, t)$.

There are also useful special cases where a formula F *does not* imply an equality $x \simeq t$, but the same equality may still be used to simplify F.

Example 3. In the formula $F := ((x \simeq 3 \wedge x > u) \vee y > u) \wedge u > z$ we can substitute $x \mapsto 3$ and retain simplification. The formula F simplifies to $F[3/x] := (3 > u \vee y > u) \wedge u > z$, but F does not imply $x = 3$.

There are also cases where substitutions are not suitable to describe the relation between F and F'. It is easier to characterize these by the property that F' is a proper subset of F.

Example 4. A blocked clause $p \vee C$ can be removed from a set of formulas without changing satisfiability: $F, (p \vee C) \succeq_{p \mapsto p \vee \neg C} F$. If we were to substitute p by $p \vee \neg C$ everywhere in F it would weaken clauses where p occurs positively.

Finally, it is possible to accomodate cases where pre-processing *introduces* definitions, such as through the *unfold* transformation (see Section 6.5), or by Skolemization and Tseitin transformations.

Example 5. The Skolemization of $\forall x \,.\, \exists y \,.\, p(x, y)$ is $\forall x \,.\, p(x, f_{sk}(x))$. Here the original quantified formula is replaced by the Skolemized formula.

We model the pre-processing performed by an SMT solver as a sequence of abstract states where each state consists of two components: a formula F and an ordered sequence of labeled substitutions θ. Based on the shown cases, we formulate the following conditions for applying simplification rules in Figure 1.

$$
\begin{array}{lll}
\textsc{Rigid}: & & \\
F \parallel \theta & \Longrightarrow F[t/x] \parallel \theta\langle x \leftarrow t; \Psi\rangle^{\bot} & \text{if } \Psi \subseteq F, x \notin t, \text{ and } \Psi \Rightarrow \exists y \,.\, x \simeq t[y] \\
\textsc{Flex}: & & \\
F, \Psi \parallel \theta & \Longrightarrow F, \Psi[t/x] \parallel \theta\langle x \leftarrow t; \Psi\rangle^{\top} & \text{if } x \in \Psi, x \notin F \text{ and } \Psi \succeq_{x \mapsto t} \Psi[t/x] \\
\textsc{Update}: & & \\
F, \Psi \parallel \theta & \Longrightarrow F, \Phi \parallel \theta\langle x \leftarrow t; \Psi\rangle^{\top} & \text{if } F, \Psi \succeq_{x \mapsto t} F, \Phi
\end{array}
$$

Fig. 1. A calculus for pre-processing in SMT

We formulated the side conditions that allow to identify a minimal set of conjuncts Ψ of F involved with the solution for x. Note that a simplification remains valid when adding conjuncts that do not contain x. The Update rule handles broadly a set of simplifications, including proof rules from DRAT systems and introduction of definitions and Skolemization. It may be presented in forms where Φ or Ψ or the substitution are empty. The substitution $x \mapsto t$ generally represents a tuple of symbols x replaced by terms t. To simplify presentation we only discuss the case where x is a single symbol and we elide rules that preserve equivalence. The Update rule records Ψ so it can later be re-added in case a new constraint mentions x. This may be overkill when $\Phi[t/y] = \Psi$ for y fresh

(in Section 4 we will show another rule, INVERT, that adds only the equality $y \simeq t$ in such cases).

Lemma 3. *If $F \Rightarrow \exists y \,.\, x \simeq t[y]$, s.t. $y \notin F$, $x \notin t$, and t is substitutable for x in F, then $F \succeq_{x \mapsto t} F[t[y]/x]$.*

Proof. Let $\mathcal{M}$ be an interpretation s.t. $\mathcal{M} \models F$. Then $\mathcal{M} \models F \wedge \exists y \,.\, x \simeq t[y]$ and by definition of the satisfaction relation, there must exists an $a \in \mathcal{U}_{\mathcal{M}}$, s.t. $\mathcal{M}[y \mapsto a] \models F \wedge x \simeq t[y]$. Let $\mathcal{M}'$ note $\mathcal{M}[y \mapsto a]$. From $\mathcal{M}' \models F \wedge x \simeq t[y]$ follows that $x^{\mathcal{M}'} = t[y]^{\mathcal{M}'}$ and so $F^{\mathcal{M}'} = F[t[y]/x]^{\mathcal{M}'}$. Since $\mathcal{M}' \models F$, we have that $\mathcal{M}' \models F[t[y]/x]$. For the other direction, when $\mathcal{M}' \models F[t[y]/x]$, due to Lemma 1, $\mathcal{M}'[x \mapsto t[y]^{\mathcal{M}'}] \models F$. Hence, $F \succeq_{x \mapsto t} F[t[y]/x]$. □

Corollary 1. *The side-condition for* RIGID *implies that $F \succeq_{x \mapsto t} F[t/x]$.*

Lemma 4. *Assume $\Psi \subseteq F, x \notin F \setminus \Psi$ and $\Psi \succeq_{x \mapsto t} \Psi[t/x]$, then $F \succeq_{x \mapsto t} F[t/x]$.*

Proof. Since $x \notin F$, $(F \setminus \Psi) = (F \setminus \Psi)[t/x]$, thus $(F \setminus \Psi) \succeq_{x \mapsto t} (F \setminus \Psi)[t/x]$. Then, from $\Psi \succeq_{x \mapsto t} \Psi[t/x]$ follows that $F \succeq_{x \mapsto t} F[t/x]$.

