Finding All Minimal Safe Inductive Sets

Ryan Berryhill[1]*, Alexander Ivrii[3], and Andreas Veneris[1,2]

[1] University of Toronto, Department of Electrical and Computer Engineering
{ryan,veneris}@eecg.utoronto.ca
[2] University of Toronto, Department of Computer Science
[3] IBM Research Haifa ALEXI@il.ibm.com

Abstract. Computing minimal (or even just small) certificates is a central problem in automated reasoning and, in particular, in automated formal verification. For unsatisfiable formulas in CNF such certificates take the form of Minimal Unsatisfiable Subsets (MUSes) and have a wide range of applications. As a formula can have multiple MUSes that each provide different insights on unsatisfiability, commonly studied problems include computing a smallest MUS (SMUS) or computing all MUSes (AllMUS) of a given unsatisfiable formula. In this paper, we consider certificates to safety properties in the form of Minimal Safe Inductive Sets (MSISes), and we develop algorithms for exploring such certificates by computing a smallest MSIS (SMSIS) or computing all MSISes (AllMSIS) of a given safe inductive invariant. More precisely, we show how the well-known MUS enumeration algorithms CAMUS and MARCO can be adapted to MSIS enumeration.

1 Introduction

Computing minimal (or even just small) certificates is a central problem in automated reasoning, and, in particular, in Model Checking. Given an unsatisfiable Boolean formula in conjunctive normal form (CNF), a *minimal unsatisfiable subset* (MUS) is a subset of the formula's clauses that is itself unsatisfiable. MUSes have a wide range of applicability, including Proof-Based Abstraction [18], improved comprehension of verification results through vacuity [22], and much more. It is not surprising that a large body of research is dedicated to efficiently computing MUSes. As a formula can have multiple MUSes, each of which may provide different insights on unsatisfiability, several algorithms have been developed to extract all MUSes from an unsatisfiable formula (`AllMUS`) [2,14,15,19,21], and in particular a smallest MUS of an unsatisfiable formula (`SMUS`) [11]. For a recent application of `AllMUS` and `SMUS` to Model Checking, see [8].

For safety properties, certificates come in the form of safe inductive invariants. A recent trend, borrowing from the breakthroughs in Incremental Inductive Verification (such as `IMC` [17], `IC3` [6], and `PDR` [7]), is to represent such invariants as a conjunction of simple lemmas. Lemmas come in the form of clauses encoding facts about reachable states, and hence the invariant is represented in CNF.

* This work was completed while Ryan Berryhill was an intern at IBM Research Haifa

The problem of efficiently minimizing the set of such lemmas, and especially constructing a *minimal safe inductive subset* (MSIS) of a given safe inductive invariant has applications to SAT-based model checking [5, 10], and has been further studied in [12].

By analogy to MUS extraction, in this paper we consider the problem of computing all MSISes of a given safe inductive invariant (`AllMSIS`), and in particular finding the smallest MSIS (`SMSIS`). The problem of minimizing safe inductive invariants appears on its surface to share many commonalities with minimizing unsatisfiable subsets. However, a key aspect of MUS extraction is *monotonicity*: adding clauses to an unsatisfiable formula always yields an unsatisfiable formula. On the other hand, MSIS extraction seems to lack this monotonicity: adding clauses to a safe inductive formula always yields a safe formula, but it may not yield an inductive one. In spite of non-monotonicity, this paper lifts existing MUS enumeration algorithms to the problem of MSIS enumeration.

The `CAMUS` [15] algorithm solves the `AllMUS` problem using a well-known hitting set duality between MUSes and *minimal correction subsets* (MCSes). This work defines analogous concepts for safe inductive sets called *support sets* and *collapse sets*, and, using a hitting set duality between them, lifts `CAMUS` to MSIS extraction. When considering MSIS (resp., MUS) extraction, the algorithm works by enumerating the collapse sets (resp. MCSes) and then exploiting the duality to enumerate MSISes (resp. MUSes) in ascending order of size (*i.e.,* from smallest to largest). When considering the `AllMSIS` problem, the collapse set enumeration step fundamentally limits the algorithm's anytime performance, as MSIS discovery can only begin after that step. However, when considering `SMSIS`, the ability to discover the smallest MSIS first is a significant advantage.

`MARCO` [14], another significant `AllMUS` algorithm, addresses this limitation by directly exploring the power set of a given unsatisfiable CNF formula. In this work, we translate MSIS extraction to a monotone problem and demonstrate how to solve it with `MARCO`. This improves anytime performance when considering the `AllMSIS` problem. However, when considering `SMSIS`, `MARCO` may have to compute every MSIS before concluding that it has found the smallest one.

Towards the goal of better understanding the complexity of MSIS problems, we also lift some well-studied MUS-based decision problems to their MSIS analogs and demonstrate complexity results for those problems. Specifically, we prove that the MSIS identification problem "is this subset of a CNF formula an MSIS?" is D^P-complete (*i.e.,* it can be expressed as the intersection of an NP-complete language and a co-NP-complete language). Further, the MSIS existence problem "does this inductive invariant contain an MSIS with k or fewer clauses?" is found to be Σ_2^P-complete. Both of these results match the corresponding MUS problems' complexities.

Experiments are presented on hardware model checking competition benchmarks. On 200 benchmarks, it is found that the `CAMUS`-based algorithm can find all MSISes within 15 minutes for 114 benchmarks. The most successful `MARCO`-based algorithm is almost as successful, finding all MSISes on 110 benchmarks within the time limit. Further, the `CAMUS`-based algorithm solves the `SMSIS` problem within 15 minutes on 156 benchmarks.

The rest of this paper is organized as follows. Section 2 introduces necessary background material. Section 3 formulates the `AllMSIS` and `SMSIS` problems. Section 4 presents the `CAMUS`-inspired MSIS algorithm while section 5 presents the `MARCO`-based one. Section 6 introduces complexity results for MSIS problems. Section 7 presents experimental results. Finally, section 8 concludes the paper.

2 Preliminaries

2.1 Basic Definitions

The following terminology and notation is used throughout this paper. A literal is either a variable or its negation. A clause is a disjunction of literals. A Boolean formula in Conjunctive Normal Form (CNF) is a conjunction of clauses. It is often convenient to treat a CNF formula as a set of clauses. For a CNF formula φ, $c \in \varphi$ means that clause c appears in φ. A Boolean formula φ is satisfiable (SAT) if there exists an assignment to the variables of φ such that φ evaluates to `1`. Otherwise it is unsatisfiable (UNSAT).

