DOI: 10.2298/CSIS101019024Z

Extracting Minimal Unsatisfiable Subformulas in Satisfiability Modulo Theories

Jianmin Zhang, Shengyu Shen, Jun Zhang, Weixia Xu, and Sikun Li

Dept. of Computer Science, National University of Defense Technology
410073 Changsha, China
{jmzhang,syshen,junzhang,wxxu,skli}@nudt.edu.cn

Abstract. Explaining the causes of infeasibility of formulas has practical applications in various fields, such as formal verification and electronic design automation. A minimal unsatisfiable subformula provides a succinct explanation of infeasibility and is valuable for applications. The problem of deriving minimal unsatisfiable cores from Boolean formulas has been addressed rather frequently in recent years. However little attention has been concentrated on extraction of unsatisfiable subformulas in Satisfiability Modulo Theories(SMT). In this paper, we propose a depth-first-search algorithm and a breadth-first-search algorithm to compute minimal unsatisfiable cores in SMT, adopting different searching strategy. We report and analyze experimental results obtaining from a very extensive test on SMT-LIB benchmarks.

Keywords: Satisfiability Modulo Theories, minimal unsatisfiable subformula, depth-first-search, breadth-first-search.

1. Introduction

Formal verification has been based on efficient reasoning engines, such as Binary Decision Diagrams(BDD), and more recently propositional satisfiability(SAT) procedures, reasoning at Boolean level. Especially, during the last decade of impressive advances in the efficiency of SAT solvers, SAT-based method are now a fundamental technique in many industrial applications, including many steps of design flows for VLSI chips, such as equivalence checking, property verification, Auto Test Pattern Generation and static timing analysis etc. However the source of hardware design and verification problems has increasingly moved from Boolean level to higher levels: most designers work at register transfer level or even higher levels. The formalism of plain propositional logic is often not suitable or expressive enough for representing many real-world problems, including the verification of RTL or behavioral designs. The high-level structural information is identified as a suitable representation formalism for practical problems of many applications, and thus such problems are more naturally expressible as satisfiability problems in first-order theories, namely Satisfiability Modulo Theories (SMT).

Many real-life problems, arising in formal verification, electronic design, artificial intelligence and other areas, can be formulated as constraint satisfaction

problems, which can be translated into propositional or first-order formulas in conjunctive normal form(CNF). Modern SAT or SMT solvers are able to determine whether a large formula is satisfiable or not. When a formula is unsatisfiable, we are generally interested in a minimal explanation of infeasibility that excludes irrelevant information. Thus it is often required to find a minimal unsatisfiable subformula, or called a minimal unsatisfiable core, that is, an unsatisfiable subset if it becomes satisfiable whenever any of its clauses is removed. Localizing a minimal unsatisfiable subformula is necessary to determine the underlying reasons for the failure, and is used in many practical applications, including model checking on predicate abstraction[1], vacancy detection[2], error localization[3], and synthesizing circuits[4], etc.

In the past decade, there have been considerable research works in finding Boolean unsatisfiable cores[5–29]. However propositional logic is often not expressive enough for representing many practical problems, which can be more naturally addressed in the framework of SMT. The impressive advances of the computational power of SMT solvers makes it possible to extract the unsatisfiable cores from the formulas in SMT. Consequently, the development of effective methods for computing unsatisfiable subformulas in SMT is highlighted as an important goal for the research community. Although some SMT solvers support unsatisfiable subformulas generation, such as CVCLite[30], MathSAT[31] and Yices[32]. A simple and flexible algorithm [33] is the first published works on deriving unsatisfiable cores from formulas in SMT. However, as they said, one limitation of all these approaches is that the resulting unsatisfiable core is not guaranteed to be minimal.

In this paper, we tackle the problem of extracting minimal unsatisfiable cores from practical problem instances in SMT, by the depth-first-search algorithm (Depth-first-search Minimal Unsatisfiable Subformulas Extractor, DFS-MUSE) and the breadth-first-search algorithm (Breadth-first-search Minimal Unsatisfiable Subformulas Extractor, BFS-MUSE). To the best of our knowledge, they are the first works aiming at finding minimal unsatisfiable cores in SMT. Firstly, we present the definitions of the searching graph, and the live node, the dead node and the pending node in the searching graph. Next our algorithms construct the subformulas of original instance as a searching graph, and then recursively remove the clauses not included by the minimal unsatisfiable core from the original formula, adopting depth-first-search or breadth-first-search strategy. Simultaneously they change dynamically the order of variables in the subformula. Some pruning techniques are integrated into the algorithms to remove those unnecessary satisfiability checks as soon as possible, such as conflict clauses sharing and subformulas caching. To evaluate the efficiency of DFS-MUSE and BFS-MUSE, we have performed many experimental tests, and compared them on a large number of SMT-LIB problem instances coming from SMT solver competition benchmarks of CAV 2009.

The binaries of DFS-MUSE and BFS-MUSE are available for downloaded at http://www.ssypub.org/∼zjm/.

The paper is organized as follows. The next section surveys the related work on computing unsatisfiable cores. Section 3 introduces the basic definitions used throughout the paper. Section 4 gives the theoretical analysis for our algorithms. Section 5 proposes DFS-MUSE adopting depth-first-search strategy. Section 6 presents BFS-MUSE using breadth-first-search algorithm. Section 7 shows and analyzes experimental results on the benchmarks used by SMT solver competition of CAV 2009. Finally, Section 8 concludes the paper and outlines future research work.

2. Related Work

There have been many different contributions to research on unsatisfiable subformulas extraction in the last few years, owing to the increasing importance in numerous practical applications. An algorithm[5, 6] for deriving small unsatisfiable cores are based on the ability of DPLL-based SAT solvers to produce resolution refutations. In [7, 8], a method of adaptive core search guided by clauses hardness is employed to extract small unsatisfiable subformulas, but in [9] an exact minimal unsatisfiable core is obtained.