Lemma 3 established that the side-condition for RIGID ensures simplification modulo θ. We therefore have the following corollaries.

Corollary 2. *If a formula F' is derived from F by the inferences from Figure 1, then it has the property $F \succeq_{x \mapsto t} F'$.*

The other rules enforce preservation of satisfiability in their side-conditions.

Corollary 3. *The rules from Figure 1 preserve satisfiability.*

The transitive application of the simplifications also preserve satisfiability in a way that extends the notion of simplification modulo a substitution.

Proposition 1. *Consider a formula F_0 and a state $F \parallel \theta$ derived from $F_0 \parallel \varepsilon$ using the rules from Figure 1. Then $F_0 \succeq_\theta F$.*

Proof. It follows as Corollary 2 notes that each application of a rule from Figure 1 is a simplification modulo and Lemma 2 notes that simplification modulo is transitive.

Informally, Proposition 1 means that using θ, one can transform any model of the simplified formula into a model of the original input formula. Note that the simplified F may contain fresh Skolem symbols that are not occurring in F_0.

3.2 Pre-processing Replay

Rules of Fig. 1 captured possible pre-processing steps that can be applied on a single SMT problem. We now describe the scenario where we add additional constraints Φ to a pre-processed state. Without incremental pre-processing we have the option to conjoin Φ to the original formula F_0 and re-run pre-processing.

The goal of incremental pre-processing is to retain as much of the effect of previous work as possible.

We will show that for pre-processing steps derived by rule RIGID it is possible to apply the corresponding substitution to Φ directly, while the other simplification steps may require to re-introduce formulas that were previously removed. We call this process of applying the effect of simplifications on a new formula as pre-processing *replay*. Figure 2 shows an imperative implementation of pre-processing replay.

```
Replay (formula Φ, substitution sequence θ = σ_1, ... σ_n )
1  θ' := ⟨⟩
2  for ⟨x_i ← t_i; Ψ_i⟩^{B_i}  from σ_1  to σ_n
3      if x_i ∈ FV(Φ)  then
4          if B_i = ⊤  then // substitution is not RIGID
5              Φ := Φ ∪ Ψ_i // re-introduce
6          else
7              Φ := Φ[t_i/x_i] // apply
8              θ' := θ'⟨x_i ← t_i; Ψ_i⟩^{B_i}
9      else
10         θ' := θ'⟨x_i ← t_i; Ψ_i⟩^{B_i}
11 return ⟨Φ, θ'⟩
```

Fig. 2. Algorithm Replay

Our main proposition summarizes the main property of Replay and ensures that an arbitrary formula Φ can be added mid-stream after pre-processing.

Proposition 2. *Let $F \parallel \theta$ be a state resulting from pre-processing F_0, and let $F \wedge \Phi' \parallel \theta'$ be a state produced by applying procedure Replay to Φ and θ, then $F_0 \wedge \Phi$ is equi-satisfiable to $F \wedge \Phi'$.*

To establish Proposition 2 we will introduce a calculus for reverting the effect of simplifications. It is shown in Figure 3 and comprises of two rules, one for adding a formula with a substitution to F, the other both reverts the effect of a simplification and adds the reverted formula to F. The inferences rely on a side-condition that the formulas Φ, Ψ are *clean* relative to the substitution θ.

Definition 5. *A formula Φ is clean w.r.t. a substitution sequence θ iff*

- $\theta = \varepsilon$, *or*
- $\theta = \langle x \leftarrow t; \Psi \rangle^{\mathbb{B}} \theta'$, $x \notin \Phi$ *and Φ is clean with respect to θ', or*
- $\theta = \langle x \leftarrow t; \Psi \rangle^{\bot} \theta'$ *and $\Phi[t/x]$ is clean with respect to θ'.*

Thus, intuitively, Φ is clean w.r.t. θ if $\Phi\theta$ uses only RIGID substitutions from θ.

$$
\begin{array}{lll}
\textsc{Add}: & & \\
F \parallel \theta & \Longrightarrow \quad F, \Phi\theta \parallel \theta & \text{if } \Phi \text{ is clean w.r.t. } \theta \\
\textsc{Undo}: & & \\
F \parallel \theta_0\langle x \leftarrow t; \Psi\rangle^{\mathbb{B}}\theta & \Longrightarrow F, \Psi\theta \parallel \theta_0\theta & \text{if } \Psi \text{ is clean w.r.t. } \theta
\end{array}
$$

Fig. 3. A calculus for reverting pre-processing. Undo reverts a simplification by re-introducing a constraint. It prunes θ until Add applies for a new constraint Φ.

We now establish that formulas that are clean relative to θ can be added (after substitution) to formulas while maintaining models. The substitution used in rigid updates corresponds to equalities that are consequences.

Lemma 5. *Given a state $F' \parallel \theta\theta'$ derived from the state $F \parallel \theta$ and formula Φ that is clean with respect to θ', then $F \wedge \Phi \succeq_{\theta'} F' \wedge \Phi\theta'$.*

Proof. We examine the two directions.