2.2 MUSes, MCSes and Hitting Set Duality

If φ is UNSAT, an UNSAT subformula $\varphi_1 \subseteq \varphi$ is called an UNSAT core of φ. If the UNSAT core is minimal or irreducible (*i.e.,* every proper subset of the core is SAT) it is called a Minimal Unsatisfiable Subset (MUS). A subset $C \subseteq \varphi$ is a Minimal Correction Subset (MCS) if $\varphi \setminus C$ is SAT, but for every proper subset $D \subsetneq C$, $\varphi \setminus D$ is UNSAT. In other words, an MCS is a minimal subset such that its removal would render the formula satisfiable.

The *hitting set duality* between MUSes and MCSes states that a subset C of φ is a MUS if and only if C is a minimal hitting set of $MCSes(\varphi)$, and vice versa. For example, if C is a hitting set of $MCSes(\varphi)$, then C contains at least one element from every MCS and therefore corresponds to an UNSAT subset of φ. Moreover, if C is minimal, then removing any element of C would result in at least one MCS not being represented. Therefore, the resulting formula would be SAT implying that C is in fact a MUS. For more details, see Theorem 1 in [15].

2.3 Safe Inductive Invariants and MSIS

Consider a finite transition system with a set of state variables $\mathcal{V}$. The primed versions $\mathcal{V}' = \{v' | v \in \mathcal{V}\}$ represent the next-state functions. For each $v \in \mathcal{V}$, v' is a Boolean function of the current state and input defining the next state for v. For any formula F over $\mathcal{V}$, the primed version F' represents the same formula with each $v \in \mathcal{V}$ replaced by v'.

A model checking problem is a tuple $P = (Init, Tr, Bad)$ where $Init(\mathcal{V})$ and $Bad(\mathcal{V})$ are CNF formulas over $\mathcal{V}$ that represent the initial states and the unsafe states, respectively. States that are not unsafe are called safe states. The transition relation $Tr(\mathcal{V}, \mathcal{V}')$ is a formula over $\mathcal{V} \cup \mathcal{V}'$. It is encoded in CNF such that $Tr(\boldsymbol{v}, \boldsymbol{v}')$ is satisfiable iff state $\boldsymbol{v}$ can transition to state $\boldsymbol{v}'$.

A model checking instance is Unsafe iff there exists a natural number N such that the following formula is satisfiable:

$$Init(\boldsymbol{v_0}) \wedge \Big(\bigwedge_{i=0}^{N-1} Tr(\boldsymbol{v_i}, \boldsymbol{v_{i+1}})\Big) \wedge Bad(\boldsymbol{v_N}) \tag{1}$$

The instance is Safe iff there exists a formula $Inv(\mathcal{V})$ that meets the following conditions:

$$Init(\boldsymbol{v}) \Rightarrow Inv(\boldsymbol{v}) \tag{2}$$

$$Inv(\boldsymbol{v}) \wedge Tr(\boldsymbol{v}, \boldsymbol{v'}) \Rightarrow Inv(\boldsymbol{v'}) \tag{3}$$

$$Inv(\boldsymbol{v}) \Rightarrow \neg Bad(\boldsymbol{v}) \tag{4}$$

A formula satisfying (2) satisfies *initiation*, meaning that it contains all initial states. A formula satisfying (3) is called *inductive*. An inductive formula that satisfies initiation contains all reachable states and is called an *inductive invariant*. A formula satisfying (4) is *safe*, meaning that it contains only safe states. A safe inductive invariant contains all reachable states and contains no unsafe states, so it is a certificate showing that P is Safe.

For a model checking problem P with a safe inductive invariant Inv_0 in CNF, a subset $Inv_1 \subseteq Inv_0$ is called a *Safe Inductive Subset* (SIS) of Inv_0 relative to P if Inv_1 is also a safe inductive invariant. Furthermore, if no proper subset of Inv_1 is a SIS, then Inv_1 is called a *Minimal Safe Inductive Subset* (MSIS).

2.4 Monotonicity and MSMP

Let $\mathcal{R}$ denote a reference set, and let $p : 2^{\mathcal{R}} \mapsto \{0, 1\}$ be a predicate defined over elements of the power set of $\mathcal{R}$. The predicate p is *monotone* if $p(\mathcal{R})$ holds and for every $\mathcal{R}_0 \subseteq \mathcal{R}_1 \subseteq \mathcal{R}$, $p(\mathcal{R}_0) \Rightarrow p(\mathcal{R}_1)$. In other words, adding elements to a set that satisfies the predicate yields another set that satisfies the predicate.

Many computational problems involve finding a minimal subset that satisfies a monotone predicate. Examples include computing prime implicants, minimal models, minimal unsatisfiable subsets, minimum equivalent subsets, and minimal corrections sets [16]. For example, for MUS extraction the reference set is the original formula φ, and for a subset $C \subseteq \varphi$, the monotone predicate is $p(C) = 1$ iff C is UNSAT. The *Minimal Set over a Monotone Predicate* problem (`MSMP`) [16] generalizes all of these notions to the problem of finding a subset $M \subseteq \mathcal{R}$ such that $p(M)$ holds, and for any $M_1 \subsetneq M$, $p(M_1)$ does not hold. State-of-the-art `MSMP` algorithms heavily rely on monotonicity.

On the other hand, MSIS extraction does not appear to be an instance of `MSMP`. The natural choice of predicate is "$p(Inv) = 1$ iff Inv is a SIS." This predicate is not monotone, as adding clauses to a safe inductive invariant can yield a non-inductive formula.

3 Problem Formulation: AllMSIS and SMSIS

Modern safety checking algorithms (such as `IC3` [6] and `PDR` [7]) return safe inductive invariants represented as a conjunction of clauses, and hence in CNF. In general there is no guarantee that these invariants are simple or minimal. On the other hand, some recent SAT-based model-checking algorithms [5, 10] benefit from simplifying and minimizing these invariants. Given a safe inductive invariant Inv_0 in CNF, some common techniques include removing literals from clauses of Inv_0 [5] and removing clauses of Inv_0 [5, 12].

In this paper, we address the problem of minimizing the set of clauses in a given safe inductive invariant. We are interested in computing a smallest safe inductive subset or computing all minimal safe inductive subsets, as stated below.
Enumeration of all minimal safe inductive subsets (`AllMSIS`): Given a model checking problem $P = (Init, Tr, Bad)$ and safe inductive invariant Inv_0, enumerate all MSISes of Inv_0.
Finding a smallest-sized safe inductive subset (`SMSIS`): Given a model checking problem $P = (Init, Tr, Bad)$ and safe inductive invariant Inv_0, find a minimum-sized MSIS of Inv_0.