In [10], an algorithm of enumerating all possible subsets is suggested to compute a minimum unsatisfiable core. Another approach is called AMUSE[11], in which selector variables are added to each clause and the unsatisfiable core is derived by a branch-and-bound algorithm on the updated formula. A different algorithm that guarantees minimality is MUP[12], which is mainly a prover of minimal unsatisfiability, as opposed to an unsatisfiable core extractor. Some research works[13–18], based on a relationship between maximal satisfiability and minimal unsatisfiability, have developed some sound techniques for finding all minimal unsatisfiable cores or a minimum unsatisfiable core.

CoreTimmer[19, 20] iterates over each internal node that consumes a large number of clauses and attempts to prove them without these clauses. A scalable algorithm[21], adopting a deeper exploration of resolution refutation, is proposed for minimal unsatisfiable cores extraction. In [22–26], the authors present the algorithms which tracks minimal unsatisfiable subformulas according to the trace of a failed local search run for consistency checking.

A novel algorithm[27] to find minimal unsatisfiable cores is based on Brouwer's fixed point approximation theorem to satisfiability. In [28], the authors present a new framework called constructive, which is based on a combination of a local search procedure and an exhaustive DPLL-like algorithm. Two new resolution-based algorithms are proposed in [29]. The algorithms is used to compute a minimal unsatisfiable core or, if time-out encountered, a small non-minimal unsatisfiable core. These algorithms can be applied to either standard clause-level unsatisfiable subformulas extraction or high-level unsatisfiable subformulas extraction, that is, an extraction of an unsatisfiable subformulas in term of propositional constraints supplied by user application.

However, existing works have very little concern in the literature regarding extraction of unsatisfiable subformulas in SMT. In 2007, Cimatti et al[33] firstly

proposed an algorithm to compute the small unsatisfiable subformulas in SMT. It combines a SMT solver and an external propositional core extractor: Firstly, the SMT solver produces the theory lemmas found during the search; Secondly, the propositional core extractor is called on the boolean abstraction of the original SMT problem and of the theory lemmas, and then obtain a small unsatisfiable core of the original SMT formula. Just as the authors said, one limitation of this algorithm is that the resulting unsatisfiable core in SMT is not guaranteed to be minimal, even if external propositional core extractor returns minimal boolean unsatisfiable subformulas. Whereas up till now, there is no published work devoted to the minimal unsatisfiable subformulas extraction from problem instances in SMT.

3. Preliminaries

Satisfiability Modulo Theories arises in many industrial applications, especially in formal verification and electronic design automation. SMT is defined as the problem of determining the satisfiability of a quantifier-free first-order logic formula with respect to one or more of decidable theories. In a number of practical applications, SMT problem instances typically consist of logical combinations of atoms from different theories, such as the theory of integer linear arithmetic, the theory of arrays, the theory of equality with uninterpreted functions, set theory and so on.

In recent years, a large number of researchers are actively exploring a variety of approaches for solving SMT problems. Most of these methods are classified as eager or lazy techniques. In eager techniques, the input formula is translated using a satisfiability-preserving transformation into a propositional formula which is then checked by a SAT solver for satisfiability. The lazy techniques instead abstract each atom of the input formula by a distinct propositional variable, use a SAT solver to find a propositional model of the formula, and then check that model against the theory solvers. In a word, the substantial advances in algorithms and implementations of SMT solver for several years have inspired the quest of efficient solutions for the problem of minimal unsatisfiable subformulas extraction.

Next some basic definitions and notations used throughout the paper are given as follows:

Definition 1. ***(Unsatisfiable Subformula/Unsatisfiable Core).*** *Given an unsatisfiable formula φ, ψ is an unsatisfiable subformula(core) of φ iff ψ is an unsatisfiable formula and $\psi \subseteq \varphi$.*

Observe that an unsatisfiable core can be defined as any subset, which is infeasible, of the original formula. Consequently, there may exist many different unsatisfiable cores, with different number of clauses, for the same problem instance, such that some of these cores are subsets of others.

Definition 2. ***(Minimal Unsatisfiable Subformula/Minimal Unsatisfiable Core).*** *Given an unsatisfiable formula φ, and ψ is an unsatisfiable subformula(core) of*

φ. Then ψ is a minimal unsatisfiable subfromula(core) iff removing any clause $\omega \in \psi$ from ψ implies that $\psi - \{\omega\}$ is satisfiable.

According to the definition, an unsatisfiable subformula is minimal if it becomes satisfiable whenever any of its clauses is removed, in other words, all of its proper subsets are satisfiable. We should note that in many cases the size of an unsatisfiable subformula may be much larger than the size of a minimal unsatisfiable subformula, because the non-minimal unsatisfiable subformulas probably contain many redundant clauses which cannot be found by simple resolution rules.

4. Theoretical Analysis

We propose two algorithms to extract minimal unsatisfiable cores from the formulas in SMT. The searching strategy of two algorithms are respectively depth-first and breadth-fist. An efficient SMT decision procedure, which is based on the DPLL(*T*) decision scheme[34], is integrated into the algorithms of deriving unsatisfiable subformulas. Our algorithms thoroughly utilizes the information generated by the SMT solving process. The algorithms construct the subformulas of original instance in SMT as a searching graph, and derive the minimal unsatisfiable cores adopting different search ways. Firstly, the definition of the searching graph of a formula in SMT is given as follows:

Definition 3. ***(Searching graph).*** *Given an unsatisfiable formula φ in SMT, if a directed acyclic graph $G(V, E, s)$ satisfies the following conditions: (a) it contains only one sink node, which is on behalf of φ; (b) $\forall p \in V \setminus \{s\}$, the node p represents the formula $\psi = \wedge_1^n C_i$; If v is the k-th child node of p, the node v denotes the formula $\phi_k = \wedge_1^n C_i \setminus C_k$, where $v \in V$, $1 \leq k \leq n$; e_{pv} is an edge from the parent node p to the child node v, where $e_{pv} \in E$. Then $G(V, E, s)$ is called a searching graph of φ.*

Suppose φ is an unsatisfiable formula in SMT, $G(V, E, s)$ is the searching graph of φ. Furthermore, we can classify all of the nodes of $G(V, E, s)$ into three categories: the live nodes, the dead nodes and the pending nodes. The following shows the definitions of the three types of nodes.