- Let $\mathcal{M} \models F \wedge \Phi$. Induction on the length of the derivation from F to F' establishes that if $\mathcal{M} \models F$, then there is a corresponding $\mathcal{M}'$ such that $\mathcal{M}' \models F' \wedge \bigwedge_{(x \mapsto t) \in \theta'} x \simeq t$: Each time Rigid is applied a new equality is used for simplification $F_1[t_1/x_1]$. The equality can be added to the result, $F_1[t_1/x_1] \wedge x_1 \simeq t_1$ without changing satisfiability because x_1 does not occur in $F_1[t_1/x_1]$. Thus, the resulting model $\mathcal{M}'$ can be constrained to satisfy all equalities used in rigid substitutions. Since $\mathcal{M}' \models \Phi$ already, then $\mathcal{M}' \models \Phi\theta'$.
- Let $\mathcal{M}' \models F' \wedge \Phi\theta'$. Then from the assumption of simplification modulo θ', we get $\mathcal{M}'\theta' \models F$. Lemma 1 ensures $\mathcal{M}'\theta' \models \Phi$. Thus, $\mathcal{M}'\theta' \models F \wedge \Phi$.

The correctness of the Add rule is now immediate:

Corollary 4. *Let $F \parallel \theta$ be derived from $F_0 \parallel \varepsilon$, and Φ clean with respect to θ, then $F_0 \wedge \Phi$ simplifies modulo θ to $F \wedge \Phi\theta$.*

Proof. It follows from Lemma 5.

With Proposition 1 we established that Rigid, Flex and Update maintain $F_0 \succeq_\theta F$. We need to show that also for rule Undo. The first step is to establish that the formula removed by each of the pre-processing rules can be re-added without affecting simplification.

Lemma 6. *Given an inference $F \parallel \theta \Longrightarrow F' \parallel \theta\langle x \leftarrow t; \Psi\rangle^{\mathbb{B}}$ by either of the rules Rigid, Update, Flex the formula F simplifies to F', Ψ modulo ε.*

Proof. The proof is by case analysis by the rule that is applied.

- Flex: Then $F' = F[t/x]$, $\Psi \subseteq F$ and therefore $F' \wedge \Psi = F \wedge \Psi[t/x]$. From the side condition $\Psi \succeq_{x \mapsto t} \Psi[t/x]$ every model of F there is a model of $\Psi[t/x]$ that agrees with symbols from F. Conversely F', Ψ properly contains F and therefore implies it. Therefore, $F \succeq_\varepsilon F', \Psi$.

– UPDATE: We want to show that F, Ψ simplifies to F, Ψ, Φ modulo ε. The premise of UPDATE ensures that for every $\mathcal{M} \models F, \Psi$ there is a model agreeing with $\mathcal{M}$ on symbols in F, Ψ, that satisfies F, Φ. Since interpretation of the symbols in Ψ is unchanged it also satisfies Ψ. Conversely, if $\mathcal{M}' \models F, \Psi, \Phi$, then already $\mathcal{M}' \models F, \Psi$ and therefore $\mathcal{M}'\varepsilon \models F, \Psi$.
– RIGID: We wish to establish that $F \succeq_\varepsilon F', \Psi$. First observe that $F', \Psi = F, \Psi[t/x]$. Since Ψ implies the equation $\exists y \,.\, x \simeq t$, every model of F implies there is a solution to y such that $\Psi[t/x]$ that agrees with the variables in F. Conversely, if $F, \Psi[t/x]$ is satisfied by $\mathcal{M}'$, then $\mathcal{M}'$ already satisfies F.

Lemma 7. *Given* $F \parallel \theta\langle x \leftarrow t; \Psi\rangle^{\mathbb{B}}\theta' \Longrightarrow^{\text{UNDO}} F, \Psi\theta' \parallel \theta\theta'$*, s.t.* $F_0 \succeq_{\theta\langle x \leftarrow t; \Psi\rangle^{\mathbb{B}}\theta'} F$*, then* $F_0 \succeq_{\theta\theta'} F, \Psi\theta'$ *holds.*

Proof. Given an inference $F_1 \parallel \theta \Longrightarrow F_2 \parallel \theta\langle x \leftarrow t; \Psi\rangle^{\mathbb{B}}$. Lemma 6 establishes that the formula F_1 simplifies to F_2, Ψ modulo ϵ. Lemma 5 establishes that F_2, Ψ simplifies to $F, \Psi\theta'$ modulo θ'. Chaining the definition of simplification modulo transitively establishes the lemma.

With Corollary 4 and Lemma 7 we have then established Proposition 2.

It is worth examining why the side-conditions for simplification modulo are used. As the following example shows, transformations that only preserve satisfiability but strengthen formulas cannot be used easily in an incremental setting.

Example 6. Let F_0 be the satisfiable formula $x \simeq y \wedge y \leq z \wedge z \simeq v$. In that formula x, y are equal, and z, v are equal. Lets assume that we simplify via the solution where the classes are merged (i.e. where $y \simeq z$). It is satisfiability preserving. It suggests a transformation that we call FLEX†.

$$\frac{x \simeq y \wedge y \leq z \wedge z \simeq v \parallel \varepsilon}{x \simeq z \wedge z \simeq v \parallel \langle y \leftarrow z; (x \simeq y \wedge y \leq z)\rangle^\top} \text{ FLEX}^\dagger$$

The resulting state is still satisfiable. Now UNDO can be applied without any problems. The result is still satisfiable, but not equivalent to F_0 (does not have the models where the two equivalence classes are not merged).

$$\frac{x \simeq z \wedge z \simeq v \parallel \langle y \leftarrow z; (x \simeq y \wedge y \leq z)\rangle^\top}{(x \simeq y \wedge y \leq z) \wedge x \simeq z \wedge z \simeq v \parallel \varepsilon} \text{ UNDO}$$

Adding the constraint $y \simeq z - 1$ to F_0 would be satisfiable, but adding it to our formula is unsatisfiable.