On the surface, computing minimal safe inductive subsets of an inductive invariant appears closely related to computing minimal unsatisfiable subsets of an unsatisfiable formula. However, we are not aware of a direct simple translation from `SMSIS` and `AllMSIS` to the analogous MUS problems. This may be due to the lack of monotonicity noted in the previous subsection.

4 MSIS Enumeration Using Hitting Set Duality

In this section we examine precise relationships between different clauses in a safe inductive invariant. We define the notions of a *support set* and a *collapse set* of an individual clause in the invariant, which are somewhat analogous to MUSes and MCSes, respectively. A hitting set duality is identified between support and collapse sets and used to develop an MSIS enumeration algorithm. The algorithm is based on `CAMUS` [15], a well-known algorithm for MUS enumeration. We present a detailed example to illustrate the concepts and algorithm.

4.1 Inductive Support and Collapse Sets

For a clause c in an inductive invariant Inv, c is inductive relative to Inv by definition. However, it may be the case that c is inductive relative to a small subset of Inv. The notions of *support sets*, borrowed from [4], and *minimal support sets* formalize this concept:

Definition 1. *Given a model checking problem $P = (Init, Tr, Bad)$, a safe inductive invariant Inv, and a clause $c \in Inv$, a support set Γ of c is a subset of clauses of Inv relative to which c is inductive (i.e., the formula $\Gamma \wedge c \wedge Tr \wedge \neg c'$ is UNSAT). A minimal support set Γ of c is a support set of c such that no proper subset of Γ is a support set of c.*

Intuitively, minimal support sets of $c \in \mathit{Inv}$ correspond to MUSes of $\mathit{Inv} \wedge c \wedge \mathit{Tr} \wedge \neg c'$ (where the minimization is done over Inv). Thus support sets provide a more refined knowledge of why a given clause is inductive. Note that as c appears unprimed in the formula, it never appears in any of its minimal support sets. Support sets have various applications, including MSIS computations [12] and a recent optimization to `IC3` [3]. The set of all minimal support sets of c is denoted $\mathtt{MinSups}(c)$, and $\mathtt{MinSup}(c)$ denotes a specific minimal support set of c.

Inspired by the duality between MUSes and MCSes, we also consider sets of clauses that cannot be simultaneously removed from a support set. *Collapse sets* and *minimal collapse sets*, defined below, formalize this concept.

Definition 2. *Given a model checking problem $P = (\mathit{Init}, \mathit{Tr}, \mathit{Bad})$, a safe inductive invariant Inv, and a clause $c \in \mathit{Inv}$, a collapse set Ψ of c is a subset of clauses of Inv such that $\mathit{Inv} \setminus \Psi$ is not a support set of c. A minimal collapse set Ψ of c is a collapse set such that no proper subset of Ψ is a collapse set of c.*

We denote by $\mathtt{MinCols}(c)$ the set of all collapse sets of c. Somewhat abusing the notation, we define the *support sets* and *collapse sets* of $\neg \mathit{Bad}$ as related to safety of P. Formally, $\Gamma \subseteq \mathit{Inv}_0$ is a support set of $\neg \mathit{Bad}$ iff $\Gamma \wedge \mathit{Bad}$ is UNSAT. The set $\Psi \subseteq \mathit{Inv}_0$ is a collapse set of $\neg \mathit{Bad}$ if $\mathit{Inv}_0 \setminus \Psi$ is not a support set of $\neg \mathit{Bad}$. Minimal support sets and minimal collapse sets of $\neg \mathit{Bad}$ are defined accordingly. The following lemma summarizes the relations between various definitions.

Lemma 1. *Let Inv_1 be a SIS of Inv_0.*

1. *There exists $\Gamma \in \mathtt{MinSups}(\neg \mathit{Bad})$ such that $\Gamma \subseteq \mathit{Inv}_1$;*
2. *For each $c \in \mathit{Inv}_1$, there exists $\Gamma \in \mathtt{MinSups}(c)$ such that $\Gamma \subseteq \mathit{Inv}_1$;*
3. *For each $\Psi \in \mathtt{MinCols}(\neg \mathit{Bad})$ we have that $\Psi \cap \mathit{Inv}_1 \neq \emptyset$;*
4. *For each $c \in \mathit{Inv}_1$ and for each $\Psi \in \mathtt{MinCols}(c)$ we have that $\Psi \cap \mathit{Inv}_1 \neq \emptyset$.*

The following example illustrates the concept of support sets, which are somewhat analogous to MUSes. The example is extended throughout the paper to illustrate additional concepts and algorithms.

Running Example: Let us suppose that $\mathit{Inv} = \{c_1, c_2, c_3, c_4, c_5, c_6\}$ is a safe inductive invariant for P. Omitting the details on the actual model checking problem, let us suppose that the minimal support sets are given as follows: $\mathtt{MinSups}(\neg \mathit{Bad}) = \{\{c_1, c_2\}, \{c_1, c_3\}\}$, $\mathtt{MinSups}(c_1) = \{\emptyset\}$, $\mathtt{MinSups}(c_2) = \{\{c_4\}, \{c_6\}\}$, $\mathtt{MinSups}(c_3) = \{\{c_5\}\}$, $\mathtt{MinSups}(c_4) = \{\{c_2, c_5\}\}$, $\mathtt{MinSups}(c_5) = \{\{c_3\}\}$, $\mathtt{MinSups}(c_6) = \{\emptyset\}$. In particular, all the following formulas are unsatisfiable: $c_1 \wedge c_2 \wedge \neg \mathit{Bad}$, $c_1 \wedge c_3 \wedge \neg \mathit{Bad}$, $c_1 \wedge \mathit{Tr} \wedge \neg c_1'$, $c_4 \wedge c_2 \wedge \mathit{Tr} \wedge \neg c_2'$, $c_6 \wedge c_2 \wedge \mathit{Tr} \wedge \neg c_2'$, $c_5 \wedge c_3 \wedge \mathit{Tr} \wedge \neg c_3'$, $c_2 \wedge c_5 \wedge c_4 \wedge \mathit{Tr} \wedge \neg c_4'$, $c_3 \wedge c_5 \wedge \mathit{Tr} \wedge \neg c_5'$, $c_6 \wedge \mathit{Tr} \wedge \neg c_6'$. Conversely, the following formulas are satisfiable: $c_1 \wedge \neg \mathit{Bad}$, $c_2 \wedge \neg \mathit{Bad}$, $c_3 \wedge \neg \mathit{Bad}$, $c_2 \wedge \mathit{Tr} \wedge \neg c_2'$, $c_3 \wedge \mathit{Tr} \wedge \neg c_3'$, $c_2 \wedge c_4 \wedge \mathit{Tr} \wedge \neg c_4'$, $c_5 \wedge c_4 \wedge \mathit{Tr} \wedge \neg c_4'$, $c_5 \wedge \mathit{Tr} \wedge \neg c_5'$.