Definition 4. ***(live node).*** *Given an unsatisfiable formula φ in SMT, and $G(V, E, s)$ is the searching graph of φ. Suppose $v \in V$, and ϕ denoted by v, where $\phi \subseteq \varphi$. Then v is a live node iff ϕ is unsatisfiable.*

Definition 5. ***(dead node).*** *Given an unsatisfiable formula φ in SMT, and $G(V, E, s)$ is the searching graph of φ. Suppose $v \in V$, and ϕ represented by v, where $\phi \subseteq \varphi$. Then v is a dead node iff ϕ is satisfiable.*

Definition 6. ***(pending node).*** *Given an unsatisfiable formula φ in SMT, and $G(V, E, s)$ is the searching graph of φ. Suppose $v \in V$. If the search process has not reach the node v, v is called a pending node.*

Next the definition of transition relation of the live nodes, the dead nodes and the pending nodes is given as follows:

Definition 7. ***(the transition of nodes).*** *Given an unsatisfiable formula φ in SMT, and $G(V, E, s)$ is the searching graph of φ. In $G(V, E, s)$, the transitions from pending nodes to dead nodes and live nodes are defined as: (a) pending nodes $\rightarrow$ dead nodes: When a subformula corresponding to a pending node is proved to be satisfiable, the pending node is changed to a dead node; (b) pending nodes $\rightarrow$ live nodes: When a subformula denoted by a pending node is unsatisfiable, the pending node is changed to a live node.*

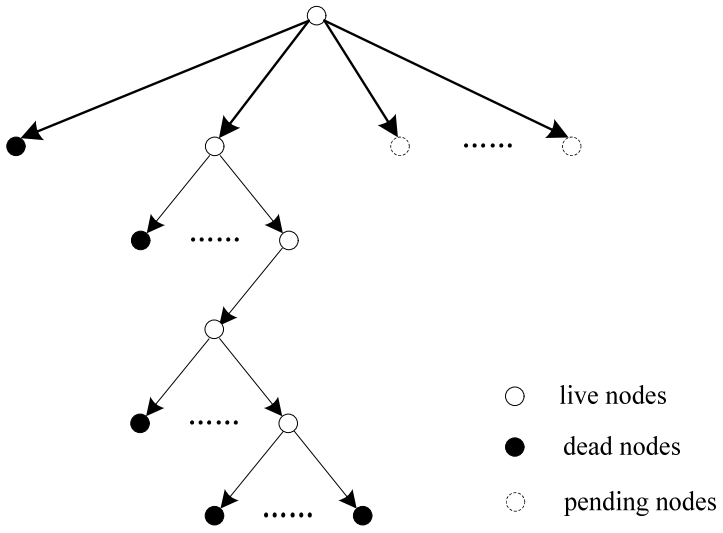

Fig. 1. An example of a searching graph including three types of nodes

Fig.1 shows the sketch of a searching graph including three types of nodes. Suppose φ is an unsatisfiable formula in SMT, and then DFS-MUSE or BFS-MUSE builds a searching graph $G(V, E, s)$, in which the original formula φ is represented by the sink node *s*, and each internal node corresponds to one of the subformulas of φ. However, what is the relationship between the minimal unsatisfiable cores of a formula and the nodes of the corresponding searching graph? According to the above definitions, we may come to the following conclusions.

Theorem 1. *Given an unsatisfiable formula φ in SMT, and $G(V, E, s)$ is the searching graph of φ. Then the subgraph, in which the sink corresponds to a dead node, cannot contain an unsatisfiable subformula.*

Proof. Proof by contradiction. Given a dead node $v \in V$, and ψ is the subformula corresponding to v, where $\psi \subseteq \varphi$. According to Definition 5, ψ is satisfiable. We assume there exists an unsatisfiable subformula $\phi \subseteq \psi$. Further n clauses $\{\omega_1, \cdots, \omega_n\}$ are joined into ϕ, and then we get η=$\phi \cup\{\omega_1, \cdots, \omega_n\}$, where $\{\omega_1, \cdots, \omega_n\} \subseteq \{\psi - \phi\}$. Since η contains all clauses belonging to ϕ, namely $\phi \subseteq \eta$, the truth assignment to the literals in η will eventually lead to a conflict. It indicates that η is unsatisfiable. Eventually, when $\{\omega_1, \cdots, \omega_n\}$=$\{\psi - \phi\}$, η= $\phi \cup \{\psi - \phi\}$=$\psi$ is unsatisfiable. However, v is a dead node and ψ is satisfiable. Therefore, it results in a contradiction, and the assumption is false.

Theorem 2. *Given an unsatisfiable formula φ in SMT, and $G(V, E, s)$ is the searching graph of φ. Then a subformula ϕ denoted by the live node v is a minimal unsatisfiable core, iff all children of v are the dead nodes, where $\phi \subseteq \varphi$ and $v \in V$.*

Proof. Given a live node $v \in V$, and ψ is the subformula corresponding to v, where $\psi = \wedge_1^n C_i$ and $\psi \subseteq \varphi$. According to Definition 4, ψ is an unsatisfiable core of φ. Firstly, suppose ϕ is a minimal unsatisfiable core of φ, and we try to prove all children of v are the dead nodes. According to Definition 2, $\phi_i = \psi - \{C_i\}$ is satisfiable, where $\forall 1 \leq i \leq n$, $C_i \in \psi$. Moreover from Definition 3, the i-th child node of v is represented by the subformula ϕ_i. Therefore, all children of v are the dead nodes. Secondly, we assume all children of v are the dead nodes, and try to prove ϕ denoted by the live node v is a minimal unsatisfiable core. According to Definition 3 and 5, $\forall 1 \leq k \leq n$, each $\phi_k = \wedge_1^n C_i - \{C_k\}$, corresponding to the k-th child node of v, is satisfiable. Eventually from Definition 2, ϕ is a minimal unsatisfiable core of φ.