4 Simplifcation Methods

Many simplification methods used in practice during pre-processing are equivalence preserving. These methods include formula rewriting, constant propagation, NNF conversion, quantifier elimination, and bit-blasting. They do not require the methodology from this paper and have been integral in Z3 since its inception. We will here discuss main simplification pre-processing routines that do not preserve equivalence and how they relate to our taxonomy.

4.1 Equality Solving

One of the most useful pre-processing techniques eliminates symbols when they can be *solved*, that is, a constraint implies an equality $x \simeq t$, where t is a term that does not contain x. Equality solving corresponds to finding unitary solutions to unification problems modulo theories. Most uses of equality solving are captured by transformations justified by rule RIGID. In Z3, equality solving comprises of a two stage process:

1. Extract a set of solution candidates $\mathcal{E}$ implied by the current formula φ.
2. Extract from $\mathcal{E}$ a subset of *solutions* that can be oriented without introducing cyclic dependencies.

To elaborate, let $\mathcal{E}$ be a set of solution candidates $x_1 = t_1, \ldots x_n = t_n$. The candidates may contain multiple equalities using the same symbol. For example, $\mathcal{E}$ could be $x = f(x), x = g(y), y = h(z)$. We can't use the solution $x = f(x)$ because x already occurs in $f(x)$. But we can use the solution $x = g(y), y = h(z)$ processed in this order as first x is replaced by $g(y)$, then y is replaced by $h(z)$. In the second stage we extract from $\mathcal{E}$ a subset of equalities $x_{i_1} = t_{i_1}, \ldots, x_{i_k} = t_{i_k}$, where x_{i_j} are distinct and t_{i_j} are terms such that $x_{i_j} \notin t_{i_{j'}}$ for $j \leq j'$. The subset is in triangular form.

Example 7. We illustrate two application of RIGID for eliminating two symbols from three equations. The choice of the first two equations is arbitrary. An alternative simplification could choose to eliminate x and z instead. It is not possible, however, to eliminate all three variables.

$$
\begin{array}{l}
F, x \simeq y+1, y \simeq z+1, z \simeq f(x) \parallel \theta \Longrightarrow^{\text{RIGID}} \\
F[y+1/x], y \simeq z+1, z \simeq f(y+1) \parallel \theta\langle x \leftarrow y+1; x \simeq y+1\rangle^{\perp} \Longrightarrow^{\text{RIGID}} \\
F[y{+}1/x, z{+}1/y], z \simeq f(z{+}2) \parallel \theta\langle x \leftarrow y{+}1; x \simeq y+1\rangle^{\perp}\langle y \leftarrow z+1; y \simeq z+1\rangle^{\perp}
\end{array}
$$

The set of unification modulo theories facilities used in Z3 is based on extracting simple definitions. Foremost, for a conjunct $x \simeq t$ of φ, where x is uninterpreted, $x \neq t$, include the equality candidate $x \simeq t$. Other equality candidates are included from formulas of the form $ite(c, x \simeq t, x \simeq s)$ and arithmetic equalities of the form $x + s \simeq t$, such that $x \simeq t - s$ is a solution candidate for x. Note that solution candidates are not necessarily unique for an equality. The constraint $x + y \simeq t$ can be used as solution to both x and y. If x has a nested occurrence within t, the solution for y, but not x, can be used. Equality solving interacts with simplification pre-processing: equalities over algebraic data-types can be assumed to be in decomposed form already since rewriting simplification decomposes equalities of the form $cons(h_1, t_1) \simeq cons(h_2, t_2)$ into $h_1 \simeq h_2 \wedge t_1 \simeq t_2$. Equality solving can be extended modulo theories in several directions. Arithmetical equalities can be extracted from Diophantine equations solving and polynomial equality factorization as part of establishing a Gröbner basis. Equalities can be extracted from inequalities [6, 31], other theories, such as the theory of arrays allow extracting solutions from equalities $store(a, i, v) \simeq t$,

where a is a symbol that does not occur in t, i, v, as $a \simeq \mathit{store}(t, i, w)$, together with the constraint $\mathit{select}(t, i) \simeq v$, where w is fresh. We leave a study of the cost/benefits of these approaches within the context of incremental preprocessing to future work.

Equality solving is extended to sub-formulas in the following way: When a positive sub-formula implies an equality $x \simeq t$ and the symbol x does not occur outside of the sub-formula then x can be replaced by t within the subformula. The solution is no longer *rigid constrained* but can be justified by FLEX.

Example 8. Suppose $x \notin F, \Psi$, then we can use FLEX to justify the simplification

$$F, (x \simeq t \wedge \Phi[x]) \vee \Psi \parallel \theta \Longrightarrow^{\text{FLEX}} F, \Phi[t] \vee \Psi \parallel \theta\langle x \leftarrow t; (x \simeq t \wedge \Phi[x]) \vee \Psi\rangle^{\top}$$

4.2 Unconstrained sub-terms

Symbols that have a single occurrence in a formula may be solved for based on context. For example, with the formula $x \leq y, y < z, z \leq u, p(u), q(u)$, the constant x can be eliminated by using the solution $x \simeq y$. Then y can be eliminated by setting $y \simeq z - 1$, and finally $z \simeq u$.