Further, the following example illustrates the "dual" concept of collapse sets, which are somewhat analogous to MCSes.

Running Example (cont.): The minimal collapse sets are given as follows: $\mathtt{MinCols}(\neg \mathit{Bad}) = \{\{c_1\}, \{c_2, c_3\}\}$, $\mathtt{MinCols}(c_1) = \{\emptyset\}$, $\mathtt{MinCols}(c_2) = \{\{c_4, c_6\}\}$,

$\texttt{MinCols}(c_3) = \{\{c_5\}\}$, $\texttt{MinCols}(c_4) = \{\{c_2\},\{c_5\}\}$, $\texttt{MinCols}(c_5) = \{\{c_3\}\}$, $\texttt{MinCols}(c_6) = \{\emptyset\}$.

One way to construct a (not necessarily minimal) SIS of Inv is to choose a minimal support for each clause in the invariant, and then, starting from the support of $\neg Bad$, recursively add all the clauses participating in the supports. In the running example, we only need to make the choices for $\texttt{MinSup}(\neg Bad)$ and for $\texttt{MinSup}(c_2)$, as all other minimal supports are unique. The following example illustrates three different possible executions of such an algorithm, demonstrating that such an approach does not necessarily lead to an MSIS.

Running Example (cont.): Fixing $\texttt{MinSup}(\neg Bad) = \{c_1, c_2\}$ and $\texttt{MinSup}(c_2) = \{c_4\}$ leads to $Inv_1 = \{c_1, c_2, c_3, c_4, c_5\}$. The clauses c_1 and c_2 are chosen to support $\neg Bad$, c_4 is chosen to support c_2, c_5 is needed to support c_4, and c_3 is needed to support c_5. A second possibility fixes $\texttt{MinSup}(\neg Bad) = \{c_1, c_2\}$ and $\texttt{MinSup}(c_2) = \{c_6\}$, which leads to $Inv_2 = \{c_1, c_2, c_6\}$. A third possibility chooses $\texttt{MinSup}(\neg Bad) = \{c_1, c_3\}$ and leads to the $Inv_3 = \{c_1, c_3, c_5\}$, regardless of the choice for $\texttt{MinSup}(c_2)$.

We can readily see that certain choices for minimal supports to do not produce a minimal safe inductive invariant. Indeed, Inv_3 is minimal but Inv_1 is not. The problem has exactly two MSISes represented by Inv_2 and Inv_3, and both also happen to be smallest minimal inductive invariants.

4.2 CAMUS for MSIS Extraction

Our MSIS enumeration algorithm is strongly motivated by `CAMUS` [15], which enumerates all MUSes of an unsatisfiable formula in CNF. Given an unsatisfiable formula φ, `CAMUS` operates in two phases. The first enumerates all MCSes of φ using a MaxSAT-based algorithm. The second phase enumerates all MUSes of φ based on the hitting set duality between MCSes and MUSes. Our algorithm performs similar operations involving the analogous concepts of support and collapse sets.

4.3 The CAMSIS Algorithm

Given a model checking problem $P = (Init, Tr, Bad)$ and safe inductive invariant Inv_0 for P, the algorithm also operates in two phases. The first phase iterates over all $c \in Inv_0 \cup \{\neg Bad\}$ and computes the set $\texttt{MinCols}(c)$ of all minimal collapse sets of c. This is analogous to the first phase of `CAMUS` and is done very similarly. Indeed, collapse sets of c are enumerated by computing an UNSAT core of $Inv_0 \wedge c \wedge Tr \wedge \neg c'$, while minimizing with respect to the clauses of Inv_0.

Now, one possibility is to enumerate all minimal support sets of each $c \in Inv_0 \cup \{\neg Bad\}$, based on the duality between $\texttt{MinCols}(c)$ and $\texttt{MinSups}(c)$, and then to enumerate all MSISes of Inv_0 using a dedicated algorithm that chooses support sets in a way to produce minimal invariants. Instead, we suggest Algorithm 1 to enumerate MSISes of Inv_0 directly, only based on `MinCols` (and without computing `MinSups` first). One can think of this as a SAT-based algorithm for hitting-set duality "with a twist." It uses the last two statements in Lemma 1 to construct a formula in which satisfying assignments correspond to SISes, and then finding all satisfying assignments that correspond to MSISes.

Algorithm 1 `CAMSIS`

```
Input: $Inv_0 = \{c_1, \ldots, c_n\}$, $\texttt{MinCols}(\neg Bad)$, $\texttt{MinCols}(c)$ for every $c \in Inv_0$
Output: $\texttt{MSISes}(Inv_0)$ relative to $P$
 1: introduce new variable $s_c$ for each $c \in Inv_0$
 2: $\vartheta_1 = \bigwedge_{\{d_1,\ldots,d_k\} \in \texttt{MinCols}(\neg Bad)} (s_{d_1} \vee \cdots \vee s_{d_k})$
 3: $\vartheta_2 = \bigwedge_{c \in Inv_0} \bigwedge_{\{d_1,\ldots,d_k\} \in \texttt{MinCols}(c)} (\neg s_c \vee s_{d_1} \vee \cdots \vee s_{d_k})$
 4: $\vartheta = \vartheta_1 \wedge \vartheta_2$
 5: $j \leftarrow 1$
 6: loop
 7:   while $(\vartheta \wedge \texttt{AtMost}(\{s_{c_1}, \ldots, s_{c_n}\}, j))$ is SAT (with model $M$) do
 8:     Let $Inv = \{c_i \mid M \models (s_{c_i} = 1)\}$
 9:     $\vartheta \leftarrow \vartheta \wedge (\bigvee_{c_i \in Inv} \neg s_{c_i})$
10:     $\texttt{MSISes} \leftarrow \texttt{MSISes} \cup \{Inv\}$
11:   end while
12:   break if $\vartheta$ is UNSAT
13:   $j \leftarrow j + 1$
14: end loop
15: return MSISes
```

The algorithm accepts the initial safe inductive invariant $Inv_0 = \{c_1, \ldots, c_n\}$, the set of minimal collapse sets of $\neg Bad$, and the set of minimal collapse sets for each clause in the invariant. All SAT queries use an incremental SAT solver. On line 1, an auxiliary variable s_c is introduced for each clause c. The intended meaning is that $s_c = 1$ iff c is selected as part of the MSIS. On lines 2–4, the algorithm constructs a formula ϑ that summarizes Lemma 1. First, for each minimal collapse set Ψ of $\neg Bad$, at least one clause of Ψ must be in the invariant. This ensures that the invariant is safe. Further, for each selected clause c (*i.e.,* where $s_c = 1$) and for each minimal collapse set Ψ of c, at least one clause of Ψ must be in the invariant. This ensures that each selected clause is inductive relative to the invariant, thereby ensuring that the resulting formula is inductive.