Suppose φ is an unsatisfiable formula in SMT, our algorithms construct a searching graph $G(V, E, s)$ for φ, where the original formula φ is represented by the sink node, and each internal node corresponds to a subformula of φ. According to the above theorems, the algorithms recursively remove the clauses not included by the minimal unsatisfiable core from the original formula φ, respectively adopting depth-first-search or breadth-first-search strategy, until a live node with all children being dead nodes is found. Simultaneously they change dynamically the order of variables in the subformula. Some pruning techniques are integrated into the algorithms to remove those unnecessary satisfiability checks as soon as possible, such as conflict clauses sharing and subformulas caching.

5. Depth-first-search Algorithm

The depth-first-search algorithm, also called DFS-MUSE to compute the minimal unsatisfiable subformulas introduces the searching graph as an organizing framework. Based on the theorems, DFS-MUSE firstly builds a searching graph for the input formula in SMT. Then adopting the depth-first-search strategy, the algorithm heuristically remove the clauses not included by the minimal

unsatisfiable core from the original formula. During the search, a SMT decision procedure using lazy techniques is called to determine the satisfiability of subformulas. Simultaneously some techniques are employed to accelerate the searching process that the dead nodes are cached and the conflict clauses are shared among different subformuals. Moreover, the algorithm changes dynamically the order of variables in the subformula. The depth-first-search algorithm iterates over and over, and will not stop, until a living node with all children being dead nodes is found. Then the subformula denoted by this living node is a minimal unsatisfiable core. Fig.2 provides the pseudo code of DFS-MUSE.

```
DFS_MUSE(formula)
1    SmallUS = ComputeUS(formula)
2    if (SmallUS == formula) then
3        return formula
4    else
5        IsMinUS = VerifyMinimalUS(SmallUS)
6        if (IsMinUS) then
7            return SmallUS
8        else
9            MinimalUS = DFS_MUSE(SmallUS)
10           return MinimalUS

ComputeUS(formula)
1    ite = EliminateITE(formula)
2    abs = AbstratExpression(ite)
3    for (arity = 0; arity < formula.size; arity++) do
4        interim = abs
5        for (count = formula.size; count > 0; count--) do
6            SmallUS = GraphPruning(interim)
7            cnf = BooleanConversion(SmallUS)
8            IsSAT = SATSolve(cnf)
9            if (!IsSAT) then
10               return SmallUS
11       abs = DynamicVarOrder(abs)
12   return formula
```

Fig. 2. Pseudo code of the depth-first-search algorithm

The depth-first-search algorithm employs an incremental way: Firstly, it computes an unsatisfiable subformula; Further, it derives the minimal unsatisfiable subformula. The function called $ComputeUS$ returns an unsatisfiable core of the input formula. After getting an unsatisfiable core, DFS-MUSE judges and branches. If the returned unsatisfiable core is the input subformula named $formula$, $formula$ is the derived minimal unsatisfiable core. Otherwise, according to the above conclusion, the function called $VerifyMinimalUS$ is used to determine whether the unsatisfiable core $SmallUS$ is minimal or not. If $SmallUS$

is the derived minimal unsatisfiable core, DFS-MUSE will stop; or else the approach regards $SmallUS$ as the new input formula, and recursively computes the minimal unsatisfiable core in the depth-first-search way. When the order of branches changes, the depth-first-search algorithm may obtain different minimal unsatisfiable subformulas.

Fig.2 also shows the process of $ComputeUS$ to extract an unsatisfiable core from input formula in SMT. This function firstly builds a searching graph for the input formula, and then finds a live node in the depth-first-search way. The function called *EliminateITE* is to remove the ITE(If-Then-Else) terms from the formulas. Next *AbstratExpression* replaces the literals in the formula by the abstract variables. Then an unsatisfiable subformula is explored in the space of the searching graph. The function named *GraphPruning* is used to prune the redundant subformulas and clauses from the subgraph, by the way of sharing the conflict clauses and caching dead nodes to avoid the unnecessary satisfiability checks. *BooleanConversion* converts the formula to a Boolean formula, and a SAT solver with DPLL procedure engages in determining satisfiability of the Boolean formula. If unsatisfiable, we get an unsatisfiable subformula denoted by that live node. Otherwise the current node is dead and should be abandon. The function called *DynamicVarOrder* changes the order of variables in current subformula according to the frequency of false assignment. Finally, if the iteration is finished, the function will return the input formula itself as the derived unsatisfiable subformula. In Fig.2, *EliminateITE*, *AbstratExpression* and *BooleanConversion* are the functions belonging to a SMT decision procedure based on the DPLL(*T*) techniques.

6. Breadth-first-search Algorithm

The principles of the breadth-first-search algorithm are similar to the depth-first-search algorithm. BFS-MUSE firstly builds the searching graph for the original formula, and then explores a live node, of which all children are dead nodes, in the breadth-first way. The main difference between BFS-MUSE and DFS-MUSE is the searching strategy. BFS-MUSE moves horizontally on the branches of the graph. All of the same size of subformulas are firstly evaluated, and then the smaller ones are considered. DFS-MUSE instead decides the satisfiability of all subformulas with the size decreasing in the same subgraph at first, and then moves to the neighborhood subgraph. The pseudo code of BFS-MUSE, detailed in the later, is given in Fig.3.

The input of the breadth-first-search algorithm is a formula in SMT. The approach constructs a searching graph $G(V, E, s)$ for the original formula, and tries to find a live node with all children being dead nodes, adopting the breadth-first-search strategy. Some functions of the breadth-first-search algorithm, such as *EliminateITE* and *AbstratExpression*, are almost as same as the depth-first-search algorithm. The recursive function called *ComputeMinUS* is employed to extract a minimal unsatisfiable core from the input formula in SMT. There are some differences on the *GraphPruning* function between BFS-MUSE and

```
BFS_MUSE(formula)
1    ite = EliminateITE(formula)
2    abs = AbstratExpression(ite)
3    MinimalUS = ComputeMinUS(abs)
4    return MinimalUS

ComputeMinUS(formula)
1    IsMinUS = true
2    for (arity = 0; arity < formula.size; arity++) do
3        interim = GraphPruning(formula, arity)
4        cnf = BooleanConversion(iterim)
6        IsSAT = SATSolve(cnf)
5        if (!IsSAT) then
7            IsMinUS = false
8            break
9    if (IsMinUS) then
10       return formula
11    else
12       reordered = DynamicVarOrder(interim)
13       MinUS = ComputeMinUS(reordered)
14       return MinUS
```

Fig. 3. Pseudo code of the breadth-first-search algorithm

DFS-MUSE. In Fig.3, a parameter named *arity* is added into the *GraphPruning* function, and indicates the clause from which the function begins to prune the searching graph.