Invertibility of unconstrained symbols (see e.g. [8, 7]) in an incremental setting for bit-vectors was introduced in [19]. The method implements the following proof-rule, exemplified for the term $x + t$, containing the only occurrence of x.

INVERT :

$F[x + t] \parallel \theta \Longrightarrow F[y] \parallel \theta\langle x \leftarrow y - t; y \simeq x + t\rangle^{\top}$ if x occurs uniquely in F, y is fresh

To justify rule INVERT in our setting, it suffices to check the condition from Lemma 6. Alternatively, we can use the generic rule UPDATE when applying unconstrained simplifications. The rule INVERT is more efficient than using UPDATE because the latter requires adding back an entire conjunction Ψ where the invertible term $x + t$ occurs. Invertibility can also be used to justify elimination of nested definitions. For a definition $F \wedge ((x \simeq t \wedge \Phi[x]) \vee \Psi)$ (see Example 8), where $x \notin F, \Psi$ can first be rewritten as $F \wedge ((x \simeq t \wedge \Phi[t]) \vee \Psi)$. Then $x \simeq t$ is invertible because it contains the only occurrence of x. The new constraint is $F \wedge ((y \wedge \Phi[t]) \vee \Psi)$ where y is a fresh Boolean symbol.

Invertibility conditions are theory dependent. Figure 4 exemplifies main invertibility conditions for arithmetic[3].

Z3 uses a heap ordered by occurrence counts to identify candidates for invertibility. It first processes all symbols with occurrence count 1. If it is possible to eliminate a symbol with occurrence count 1, the occurrence counts of sub-terms under the term that gets eliminated are decreased. The elimination process stops once the heap only contains symbols with occurrence counts above 1.

[3] A summary of rules used for other theories can be found online: https://microsoft.github.io/z3guide/docs/strategies/summary#tactic-elim-uncnstr

$$F[t - x] \parallel \theta \Longrightarrow^{\textsc{Invert}} F[y] \parallel \theta\langle x \leftarrow t - y; y \simeq t - x\rangle^{\top}$$
$$F[x \cdot x'] \parallel \theta \Longrightarrow^{\textsc{Invert}} F[y] \parallel \theta\langle x, x' \leftarrow y, 1; y \simeq x \cdot x'\rangle^{\top}$$
$$F[x \leq t] \parallel \theta \Longrightarrow^{\textsc{Invert}} F[y] \parallel \theta\langle x \leftarrow ite(y, t, t + 1); y \simeq x \leq t\rangle^{\top}$$
$$F[t \leq x] \parallel \theta \Longrightarrow^{\textsc{Invert}} F[y] \parallel \theta\langle x \leftarrow ite(y, t, t - 1); y \simeq t \leq x\rangle^{\top}$$

Fig. 4. Invertibility rules for symbols x, x' that occur uniquely in F; y is fresh.

4.3 Symbol Elimination and Macros

SAT solvers use symbol elimination [15] to simplify clauses. The first-order version [11] remains timely in more recent works as well [28]. A predicate p can be eliminated if it occurs at most once in every clause either positively or negatively. Clauses that contain p are replaced by resolvents by applying binary resolution exhaustively, and then remove clauses containing p.

Example 9. We illustrate symbol elimination for the ground case with two clauses, and F such that $p \notin F$, as an instance of the Update rule.

$$F, p(t) \vee \Phi, \neg p(s) \vee \Psi \parallel \theta \Longrightarrow^{\textsc{Update}}$$
$$F, s \not\simeq t \vee \Phi \vee \Psi \parallel \theta\langle p \leftarrow \lambda x \,.\, p(x) \vee (x \simeq t \wedge \neg\Phi); p(t) \vee \Phi, \neg p(s) \vee \Psi\rangle^{\top}$$

The same elimination technique can also be applied to Horn clauses where p does not occur both in the head and body of any rule. A solution for the eliminated predicate is a conjunction of the upper bounds for p or a disjunction of lower bounds for p. It is generally a quantified formula. If the involved clauses admit quantifier free interpolants, the solution can also be computed using an interpolant from a solution to the reduced system [4]. Thus, the term t in a substitution $x \mapsto t$ may only be computed *after* an initial model is known.

There are many cases where symbols can be eliminated incrementally and justified by the Rigid rule:

- Macros $\forall x \,.\, f(x) \simeq t[x]$, $\forall x \,.\, f(x) + s \simeq t$ are handled as $\forall x \,.\, f(x) \simeq t - s$, assuming f is not free in s, t. Then replace occurrenes $f(a)$ by $t[a]$, respectively $t[a] - s[a]$.
- Quasi macros $\forall x, y \,.\, f(x, y, x + y) \simeq t[x, y]$, then replace $f(a, b, c)$ by $ite(c \simeq a + b, t[a, b], f'(a, b, c))$, assuming $f \notin t$.
- Conditional macros $\forall x \,.\, f(x) \simeq t[x] \vee C[x]$, then replace $f(a)$ by $ite(C[a], f'(a), t[a])$, where $f \notin t, C$.
- $(f(x) \simeq t) \equiv \psi$, where $f \notin t, \psi$. Then replace $f(a)$ by $ite(\psi, t, f'(a))$ and add the clause $\forall x \,.\, f'(x) \not\simeq t$.