The algorithm uses the `AtMost` cardinality constraint to enumerate solutions from smallest to largest. The loop on line 6 searches for MSISes using ϑ. It starts by seeking MSISes of cardinality 1 and increases the cardinality on each iteration. Each time an MSIS is found, all of its supersets are blocked by adding a clause on line 9. Line 12 checks if all MSISes have been found using ϑ without any `AtMost` constraint. This check determines if any MSISes of *any* size remain, and if not the algorithm exits the loop. The following example illustrates an execution of the algorithm.

Running Example (cont.): Initially, $\vartheta = (s_1) \wedge (s_2 \vee s_3) \wedge (\neg s_2 \vee s_4 \vee s_6) \wedge (\neg s_3 \vee s_5) \wedge (\neg s_4 \vee s_2) \wedge (\neg s_4 \vee s_5) \wedge (\neg s_5 \vee s_3)$. Let $S = \{s_1, \dots, s_6\}$. It is easy to see that both $\vartheta \wedge \texttt{AtMost}(S, 1)$ and $\vartheta \wedge \texttt{AtMost}(S, 2)$ are UNSAT. Suppose that the first solution returned for $\vartheta \wedge \texttt{AtMost}(S, 3)$ is $s_1 = 1, s_2 = 0, s_3 = 1, s_4 = 0, s_5 = 1, s_6 = 0$. It corresponds to a (minimum-sized) safe inductive invariant $\{c_1, c_3, c_5\}$. It is recorded and ϑ is modified by adding the clause $(\neg s_1 \vee \neg s_3 \vee \neg s_5)$. Rerunning on $\vartheta \wedge \texttt{AtMost}(S, 3)$ produces another solution $s_1 = 1, s_2 = 1, s_3 = 0, s_4 = 0, s_5 = 0, s_6 = 1$, corresponding to the MSIS $\{c_1, c_2, c_6\}$. It is recorded and ϑ is modified by adding $(\neg s_1 \vee \neg s_2 \vee \neg s_6)$. Now $\vartheta \wedge \texttt{AtMost}(S, 3)$ is UNSAT. In addition, ϑ is UNSAT and the algorithm terminates.

We now prove the algorithm's completeness and soundness. The proof relies on the fact that satisfying assignments of the formula ϑ constructed on line 4 are safe inductive subsets of the given inductive invariant. The theorem below demonstrates this fact.

Theorem 1. *Each satisfying assignment $M \models \vartheta$ corresponds to a SIS of Inv_0.*

Proof. ϑ is the conjunction of ϑ_1 and ϑ_2. Each clause of ϑ_2 relates to a clause $c \in Inv_0$ and collapse set Ψ of c. It requires that either c is not selected or an element of Ψ is selected. ϑ_2 contains all such constraints, so it requires that for each clause $c \in Inv_0$, either c is not selected or a member of every minimal collapse set of c is selected. By duality, this is equivalent to requiring a support set of c is selected. Therefore ϑ_2 requires that an inductive formula is selected.

ϑ_1 encodes the additional constraint that a support set of $\neg Bad$ is selected. In other words, it requires that a safe formula is selected. Since each clause of Inv_0 must satisfy initiation by definition, a satisfying assignment of ϑ corresponds to a safe inductive invariant contained within Inv_0.

□

Corollary 1. *Algorithm 1 returns the set of all MSISes of Inv_0.*

Proof. The algorithm finds only minimal models of ϑ and finds all such models.

Several simple optimizations are possible. The technique in [12] describes an algorithm to identify certain clauses that appear in every inductive invariant. If such a set $\mathcal{N}$ is known in advance, it is sound to add constraints (s_c) to ϑ for each $c \in \mathcal{N}$ and start the search from cardinality $|\mathcal{N}|$. Further, it is possible to start by finding collapse sets only for the clauses in $\mathcal{N}$, and then find collapse sets for the clauses in those collapse sets, and so on until a fixpoint is reached.

5 MARCO and MSIS Extraction

In this section we show how the `MARCO` algorithm for MUS enumeration [14] can be adapted for MSIS enumeration. In Section 5.1 we present the `MARCO` algorithm from [14] trivially extended to a more general class of *monotone predicate* problems [16]. In Section 5.2 we describe a monotone reformulation of the MSIS extraction problem and fill in the missing details on the special functions used by the `MARCO` algorithm.

5.1 MARCO algorithm for MSMP

Algorithm 2 displays the basic `MARCO` algorithm from [14] trivially extended to the more general class of monotone predicates [16]. The algorithm accepts a monotone predicate p and a set F satisfying $p(F) = 1$. It returns the set of all minimal subsets of F satisfying p. Recall that the *monotonicity* of p means that $p(F_0) \Rightarrow p(F_1)$ whenever $F_0 \subseteq F_1$.

Algorithm 2 `MARCO` for MSMP

```
Input: monotone predicate $p$, formula $F$ in CNF s.t. $p(F) = 1$
Output: set $M$ of all minimal subsets of $F$ that satisfy $p$
 1: $map \leftarrow \top$
 2: while $map$ is SAT do
 3:     $seed \leftarrow$ getUnexplored($map$)
 4:     if $p(seed) = 0$ then
 5:         $mss \leftarrow$ grow($seed$)
 6:         $map \leftarrow map \wedge$ blockDown($mss$)
 7:     else
 8:         $mus \leftarrow$ shrink($seed$)
 9:         $M \leftarrow M \cup \{mus\}$
10:         $map \leftarrow map \wedge$ blockUp($mus$)
11:     end if
12: end while
13: return $M$
```

`MARCO` directly explores the power set lattice of the input set F. In greater detail, it operates as follows. *Seeds* are selected using a Boolean formula called the map, where each satisfying assignment corresponds to an unexplored element of the power set. This is handled by the `getUnexplored` procedure on line 3. The map has a variable for each element of F, such that the element is selected as part of the seed iff the variable is assigned to `1`. Initially, the map is empty and the first seed is chosen arbitrarily.

If $p(seed) = 0$, the `grow` procedure attempts to expand it to a larger set mss that also does not satisfy p (line 5). This can be accomplished by adding elements of $F \setminus seed$ and checking if the result satisfies p. If so, the addition of the element is backed out, otherwise it is kept. Once every such element has been tried, the result it a maximal set that does not satisfy p. Since any subset of such a set does not satisfy p, the algorithm blocks mss and all of its subsets from consideration as future seeds by adding a new clause to the map (line 6).