Then the *ComputeMinUS* function converts the pruned subgraph to Boolean formulas, and calls a SAT solver to check its satisfiability. If the formula is unsatisfiable, a child of current live node is also a live node, that is to say, a smaller unsatisfiable core is detected. Therefore, the subformula denoted by the current live node is not a minimal unsatisfiable core. Then the function makes the smaller live node as the new input, and recursively searches a minimal unsatisfiable subformula. The function will not halt until a live node with all children being dead nodes is found. That live node is the derived minimal unsatisfiable core. Similar to the depth-first-search algorithm, the breadth-first-search algorithm also can derive different minimal unsatisfiable subformulas while changing the order of branching.

7. Experimental Results and Analysis

To experimentally evaluate the effectiveness of our algorithms, we have selected a large number of problem instances from the well-known benchmark family[35], which is used in SMT solvers competition affiliated with CAV conference. We compare the depth-first-search algorithm with the breadth-first-search algorithm on these benchmarks. The inputs of two algorithms are the formulas

in SMT-LIB format. The experiments were conducted on a 2.5 GHz Athlon*2 machine having 2 GB memory and running the Linux operating system.

The depth-first-search algorithm and the breadth-first-search algorithm to find minimal unsatisfiable subformulas are implemented in C++. Some functions of an open-source SMT solver called ArgoLib[36] are integrated in our algorithms. The Argolib currently can solve satisfiability of the formulas, which consist of logic combination of atoms from some theories such as the theory of integer and real linear arithmetic(LIA/LRA), the theory of integer and real differential logic(IDL/RDL). The runtime is in seconds, and the value of timeout was set to 1800 seconds.

The experimental results[1] of the depth-first-search algorithm and the breadth-first-search algorithm on 15 typical formulas are listed in Table 1. Table 1 shows the number of variables (vars) and the number of clauses (clas) for each problem instance. Table 1 also provides the runtime in seconds (DFS-MUSE time) of DFS-MUSE and the number of clauses (DFS-MUSE size) in the derived minimal unsatisfiable subformula. The last two columns present the runtime in seconds (BFS-MUSE time) and the size of the minimal unsatisfiable subformula(BFS-MUSE size) extracted by BFS-MUSE.

Table 1. Performance results on 15 typical problem instances

Problem Instances	**vars**	**clas**	**DFS-MUSE**		**BFS-MUSE**	
			time	size	time	size
bad_echos_ascend.base	58	259	5.18	11	**5.03**	11
sc_init_frame_gap.base	58	265	**5.11**	13	5.14	13
good_frame_update.induction	89	439	29.00	161	**28.74**	161
good_frame_update.base	89	467	72.81	311	**67.75**	311
windowreal-safe2-2	37	404	0.73	188	**0.68**	188
windowreal-safe-2	37	404	0.75	195	**0.68**	195
lpsat-goal-1	83	1345	**1.67**	17	1.69	17
lpsat-goal-2	142	2650	**12.30**	1283	17.16	1283
lpsat-goal-3	201	3955	**43.47**	2548	68.55	2548
windowreal-no_t_deadlock-15	219	2933	**176.96**	1351	179.53	1351
windowreal-no_t_deadlock-16	233	3128	**208.13**	1441	236.58	1441
windowreal-no_t_deadlock-17	247	3323	**293.38**	1531	303.32	1531
windowreal-no_t_deadlock-18	261	3519	**347.24**	1622	391.02	1622
windowreal-no_t_deadlock-19	275	3714	**463.78**	1712	497.97	1712
windowreal-no_t_deadlock-20	289	3909	**547.44**	1802	633.77	1802

Fig.4 shows the experimental results of DFS-MUSE as compared with BFS-MUSE on inf-bakery-mutex benchmarks, coming from SMT solver competition

[1] More detailed performance results of two algorithms are available for downloaded at http://www.ssypub.org/∼zjm/

benchmarks. The size of 20 problem instances of inf-bakery-mutex benchmarks ranges from 65 to 1053 clauses.

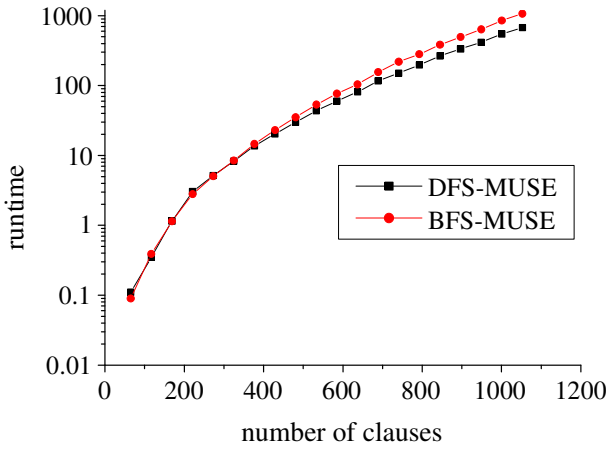

Fig. 4. Experimental results on inf-bakery-mutex benchmarks

Fig.5 shows the experimental results of the depth-first-search algorithm as compared with the breadth-first-search algorithm on windowreal-no_t_deadlock benchmarks, coming from SMT solver competition benchmarks. The size of 20 instances of windowreal-no_t_deadlock benchmarks ranges from 203 to 3908 clauses.