Macro elimination can be extended to ordered structures and in combination of theories [42]. It has been integral to making quantified reasoning with bit-vectors [44] practical. We claim that first-order in-processing rules based on blocked clauses, asymmetric tautology elimination, covered clauses known from SAT [29] can also be captured by Update. We substantiate the claim with an example, but leave a comprehensive treatment for future work:

Example 10. Consider the clause $C := p(x) \vee q(x)$ and $F := \neg p(x) \vee p(f(x)) \vee r(x), \neg p(x) \vee p(f(x)) \vee p(g(x))$. The variable x is universally quantified. Then C can be rewritten to $p(x) \vee q(x) \vee p(f(x))$ without affecting satisfiability. The covered literal $p(f(x))$ was added to C as it occurs in every resolvent with $p(x)$. The model for p has to be fixed, however. The model update is a first-order lifting of the propositional case.

$$F, p(x) \vee q(x) \parallel \theta \Longrightarrow^{\textsc{Update}}$$
$$F, p(x) \vee q(x) \vee p(f(x)) \parallel \theta\langle p \leftarrow \lambda x \,.\, p(x) \vee p(f(x)); \forall x \,.\, p(x) \vee q(x)\rangle^{\top}$$

To illustrate unification constraints in model updates, consider the clause $C := p(h(x)) \vee q(x)$ and $p' := \lambda x \,.\, p(x) \vee \exists y \,.\, x \simeq h(y) \wedge \neg q(y)$:

$$F, p(h(x)) \vee q(x) \parallel \theta \Longrightarrow^{\textsc{Update}}$$
$$F, p(h(x)) \vee q(x) \vee p(f(h(x))) \parallel \theta\langle p \leftarrow p'; \forall x \,.\, p(h(x)) \vee q(x)\rangle^{\top}$$

5 Implementation

We have implemented incremental pre-processing as an integral component of a new SMT solver, part of Z3. It can be enabled by setting the option `sat.smt=true` from the command line. It includes simplification by equality solving, elimination of uninterpreted sub-terms and macro detection as described in Section 4[4]. The primary reason for supporting incremental pre-processing has been usability. GitHub issues pointing to performance cliffs when switching to incremental mode are recurrent. A distilled example where pre-processing can solve formulas is as follows:

Example 11. Consider the benchmark.

```
(set-option :unsat_core true) (set-option :sat.smt true)
(declare-const exp Int) (push)
(assert (! (= exp 1) :named assumption))
(assert (not (= 2 (^ 2 exp)))) (check-sat) (get-unsat-core)
```

The legacy solver of z3 cannot solve it because it only knows about constant folding when expanding the definition of exponentiation (the symbol `^`). With incremental propagation, the equality `(not (= 2 (^ 2 exp)))` simplifies to `false`.

Simplifiers interoperate with user scopes: SMT solvers support scoping using operations `push` and `pop`. All assertions made within a `push` are invalidated by a matching `pop`. To allow simplifiers to inter-operate with recursive function definitions they track symbols used in the bodies of recursive functions as *frozen*. Those symbols are excluded from solving. Similar to CaDiCaL's implementation for replaying clauses (see [18]), our implementation of Replay stores the domain of θ in a hash-set to bypass processing formulas that have no symbols in θ.

[4] See https://microsoft.github.io/z3guide/docs/strategies/simplifiers for a summary of simplifiers.

6 Related Work

6.1 Pre- and in-processing for SAT and QBF

Pre-processing for SAT has received significant attention with the milestone work in Satelite [15] and then using notions of blocked clauses [27] and solution reconstruction [25]. Pre-processing techniques for QBF are discussed for example in [3, 22]. The main pre-processing methods for propositional satisfiability solvers can be captured using our rule Update (see Example 4 for an instance of blocked clause elimination simplification). For the case where $\neg p \vee D$ is a blocked clause, the model update is the de-Morgan dual: removing $\neg p \vee D$ triggers the update $\mathcal{M}[p \mapsto (p \wedge D)^{\mathcal{M}}]$.

The work [18] introduces an inference system that also addresses *redundant* clauses and represents model updates using a notion of *witness labeled clauses*. The semantic content of the rules used for SAT are captured by Update. However, we elided tracking redundant clauses in this work. The case for SMT motivate specialized rules Rigid, Flex and Invert.

6.2 Pre-processing for SMT

Pre-processing simplification is integral in all main SMT solvers, including [33, 2]. Incremental pre-processing with special attention to bit-vectors was introduced in [19]. Transformations considered in this thesis can be represented by the Rigid and Invert rules. Z3 exposes pre-processing simplifications as tactics [13] and allows users to compose them to suit specific needs of applications.

Invertibility conditions are used in [34] to guide local search. This work considers also a candidate value of all symbols. For example, $F[x \cdot t]$ is invertible to $F[y]$ if t evaluates to 1.