Conversely, if $p(seed) = 1$, it is shrunk to a minimal such set (an MSMP) by removing clauses in a similar fashion using the `shrink` procedure (line 8). Subsequently, the minimal set and all of its supersets are blocked by adding a clause to the map (line 10). This is because a strict superset of such a set is not minimal, and therefore not an MSMP.

MUS enumeration is a concrete instantiation of this algorithm. The natural choice of predicate is $p(F) = 1$ iff F is unsatisfiable. The `shrink` subroutine returns a MUS of *seed*. The `grow` subroutine returns a maximal satisfiable subset of F containing *seed*.

5.2 A monotone version of MSIS enumeration

Suppose that we are given a safe inductive invariant Inv_0. In order to extract MSISes of Inv_0 with `MARCO`, it is necessary to construct a monotone predicate such that the minimal subsets satisfying this monotone predicate are MSISes. As we saw before, the predicate $p(F) =$ "is F a SIS of Inv_0?" is not monotone. However, let us define the predicate $p_0(F) =$ "does F contain a SIS of Inv_0?'

Lemma 2. *The predicate p_0 defined above is monotone. Furthermore, minimal subsets of Inv_0 satisfying p_0 are MSISes of Inv_0.*

Proof. To show monotonicity of p_0, suppose that $F_0 \subseteq F_1 \subseteq Inv_0$ and suppose that $G \subseteq F_0$ is a SIS of F_0. Then G is also a SIS of F_1. For the second property, note that a *minimal* set that contains a SIS must be a SIS itself. □

In order to apply p_0 for computing MSISes, we need to specify the missing subroutines of the `MARCO` algorithm, or equivalently we need to show how to compute $p_0(seed)$, and how to implement `shrink` and `grow`.

In order to compute $p_0(seed)$, we need check whether *seed* contains a SIS. We accomplish this using the algorithm `MaxIndSubset` that computes a *maximal inductive subset* of a potentially non-inductive set of clauses. Following [12], we compute `MaxIndSubset`(R) of a set of clauses R by repeatedly removing those clauses of R that are not inductive with respect to R, and we check whether the fixpoint R_0 of this procedure is safe using a SAT solver. In particular we can replace the condition $p_0(F) =$ "does F contain a SIS" by an equivalent condition "is `MaxIndSubset`(F) safe?"

The `shrink` procedure involves finding an MSIS of `MaxIndSubset`$(seed)$. A basic algorithm that finds a single MSIS is presented in [5]. Given a safe inductive invariant R, this algorithm repeatedly selects a clause c in R and checks whether `MaxIndSubset`$(R\backslash\{c\})$ is safe. If so, then R is replaced by `MaxIndSubset`$(R\backslash\{c\})$. For more details and optimizations, refer to [12].

The `grow` procedure expands a seed that does not contain a SIS to a maximal subset of Inv_0 that does not contain a SIS. A basic algorithm repeatedly selects a clause $c \in Inv_0 \setminus seed$ and checks whether $p_0(seed \cup \{c\}) = 0$. If so, then *seed* is replaced by $seed \cup \{c\}$. The following continuation of the running example demonstrates several iterations of `MARCO`.

Running Example (cont.): Assume that `grow` and `shrink` are implemented as described above, and the clauses are always processed in the order of their index. On the first iteration, suppose `getUnexplored`(map) returns $seed = \emptyset$. The `grow` procedure initially sets $mss = seed = \emptyset$ and makes the following queries and updates:

- `MaxIndSubset`$(\{c_1\}) = \{c_1\}$ is not safe; $mss \leftarrow mss \cup \{c_1\}$;
- `MaxIndSubset`$(\{c_1, c_2\}) = \{c_1\}$ is not safe; $mss \leftarrow mss \cup \{c_2\}$;
- `MaxIndSubset`$(\{c_1, c_2, c_3\}) = \{c_1\}$ is not safe; $mss \leftarrow mss \cup \{c_3\}$;
- `MaxIndSubset`$(\{c_1, c_2, c_3, c_4\}) = \{c_1\}$ is not safe; $mss \leftarrow mss \cup \{c_4\}$;
- `MaxIndSubset`$(\{c_1, c_2, c_3, c_4, c_5\}) = \{c_1, c_2, c_3, c_4, c_5\}$ is safe;
- `MaxIndSubset`$(\{c_1, c_2, c_3, c_4, c_6\}) = \{c_1, c_2, c_6\}$ is safe.

Thus at the end we obtain $mss = \{c_1, c_2, c_3, c_4\}$. Next, *map* is updated to $map \wedge$ `blockDown`$(\{c_1, c_2, c_3, c_4\})$, forcing *seed* to include either c_5 or c_6 from thereon. On the second iteration, let's suppose that `getUnexplored`(map) returns $seed = \{c_1, c_2, c_3, c_4, c_5, c_6\}$. The `shrink` procedure initially sets $mus =$ `MaxIndSubset`$(seed) = \{c_1, c_2, c_3, c_4, c_5, c_6\}$ and makes the following queries and updates:

- `MaxIndSubset`$(mus \setminus \{c_1\}) = \{c_2, c_3, c_4, c_5, c_6\}$ is not safe;
- `MaxIndSubset`$(mus \setminus \{c_2\}) = \{c_1, c_3, c_5, c_6\}$ is safe; $mus \leftarrow \{c_1, c_3, c_5, c_6\}$;
- `MaxIndSubset`$(mus \setminus \{c_3\}) = \{c_1, c_6\}$ is not safe;
- `MaxIndSubset`$(mus \setminus \{c_5\}) = \{c_1, c_6\}$ is not safe;
- `MaxIndSubset`$(mus \setminus \{c_6\}) = \{c_1, c_3, c_5\}$ is safe; $mus \leftarrow \{c_1, c_3, c_5\}$.

Hence at the end we obtain $mus = \{c_1, c_3, c_5\}$. This allows to update *map* to $map \wedge$ `blockUp`$\{c_1, c_3, c_5\}$, forcing *seed* to exclude either c_1, c_3 or c_5 thereafter.

It is important to note that in practice `grow` and `shrink` are implemented using additional optimizations. The example uses the simple versions for ease-of-understanding.

6 Complexity of MSIS and MUS

This section briefly summarizes complexity results for the MUS and MSIS identification and existence problems. The results for MUS are well-known while, as far as we know, the results for MSIS are novel. Note that the algorithms presented in sections 4 and 5 solve the *function problems* of `AllMSIS` and `SMSIS`. The complexity of related MUS problems has been studied in works such as [13]. In this section, we study closely-related *decision problems*, which are also solved implicitly by the presented algorithms. We present the problems and their known complexity classes below.