Fig.6 shows the experimental results of DFS-MUSE as compared with BFS-MUSE on pursuit-safety benchmarks, which comes from SMT solver competition benchmarks. The size of 16 instances of pursuit-safety benchmarks ranges from 113 to 1763 clauses.

Fig.7 shows the experimental results of the depth-first-search algorithm as compared with the breadth-first-search algorithm on gasburner-prop3 benchmarks, coming from SMT solver competition benchmarks. The size of 20 instances of gasburner-prop3 benchmarks ranges from 28 to 522 clauses.

From Fig.4 to Fig.7, there is a data point for a result of each formula computed by the depth-first-search algorithm and the breadth-first-search algorithm labelled with different icons, with the position along the vertical axis indicating the runtime in seconds of the algorithms, and the horizontal position indicating the number of clauses contained by every instance. Note that vertical axis has a logarithmic scale. The results of DFS-MUSE are denoted by box icon, and the results of BFS-MUSE are instead denoted by dot icon. The two curves re-

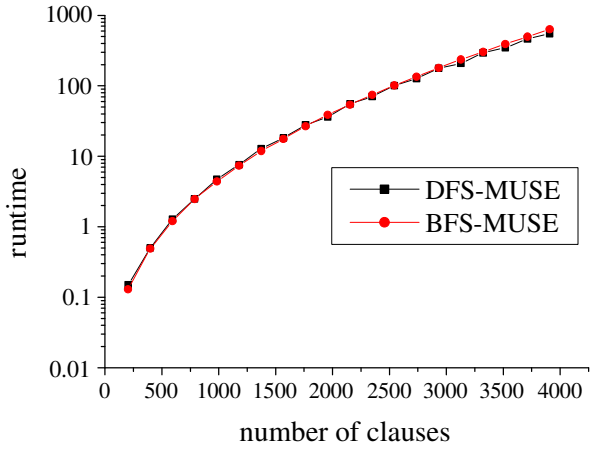

Fig. 5. Experimental results on windowreal-no_t_deadlock benchmarks

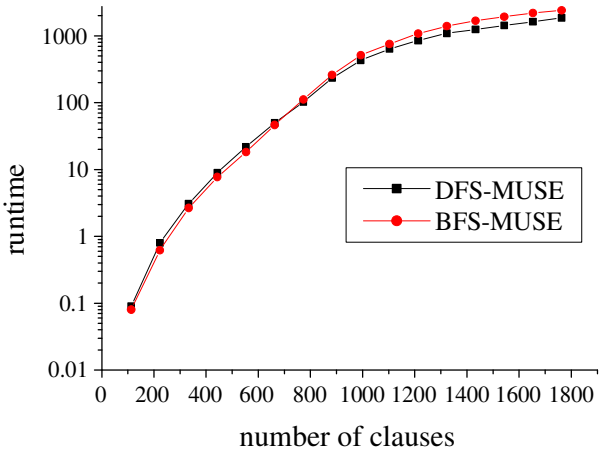

Fig. 6. Experimental results on pursuit-safety benchmarks

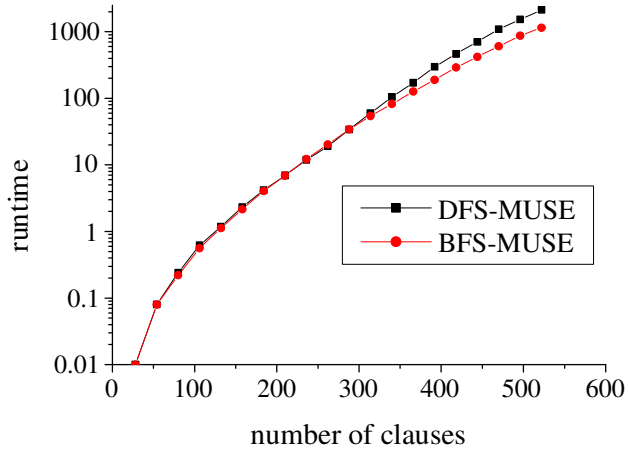

Fig. 7. Experimental results on gasburner-prop3 benchmarks

spectively are represented as the trends of runtime of two algorithms, with the increment of number of clauses included by the benchmarks.

Fig.8 gives the performance results of the depth-first-search algorithm and the breadth-first-search algorithm on more than 200 problem instances of SMT solver competition benchmarks. This figure is a log-log scatter plot that charts the runtime of DFS-MUSE along on the x-axis against the runtime of BFS-MUSE on the y-axis. In other words, each dot in this figure is denoted by the ratio of runtime of the depth-first-search algorithm and the breadth-first-search algorithm. The time unit of two axes is second, and the limit time of both tools was 1800 seconds.

From Table 1, we may observe the following. For all problem instances, the percentage of clauses in the minimal unsatisfiable subformulas is quite small, in most cases from 1% to 50%. Therefore, the minimal unsatisfiable subformulas can generally provide more succinct explanations of infeasibility, and is more valuable for a variety of practical applications.

From Table 1, we also can directly observe that the runtime of the depth-first-search algorithm and the breadth-first-search algorithm has only a little difference. From Fig.4 to Fig.7, we may come to the same conclusion, because two curves respectively corresponding to different approaches are close. Furthermore, the scatter dots are distributed closely around the diagonal in Fig.8. Then we may conclude that DFS-MUSE and BFS-MUSE effectively extract the minimal unsatisfiable cores from the formulas in SMT, and the performance of two approaches is at the same order of magnitude.