6.3 Pre-processing for MIP

Pre-solving is terminology for pre-processing for mixed-integer linear programming solvers. There is a significant repertoire of pre-solving methods integrated in leading MIP solvers. Their effects are well documented in the newer survey [1], which provides an updated perspective to [20]. Pre-solving was developed earlier in [40]. The main methods can be categorized as operating on single rows (single constraints) or single columns (single variables), multiple rows, and multiple columns, and use global information about the tableau. They include also methods known from other domains, such as literal probing also found in SAT solvers, and symmetry reduction for sparse systems [38]. We are not aware of under-constrained simplifications used in mainstream MIP solvers. Only symmetry reduction stands out as outside the scope of incremental pre-solve methods.

Example 12. Pre-processing that combines two rows or combines two columns relies on efficient indexing [21] to be effective. The two column non-zero cancellation method considers the situation where the coefficients to two variables

maintain a high degree of correlation. Consider the following formula

$$2x + 4y + z \leq 5 \ \wedge \ x + 2y + u \leq 6 \ \wedge \ 3x + y + z \leq 3 \ \wedge \ \varphi \text{ where } x, y \notin \varphi.$$

The coefficients to x, y in the first two inequalities are related by the affine relation given by $\lambda = 2$. In this case the system can be reformulated, justified by rule Rigid, by introducing a fresh variable v and using the inequalities

$$2v + z \leq 5 \ \wedge \ v + u \leq 6 \ \wedge \ 3v - 5y + z \leq 3 \ \wedge \ \varphi.$$

6.4 Pre-processing in first- and higher-order provers

Pre-processing is also an important part of first-order theorem provers. Techniques for creating small clausal normal forms have long attracted attention [35]. Main simplifications [24] are based on detecting definitions similar to what is described in Section 4.3, but with the extra twist of ensuring that simplifications preserve first-order decidability, such as ensuring that formulas remain within the EPR fragment. Furthermore a variant of AIGs with nodes representing quantifiers are used to detect shared structure. While [24] is only concerned establishing preservation of satisfiability we note that the classification as model equivalent from Section 4.3 extends to the cases considered. In-processing inspired by SAT was pursued for first-order [29, 43] and recently for higher-order settings [5].

6.5 Constrained Horn Clauses

Constrained Horn Clauses [4], enjoy a tight connection with Logic Programming where several transformation techniques were developed [10, 12], including incremental consequence propagation [36]. *Fold* [9] transformations introduce auxiliary predicates and rules that correspond to replacing a code-block with an auxiliary procedure. It is justified by Rigid. *Unfold* transformations can be justified by Update and correspond to macro elimination.

7 Summary

We introduced a calculus of pre-processing for SMT. It distinguishes simplifications that are *rigid* and so can be applied to new formulas as substitutions. Other simplified formulas may need to be re-introduced similar to re-introducing removed clauses in SAT. We examine several of the pre-processing methods studied in SAT, ATP, MIP and SMT as instances of the calculus. We leave empirical and algorithmic studies of new pre- and in-processing methods to future work. Another angle we have left on the table is reconciling pre-processing with in-processing. For SAT, it was useful to develop a calculus that accounted for both irredundant and redundant clauses. In our current effort we have set this angle aside in favour of establishing main properties on replaying substitutions.

Acknowledgment Thanks to the reviewers for their extensive constructive feedback and to Diego Olivier Fernandez Pons for introducing us to MIP presolving. The research was partially funded by the Austrian Science Fund (FWF) under project No. T-1306.

References

1. Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS J. Comput.*, 32(2):473–506, 2020.
2. Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.
3. Armin Biere, Florian Lonsing, and Martina Seidl. Blocked clause elimination for QBF. In Nikolaj S. Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 101–115. Springer, 2011.
4. Nikolaj S. Bjørner, Arie Gurfinkel, Kenneth L. McMillan, and Andrey Rybalchenko. Horn clause solvers for program verification. In Lev D. Beklemishev, Andreas Blass, Nachum Dershowitz, Bernd Finkbeiner, and Wolfram Schulte, editors, *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, volume 9300 of *Lecture Notes in Computer Science*, pages 24–51. Springer, 2015.
5. Jasmin Blanchette and Petar Vukmirović. Sat-inspired higher-order eliminations, 2023.
6. Martin Bromberger and Christoph Weidenbach. New techniques for linear arithmetic: cubes and equalities. *Formal Methods Syst. Des.*, 51(3):433–461, 2017.
7. Robert Brummayer. *Efficient SMT solving for bit vectors and the extensional theory of arrays*. PhD thesis, Johannes Kepler University of Linz, 2010.
8. Roberto Bruttomesso. *RTL Verification: From SAT to SMT(BV)*. PhD thesis, University of Trento, 2008.
9. Rod M. Burstall and John Darlington. A transformation system for developing recursive programs. *J. ACM*, 24(1):44–67, 1977.
10. Stefano Calzavara, Ilya Grishchenko, and Matteo Maffei. Horndroid: Practical and sound static analysis of android applications by SMT solving. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 47–62. IEEE, 2016.
11. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.
12. Emanuele De Angelis, Fabio Fioravanti, John P. Gallagher, Manuel V. Hermenegildo, Alberto Pettorossi, and Maurizio Proietti. Analysis and transformation of constrained horn clauses for program verification. *CoRR*, abs/2108.00739, 2021.
13. Leonardo Mendonça de Moura and Grant Olney Passmore. The strategy challenge in SMT solving. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Computer Science*, pages 15–44. Springer, 2013.