Lemma 3. *The MUS existence problem "does F_0 have a MUS of size k or less?" is Σ_2^P-complete.*

Lemma 4. *The MUS identification problem "is F_1 a MUS of F_0?' is D^P-complete.*

Proofs of the above lemmas are presented in [9] and [20], respectively. The novel result for the MSIS existence problem relies on a reduction from the MUS existence problem. The problem is similarly stated as "does Inv_0 have an MSIS of size k or less?" To see that it is in Σ_2^P, notice that it can be solved by a non-deterministic Turing machine that guesses a subset of Inv_0 and checks if it is a SIS, which requires only a constant number of satisfiability queries. We demonstrate that it is Σ_2^P-Hard by reduction from the MUS existence problem.

Theorem 2. *MUS existence problem $\leq_m^P$ MSIS existence problem*

Proof. Let (C, k) be an instance of the MUS existence problem. Construct an MSIS existence instance as follows:

$$\begin{aligned} Inv_C &= \{(c_i \vee \neg Bad) : c_i \in C\} \\ Init &= \neg Bad \\ Tr &= \{(v_i' = v_i) : v_i \in Vars(C)\} \end{aligned}$$

where Bad is a new variable that does not appear in C. Inv_C is a safe inductive invariant because:

1. $Init \Rightarrow Inv_C$ since $\neg Bad$ satisfies every clause
2. Inv_C is inductive because every formula is inductive for Tr
3. $Inv_C \Rightarrow \neg Bad$ since $Inv_C \wedge Bad$ is equi-satisfiable with C, which is UNSAT

Next, we show that every MSIS of Inv_C corresponds to a MUS of C. For any $D \subseteq C$, let $Inv_D = \{(c_i \vee \neg Bad) : c_i \in D\}$. Note that:

1. Every Inv_D satisfies initiation
2. Every Inv_D is inductive
3. $Inv_D \wedge Bad$ is equi-satisfiable with D. It is UNSAT iff D is UNSAT.

The three points above imply Inv_D is a SIS iff D is an UNSAT core, so Inv_D is an MSIS iff D is a MUS. This implies that Inv_C contains an MSIS of size k or less iff C contains a MUS of size k or less. □

Corollary 2. *MUS identification problem $\leq_m^P$ MSIS identification problem*

Proof. Follows from the same reduction used to prove Theorem 2.

We now present the proof that the MSIS identification problem is D^P-complete. It is D^P-Hard due to Corollary 2. The proof that it is in D^P is presented in Theorem 3 below.

Theorem 3. *The MSIS identification problem "is Inv_1 an MSIS of Inv_0?" is in D^P.*

Proof. Let Inv_0 be an safe inductive invariant for $P = (Init, Tr, Bad)$. Let $\mathcal{L}_1$ be the language "subsets of Inv_0 that do not (strictly) contain a SIS." $\mathcal{L}_1$ is in NP. Given a $C_0 \subseteq Inv_0$, `MaxIndSubset`(C_0) [12] would execute at most $|C_0|$ satisfiability queries to determine that C_0 does not contain a SIS. All of the queries return SAT in this case. Given a $C \in \mathcal{L}_1$, executing `MaxIndSubset` on the $|C|$ strict subsets of cardinality $|C| - 1$ yields $O(|C|^2)$ satisfying assignments. They form a certificate for a positive instance with size polynomial in $|C|$.

Let $\mathcal{L}_2$ be the language "subsets of Inv_0 that are SISes." $\mathcal{L}_2$ is in co-NP. This follows from the fact that $C \subseteq Inv$ is a SIS if $C \wedge Tr \wedge \neg C'$ and $C \wedge Bad$ are both UNSAT. The satisfying assignment for the disjunction of those two formulas forms a certificate for a negative instance (where C is not a SIS).

The language "MSISes of Inv_0" is $\mathcal{L}_1 \cap \mathcal{L}_2$, so it is in D^P. □

7 Experimental Results

This section presents empirical results for the presented algorithms and the `MSIS` algorithm of [12] on safe single-property benchmarks from the 2011 Hardware Model Checking Competition [1]. This particular benchmark set was chosen because it has a large number of problems solved during the competition. Experiments are executed on a 2.00 GHz Linux-based machine with an Intel Xeon E7540 processor and 96 GB of RAM. In order to generate inductive invariants for minimization, our implementation of `IC3` is run with a 15 minute time limit, which produces invariants in 280 cases. In 77 cases, `IC3` generates a minimal invariant (including cases where the given property is itself inductive). These benchmarks are removed from further consideration, as are 3 additional benchmarks for which none of the minimization algorithms terminated within the time limit. For each of the 200 remaining testcases, `CAMSIS` (Section 4.3), `MSIS` [12], and `MARCO` (Section 5) are used to minimize the inductive invariant. Motivated by [14], we consider 3 slightly different versions of `MARCO`, by forcing `getUnexplored` to return either any seed satisfying the map (`MARCO-ARB`), the seed of smallest possible cardinality (`MARCO-UP`), or the seed of largest possible cardinality (`MARCO-DOWN`). In this way, `MARCO-UP` favors earlier detection of *mss*es, while `MARCO-DOWN` favors earlier detection of *mus*es (using the terminology of Algorithm 2). Each of the above techniques is run with a time limit of 15 minutes, not including the time required to run `IC3`.

Table 1. Summary of Results

	preparation	k = 1	k = 5	k = 10	k = 100	k = ALL
`MSIS`		200 5,944				
`CAMSIS`	165 34,212	156 42,907	155 43,842	155 43,853	**155** **44,047**	**114** **79,908**
`MARCO-UP`		146 58,904	145 60,461	145 60,725	142 62,021	110 83,934
`MARCO-ARB`		143 56,175	139 60,297	138 61,687	127 71,747	101 93,251
`MARCO-DOWN`		**199** **7,917**	**184** **21,007**	**176** **29,305**	145 57,217	100 92,970

Table 1 summarizes the results. For each technique and for several different values of k, the first line reports the number of testcases on which the technique is able to find k MSISes (or all MSISes if this number does not exceed k). The second line reports the total time, in seconds. In addition, for the `CAMSIS` algorithm, the column "preparation" reports the number of testcases in which it was able to enumerate all collapse sets, and the total time for doing so.