In Fig.4, Fig.5 and Fig.6, the breadth-first-search algorithm is generally outperforms the depth-first-search algorithm, when the number of clauses con-

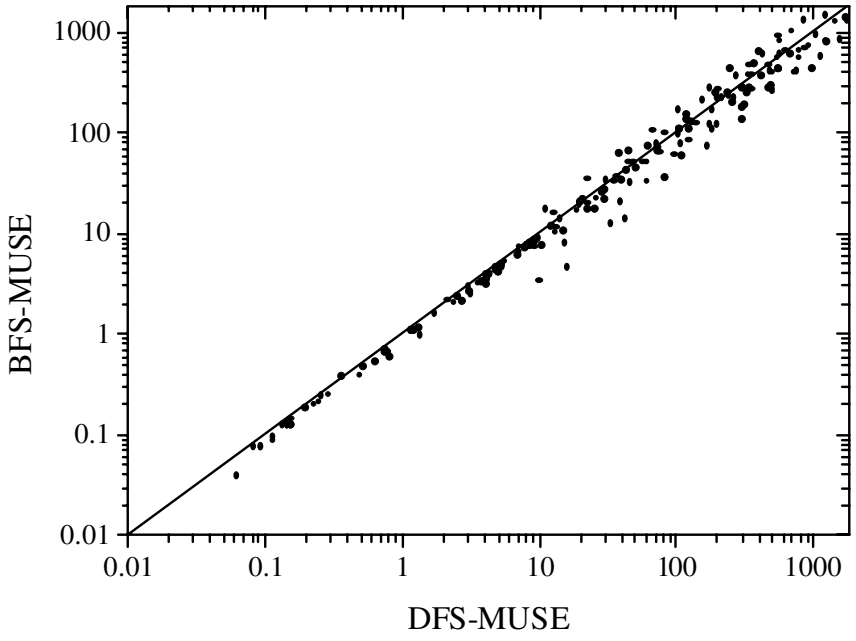

Fig. 8. Performance results on more than 200 instances

tained by original formula is small. However, with the size of formulas increasing, DFS-MUSE is more efficient, and the gap between DFS-MUSE and BFS-MUSE is enlarging. Moreover, in Fig.7, BFS-MUSE always outperforms DFS-MUSE since the size of all problem instances is comparatively small. In Fig.8, more and more scatter dots lie above the diagonal with the runtime increasing. Then we can reach a conclusion: When the size of the original formulas is small, BFS-MUSE is faster in most cases; But while the clauses in the formulas become more and more, DFS-MUSE is instead more efficient. The main causes are that BFS-MUSE is more simple for implementation and performs more moves per second, especially for those formulas with less clauses. On the other hand, after finding an unsatisfiable subformula, DFS-MUSE will continue to search the smaller ones along this subgraph, until reaching a minimal unsatisfiable subformula. This searching way of DFS-MUSE determines that it is more and more efficient while the problem instances containing more and more clauses.

8. Conclusion

We introduce the notation of a searching graph for a SMT formula, and three types of nodes in a searching graph: the live node, the dead node and the pending node. Using the searching graph as an organizing framework, we present

two algorithms called DFS-MUSE and BFS-MSUE to extract the minimal unsatisfiable cores from the formulas in SMT, respectively adopting depth-first-search and breadth-first-search strategy. DFS-MUSE and BFS-MUSE try to recursively remove the redundant clauses from the searching graph and derive a live node with all children being dead nodes. Some pruning techniques are integrated into the approaches, such as conflict clauses sharing and dead nodes caching.

A very extensive tests on SMT-LIB benchmarks are executed to evaluate the effectiveness and performance of DFS-MUSE and BFS-MUSE. The results show that the breadth-first-search algorithm generally outperforms the depth-first-search algorithm on smaller instances, and the depth-first-search algorithm is more efficient than the breadth-first-search algorithm while the formulas contain more and more clauses. The future works is to explore more aggressive techniques to prune the unnecessary satisfiability checks.

Acknowledgments. The authors would like to thank Filip Maric for his good suggestions and kind helps about using ArgoLib. The authors also thank all peer reviewers for their valuable comments and suggestions.This work is supported by the National Natural Science Foundation of China under grant No. 60603088.

References

1. Jain H, Kroening D.: Word level predicate abstraction and refinement for verifying RTL Verilog. In: Proceedings of the 42nd Design Automation Conference (DAC'05), pp. 445–450 (2005)
2. Simmonds J, et al.: Exploiting resolution proofs to speed up LTL vacuity detection for BMC. In: Proceedings of the 7th International Conference on Formal Methods in Computer Aided Design (FMCAD'07), pp. 3–12 (2007)
3. Stuckey PJ, Sulzmann M, Wazny J.: Improving Type Error Diagnosis. In: Proceedings of the 2004 ACM SIGPLAN workshop on Haskell, pp. 80–91 (2004)
4. Shen, S.Y., Qin, Y., Wang K.F., Xiao L.Q., Zhang J.M., Li, S.K.: Synthesizing complementary circuits automatically. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2010, 29(8): 1191–1202.
5. Zhang, L., Malik, S.: Extracting small unsatisfiable cores from unsatisfiable Boolean formula. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, Springer, Heidelberg (2004)
6. Zhang, L., Malik, S.: Validating SAT solvers using an independent resolution-based checker: practical implementations and other applications. In: Proceedings of 2003 Design, Automation and Test in Europe Conference (DATE2003), pp. 10880–10885 (2003)
7. Bruni, R., Sassano, A.: Restoring satisfiability or maintaining unsatisfiability by finding small unsatisfiable subformulae. Electronic Notes in Discrete Mathematics, 2001, 9: 162–173.
8. Bruni, R.: Approximating minimal unsatisfiable subformulae by means of adaptive core search. Discrete Applied Mathematics, 2003, 130(2): 85–100.
9. Bruni, R.: On exact selection of minimally unsatisfiable subformulae. Annals for Mathematics and Artificial Intelligence, 2005, 43: 35–50.
10. Lynce, I., Marques-Silva, J.P.: On computing minimum unsatisfiable cores. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 305–310. Springer, Heidelberg (2005)