14. David Déharbe, Pascal Fontaine, Stephan Merz, and Bruno Woltzenlogel Paleo. Exploiting symmetry in SMT problems. In Nikolaj S. Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2011.
15. Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.
16. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.
17. Herbert B. Enderton. *A mathematical introduction to logic*. Academic Press, 1972.
18. Katalin Fazekas, Armin Biere, and Christoph Scholl. Incremental inprocessing in SAT solving. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 136–154. Springer, 2019.
19. Anders Franzén. *Efficient Solving of the Satisfiability Modulo Bit-Vectors Problem and Some Extensions to SMT*. PhD thesis, University of Trento, Italy, 2010.
20. Gerald Gamrath, Thorsten Koch, Alexander Martin, Matthias Miltenberger, and Dieter Weninger. Progress in presolving for mixed integer programming. *Math. Program. Comput.*, 7(4):367–398, 2015.
21. Patrick Gemander, Weikun Chen, Dieter Weninger, Leona Gottwald, Ambros M. Gleixner, and Alexander Martin. Two-row and two-column mixed-integer presolve using hashing-based pairing methods. *EURO J. Comput. Optim.*, 8(3):205–240, 2020.
22. Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. squeezebf: An effective preprocessor for qbfs based on equivalence reasoning. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 85–98. Springer, 2010.
23. Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Strong extension-free proof systems. *J. Autom. Reason.*, 64(3):533–554, 2020.
24. Krystof Hoder, Zurab Khasidashvili, Konstantin Korovin, and Andrei Voronkov. Preprocessing techniques for first-order clausification. In Gianpiero Cabodi and Satnam Singh, editors, *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012*, pages 44–51. IEEE, 2012.
25. Matti Järvisalo and Armin Biere. Reconstructing solutions after blocked clause elimination. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 340–345. Springer, 2010.
26. Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010,*

Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings, volume 6015 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2010.

27. Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012.
28. Zurab Khasidashvili and Konstantin Korovin. Predicate elimination for preprocessing in first-order theorem proving. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 2016.
29. Benjamin Kiesl and Martin Suda. A unifying principle for clause elimination in first-order logic. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 274–290. Springer, 2017.
30. Benjamin Kiesl, Martin Suda, Martina Seidl, Hans Tompits, and Armin Biere. Blocked clauses in first-order logic. In Thomas Eiter and David Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 31–48. EasyChair, 2017.
31. Zachary Kincaid, Nicolas Koh, and Shaowei Zhu. When less is more: Consequence-finding in a weak theory of arithmetic. *Proc. ACM Program. Lang.*, 7(POPL):1275–1307, 2023.
32. Oliver Kullmann. On a generalization of extended resolution. *Discret. Appl. Math.*, 96-97:149–176, 1999.
33. Aina Niemetz, Mathias Preiner, and Armin Biere. Boolector 2.0. *J. Satisf. Boolean Model. Comput.*, 9(1):53–58, 2014.
34. Aina Niemetz, Mathias Preiner, and Armin Biere. Precise and complete propagation based local search for satisfiability modulo theories. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, volume 9779 of *Lecture Notes in Computer Science*, pages 199–217. Springer, 2016.
35. Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 335–367. Elsevier and MIT Press, 2001.
36. Germán Puebla and Manuel Hermenegildo. Optimized algorithms for incremental analysis of logic programs. In Radhia Cousot and David A. Schmidt, editors, *Static Analysis*, pages 270–284, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
37. Joseph E. Reeves, Marijn J. H. Heule, and Randal E. Bryant. Preprocessing of propagation redundant clauses. In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings*, volume 13385 of *Lecture Notes in Computer Science*, pages 106–124. Springer, 2022.
38. Karem A. Sakallah. Symmetry and satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 509–570. IOS Press, 2021.

39. Sabrine Saouli, Souheib Baarir, Claude Dutheillet, and Jo Devriendt. Cosysel: Improving SAT solving using local symmetries. In Cezara Dragoi, Michael Emmi, and Jingbo Wang, editors, *Verification, Model Checking, and Abstract Interpretation - 24th International Conference, VMCAI 2023, Boston, MA, USA, January 16-17, 2023, Proceedings*, volume 13881 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2023.
40. Martin W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *INFORMS J. Comput.*, 6(4):445–454, 1994.
41. Uwe Schöning. *Logik für Informatiker*, volume 56 of *Reihe Informatik*. Bibliographisches Institut, 1987.
42. Viorica Sofronie-Stokkermans. Hierarchical and modular reasoning in complex theories: The case of local theory extensions. In Boris Konev and Frank Wolter, editors, *Frontiers of Combining Systems, 6th International Symposium, FroCoS 2007, Liverpool, UK, September 10-12, 2007, Proceedings*, volume 4720 of *Lecture Notes in Computer Science*, pages 47–71. Springer, 2007.
43. Petar Vukmirović, Jasmin Blanchette, and Marijn J. H. Heule. Sat-inspired eliminations for superposition. *ACM Trans. Comput. Log.*, 24(1):7:1–7:25, 2023.
44. Christoph M. Wintersteiger, Youssef Hamadi, and Leonardo Mendonça de Moura. Efficiently solving quantified bit-vector formulas. *Formal Methods Syst. Des.*, 42(1):3–23, 2013.
45. Cunxi Yu, Maciej Ciesielski, and Alan Mishchenko. Fast algebraic rewriting based on and-inverter graphs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(9):1907–1911, 2018.