First we note that while `CAMSIS` spends a significant amount of time to compute the collapse sets, it is the winning algorithm when computing all or a large number of MSISes. It is also the winning algorithm for computing the guaranteed smallest MSIS, succeeding in 156 cases. In contrast, the best MARCO-based approach for computing all or the smallest MSIS only succeeds in 110 cases. It is interesting to note that this is the approach favoring earlier detection of *mss*es rather than *mus*es. On the other hand, the MARCO-DOWN approach, which is tailored towards finding *mus*es, shows much better anytime behavior, prevailing over the other algorithms when computing a small number of MSISes (such as 1, 5 or 10). We note that the result for MARCO-DOWN for $k = 1$ is not surprising, as in this case the first assignment returned by getUnexplored returns the original invariant as the seed, so MARCO-DOWN simply reduces to finding any arbitrary MSIS of the original invariant. Finally, we note that the MARCO-ARB algorithm is in general worse than either MARCO-DOWN or MARCO-UP.

To give some intuition on the nature of the problems considered, Table 2 shows the number of MSISes for the 115 testcases solved by at least one AllMSIS algorithm. The largest number of MSISes was 149280. Incidentally, this the only of the 115 benchmarks for which CAMSIS could not find every MSIS.

Table 2. Total Number of MSISes (115 benchmarks)

MSISes	1	2–10	11–100	101–1000	> 1000
Frequency	55	20	17	11	12

A more in-depth analysis shows that while on average the MSIS technique from [12] is significantly better for finding a single MSIS than CAMSIS for finding a smallest-size MSIS, there are several cases where CAMSIS significantly outperforms MSIS. In other words, in some cases first finding all the collapse sets and then finding a minimum inductive invariant using hitting set duality is faster than looking for a minimal inductive invariant directly.

8 Conclusion and Future Work

This work lifts the MUS extraction algorithms of CAMUS and MARCO to the non-monotone problem of MSIS extraction. The former is accomplished by identifying a hitting set duality between support sets and collapse sets, which is analogous to the MCS/MUS duality exploited by CAMUS. The latter is accomplished by converting MSIS extraction to a monotone problem and applying MARCO directly. Further, complexity results are proven demonstrating that MSIS identification is D^P-complete and the MSIS existence problem is Σ_2^P-complete, both of which match the corresponding MUS problems.

The work of [14] suggests many optimizations to MARCO algorithm, it would be interesting to explore these in our context. It would also be of interest to determine if the predicates used to convert the non-monotone MSIS problems into a monotone one suitable for use with MARCO can be applied in CAMSIS. Further, we intend to lift other MUS extraction algorithms such as dualize-and-advance (DAA) [2] to MSIS problems. Finally, further study of the application of MARCO to non-monotone problems and the complexity of doing so is a natural extension of this work.

References

1. Hardware Model Checking Competition 2011. `http://fmv.jku.at/hwmcc11`
2. Bailey, J., Stuckey, P.: Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. Practical Aspects of Declarative Languages pp. 174–186 (2005)
3. Berryhill, R., Ivrii, A., Veira, N., Veneris, A.: Learning support sets in IC3 and Quip: The good, the bad, and the ugly. In: 2017 Formal Methods in Computer Aided Design (FMCAD) (2017)
4. Berryhill, R., Veira, N., Veneris, A., Poulos, Z.: Learning lemma support graphs in Quip and IC3. In: Proceedings of the 2nd International Verification and Security Workshop. IVSW'17 (2017)
5. Bradley, A.R., Somenzi, F., Hassan, Z., Zhang, Y.: An incremental approach to model checking progress properties. In: 2011 Formal Methods in Computer-Aided Design (FMCAD). pp. 144–153 (Oct 2011)
6. Bradley, A.: Sat-based model checking without unrolling. In: International Conf. on Verification, Model Checking, and Abstract Interpretation. VMCAI'11 (2011)
7. Eén, N., Mishchenko, A., Brayton, R.: Efficient implementation of property directed reachability. In: Proceedings of the International Conference on Formal Methods in Computer-Aided Design. FMCAD '11 (2011)
8. Ghassabani, E., Whalen, M., Gacek, A.: Efficient generation of all minimal inductive validity cores. In: 2017 Formal Methods in Computer Aided Design (FMCAD) (2017)
9. Gupta, A.: Learning Abstractions for Model Checking. Ph.D. thesis, Pittsburgh, PA, USA (2006), aAI3227784
10. Hassan, Z., Bradley, A.R., Somenzi, F.: Incremental, inductive CTL model checking. In: Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings. pp. 532–547 (2012), `http://dx.doi.org/10.1007/978-3-642-31424-7_38`
11. Ignatiev, A., Previti, A., Liffiton, M.H., Marques-Silva, J.: Smallest MUS extraction with minimal hitting set dualization. In: Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings. pp. 173–182 (2015), `https://doi.org/10.1007/978-3-319-23219-5_13`
12. Ivrii, A., Gurfinkel, A., Belov, A.: Small inductive safe invariants. In: Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design. FMCAD '14 (2014)
13. Janota, M., Marques-Silva, J.: On the query complexity of selecting minimal sets for monotone predicates. Artif. Intell. 233, 73–83 (2016)
14. Liffiton, M.H., Previti, A., Malik, A., Marques-Silva, J.: Fast, flexible MUS enumeration. Constraints 21(2), 223–250 (Apr 2016), `https://doi.org/10.1007/s10601-015-9183-0`
15. Liffiton, M.H., Sakallah, K.A.: Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning 40(1), 1–33 (Jan 2008), `https://doi.org/10.1007/s10817-007-9084-z`
16. Marques-Silva, J., Janota, M., Belov, A.: Minimal Sets over Monotone Predicates in Boolean Formulae (2013), `https://doi.org/10.1007/978-3-642-39799-8_39`
17. McMillan, K.L.: Interpolation and SAT-Based Model Checking. In: Jr., W.A.H., Somenzi, F. (eds.) CAV. Lecture Notes in Computer Science, vol. 2725, pp. 1–13. Springer (2003)

18. McMillan, K.L., Amla, N.: Automatic Abstraction without Counterexamples. In: Garavel, H., Hatcliff, J. (eds.) TACAS. Lecture Notes in Computer Science, vol. 2619, pp. 2–17. Springer (2003)
19. Nadel, A.: Boosting minimal unsatisfiable core extraction. In: Proceedings of the 2010 Conference on Formal Methods in Computer-Aided Design. pp. 221–229. FMCAD '10 (2010)
20. Papadimitriou, C.H., Wolfe, D.: The complexity of facets resolved. Journal of Computer and System Sciences 37(1), 2 – 13 (1988)
21. Previti, A., Marques-Silva, J.: Partial MUS enumeration. In: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence. AAAI'13 (2013)
22. Simmonds, J., Davies, J., Gurfinkel, A., Chechik, M.: Exploiting Resolution Proofs to Speed Up LTL Vacuity Detection for BMC. In: FMCAD. pp. 3–12. IEEE Computer Society (2007)