11. Oh, Y., Mneimneh, M.N., Andraus, Z.S., Sakallah, K.A., Markov, I.L.: AMUSE: a minimally-unsatisfiable subformula extractor. In: Proceedings of the 41st Design Automation Conference (DAC'04), pp. 518–523 (2004)
12. Huang, J.: MUP: A minimal unsatisfiability prover. In: Proceedings of the 10th Asia and South Pacific Design Automation Conference (ASPDAC'05), pp. 432–437 (2005)
13. Liffiton, M.H., Sakallah, K.A.: On finding all minimally unsatisfiable subformulas. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 173–186. Springer, Heidelberg (2005)
14. Bailey, J., Stuckey, P.J.: Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In: Hermenegildo, M.V., Cabeza, D. (eds.) PADL 2005. LNCS, vol. 3350, pp. 174–186. Springer, Heidelberg (2005)
15. Mneimneh, M.N., Lynce, I., Andraus, Z.S., Marques-Silva, J.P., Sakallah, K.A.: A branch and bound algorithm for extracting smallest minimal unsatisfiable formulas. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 393–399. Springer, Heidelberg (2005)
16. Zhang, J.M., Li, S.K., Shen, S.Y.: Extracting minimum unsatisfiable cores with a greedy genetic algorithm. In:Sattar, A., Kang, B.H. (eds.) AI 2006. LNCS, vol. 4304, pp. 847–856. Springer, Heidelberg (2006)
17. Liffiton, M.H., Sakallah, K.A.: Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. Journal of Automated Reasoning. 2008, 40(1): 1–30.
18. Liffiton, M.H., Mneimneh, M.N., Lynce, I., Andraus, Z.S., Marques-Silva, J.P., Sakallah, K.A.: A branch and bound algorithm for extracting smallest minimal unsatisfiable formulas. Constraints, 2009, 14(4) 415–442.
19. Gershman, R., Koifman, M., Strichman, O.: Deriving small unsatisfiable cores with dominator. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 109–122. Springer, Heidelberg (2006)
20. Gershman, R., Koifman, M., Strichman, O.: An approach for extracting a small unsatisfiable core. Formal Methods in System Design, 2008, 33(1): 1–27.
21. Dershowitz, N., Hanna, Z., Nadel, A.: A scalable algorithm for minimal unsatisfiable core extraction. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 36–41. Springer, Heidelberg (2006)
22. Gregoire, E., Mazuer, B., Piette, C.: Tracking MUSes and strict inconsistent covers. In: Proceedings of the the Sixth Conference on Formal Methods in Computer-Aided Design (FMCAD'06), pp. 39–46 (2006)
23. Gregoire, E., Mazuer, B., Piette, C.: Extracting MUSes. In: Proceedings of the 17th European Conference on Artificial Intelligence (EAI'06), pp. 387–391 (2006)
24. Gregoire, E., Mazuer, B., Piette, C.: Local-search extraction of MUSes. Constraints, 2007, 12(3): 325–344.
25. Gregoire, E., Mazuer, B., Piette, C.: Boosting a complete technique to find MSS and MUS thanks to a local search oracle. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence(IJCAI'07), pp. 2300–2305 (2007)
26. Gregoire, E., Mazuer, B., Piette, C.: Using local search to find MSSes and MUSes. European Journal of Operational Research, 2009, 199(3): 640–646.
27. Maaren H, Wieringa S.: Finding guaranteed MUSes fast. In: Buning HK, Zhao X, (eds.) SAT 2008. LNCS, vol. 4996, pp. 291–304. Springer, Heidelberg (2008)
28. Pitte, C., Hamadi, Y., Sais, L.: Efficient combination of decision procedures for MUS computation. In: Ghilardi, S., Sebastiani, R. (eds.) FroCos 2009, LNCS, Vol. 5749, pp. 335–349. Springer, Heidelberg (2009)

29. Nadel, A.: Boosting minimal unsatisfiable core extraction. In: Proceedings of the 10th International Conference on Formal Methods in Computer Aided Design (FM-CAD'10), pp. 221–229 (2010)
30. Barrett, C., Berezin, S.: CVC Lite: A New Implementation of the Cooperating Validity Checker. In: Alur, R., Peled, D.A. (eds.) CAV 2004, LNCS, Vol. 3114, pp. 515–518. Springer, Heidelberg (2004)
31. Bozzano M., Bruttomesso R., Cimatti A., Junttila T., van Rossum P., Schulz S., and Sebastiani R.: An incremental and Layered Procedure for the Satisfiability of Linear Arithmetic Logic. In: Halbwachs, N., Zuck, L. (eds.) TACAS 2005, LNCS, Vol. 3440, pp. 317–333. Springer, Heidelberg (2005)
32. Dutertre B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
33. Cimatti A, Griggio A, Sebastiani R. A simple and flexible way of computing small unsatisfiable cores in SAT modulo theories. In: Marques-Silva J, Sakallah K, (eds.) SAT 2007. LNCS, vol. 4501, pp. 334–339. Springer, Heidelberg (2007)
34. Nieuwenhuis R, Oliveras A, Tinelli C. Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann -Loveland procedure to DPLL(T). Journal of the ACM, 2006, 53(6): 937–977.
35. Barret C, Deters M, Oliveras A, et al.: http://www.smtcomp.org/2009/benchmarks.shtml, accessed in 2010.
36. Maric F, Janicic P.: Argo-lib: a generic platform for decision procedures. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004, LNCS, vol. 3097, pp. 231–217. Springer, Heidelberg (2004)

Jianmin Zhang was born in China in 1979. He is a lecturer in computer science at National University of Defense Technology. He received Ph. D. degree in computer science from National University of Defense Technology, Changsha, China, in 2008. His major fields of study include constraint satisfaction and formal verification of hardware.

Shengyu Shen was born in China in 1975. He is a associate professor in computer science at National University of Defense Technology. He received Ph. D. degree in computer science from National University of Defense Technology, Changsha, China, in 2005. His major fields of study include formal verification of hardware.

Jun Zhang was born in China in 1975. He is Ph. D. candidate in computer science, at National University of Defense Technology, Changsha, China, from 2007. His major fields of study include formal verification of VLSI chips.

Weixia Xu was born in China in 1963. He is a professor in computer science at National University of Defense Technology. His major fields of study include computer architecture and VLSI verification.

Sikun Li was born in China in 1941. He is a professor in computer science at National University of Defense Technology. His major fields of study include VLSI designing and verification.

Received: October 19, 2010; Accepted: March 12, 2011