Topic

Event

Author: Aklima Zaman
Supervision: Erika Ábrahám
RWTH Aachen University, LuFG Informatik 2

WS 16/17

Abstract

We present a procedure as published in [1] which computes all the solutions of non-linear equalities with virtual substitution.The solution approach relies on some substitution rules of virtual substitution. The rules can be applied only on the real-arithmetic formulas which is linear or quadratic.

1 Introduction

Quantifier/ variable elimination for elementary real algebra is a fundamental problem. This problem can be solved easily by using virtual solution. In 1993, the concept of virtual substitution was first introduced. Initially it was a procedure to eliminate quantifier/variable elimination for linear real arithmetic formulas.Further, virtual substitution became a procedure of quantifier elimination for non-linear arithmetic formulas.Virtual substitution cannot eliminate quantified variables whose degree is higher than 2.

Section 2 consists some preliminaries with some definitions. How to construct the real zeros is explained in section 3. In the next section we will know some substitution rules by which we can eliminate variables from a formula with an example. In the section 4 an idea to eliminate quantifiers is explained by an example and conclude in the last section.

2 Preliminaries

2.1 CNF

A formula is said to be in CNF if and only if it is a conjunction of clauses adn a clause is a disjunction of literals. A literal could be a positive or negative instance of Boolean variable. A CNF formula φ is defined as follows,

$$page3$$

here, C_i is the i^{th} clause and n is the number of total clauses in φ, a_{ij} is the literal and m is the total number of literals in C_i.

Example2: Assume the following CNF formula,

$$clauseMarkKorteHobe\varphi = (x) \wedge (\neg x) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y)$$

where, $x, \neg x, y$ and $\neg y$ are the literals.

The aforementioned CNF formula will ne used as an example throughout this paper which is unsatisfiable.

2.2 Satisfiability Checking

Satisfiability checking determines whether the existentially quantified first-order-logic formulas are satisfiable by generating solutions automatically. The formulas are Boolean combinations of theory constraints and the form of theory constraints varies depending on with which theory we want to represent first-order-logic. Some example theory constraints from different theories are given in [2].By sat solving we can check the satisfiability of formulas, though SAT-modulo-theories (SMT) solvers are available and popular theories.

A SAT solver solves the Boolean satisfiability problem by implementing decision procedure. The DPLL algorithm is the basis for most modern SAT solvers. The input formula of it is expected to be in conjunctive normal form (CNF).

Example1: Let us consider a CNF $(x) \wedge (\neg x \vee \neg y) \wedge (z)$. First x is set to $true$. Boolean constraint propagation (BCP) indicates that y of the second clause must be $false$ to satisfy the CNF. There is still a unassigned variables. A new decision will be made and z is set to $true$. As all variables are assigned and there is no conflict, the satisfying soluiton is $x = 1, y = 0$ and $z = 1$.

SMT solvers can be applied on quantifier-free first-order-logic formulas with an underlying theory to check the satisfiability. There are two type of SMT solvers, Eager SMT solver and Lazy SMT solver. To know about SMT in details you can read [3].

2.3 Clause-Selector Variables

Authors of the paper [1] has used clause-selector variable by augmenting a CNF formula. Clause-selector variable is a variable w_i which is negated to augment each clause of C_i of a CNF such that $C'_i = (\neg w_i) \vee C_i$ in a new formula φ'. Notice that each C'_i is an implication, $C'_i = (\neg w_i \rightarrow C_i)$. It menas if w_i is set to $true$, the original clause is being enabled. Conversely, assigning w_i $false$ means C_i is being disabled or removed from the set of constraints, because C'_i is satisfied by the assignment to w_i. So, adding clause-selector variables provides the SAT solver the ability to enable and disable constraints as a part of its normal search.

For our example formula, after adding these clause-selector variables we get the following augmented formula φ',

$$\varphi' = (\neg w_1 \vee x) \wedge (\neg w_2 \vee \neg x) \wedge (\neg w_3 \vee \neg x \vee y) \wedge (\neg w_4 \vee \neg x \vee \neg y)$$

2.4 Minimal Unsatisfiable Subset and Minimal Correction Subset

A Minimal Unsatisfiable Subset (MUS) ia a subset of the clauses of an unsatisfiable CNF formula φ that is both unsatisfiable and minimal. A Minimal Correction Subset (MCS) is a subset of the clauses of φ whose removal from φ make it satisfiable and which is minimal. A formula can have multiple MUSes and MCSes. The formal definitionss of the set of MUSes and MCSes of φ is given following:

Definition1: A subset U be a MUS which is a subset of φ if U is unsatisfiable and for each clause C_i containing in U the removal of C_i from U is satisfiable.

$$khataThekeLikhteHobe$$

Defintion2: A subset M be a MCS which is a subset of φ if the removal of M from φ is satisfiable and for each clause C_i of M the removal of M from φ is unsatisfiable where M does not have C_i.

$$khataThekeLikhteHobe$$

For our example formula $\varphi = (x) \wedge (\neg x) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y)$ the MUSes and MCSes are shown in the following table:

$$tableAkteHobe$$

2.5 Hitting Sets

Assume a collection Ω of sets of a finite set D. A hitting set of Ω is a subset of D such that it contains at least one element from each subset in Ω. Formally, the definition is given below:

Definition3: A hitting set H of Ω is $H \subseteq D$ such that $\forall S \in \Omega$, $H \cap S \neq 0$ where S represents each subset of Ω.

In the paper [1], authors refer to minimal hitting sets which are the hitting sets from which no elements cannot be removed and if at least one element is removed, it will not be a hitting set anymore.

Example: Let us consider, $\Omega = \{\{a, b\}, \{b, c, d\}\}$. Then, $\{a, b\}, \{b, c, d\}, \{a, c, d\}, \{b\}, \cdots$ are the hitting sets where, $\{b\}$ is the only minimal hitting set.

2.6 MUS backSlashDiteHobe MCS Duality

There is a relationship between MUS and MCS which is the foundation of paper [1]. The relationship states that the set of MUSes of formula φ is equal to theset of minimal and irreducible hitting sets of the set of MCSes and vice-versa. This is the duality of MUS and MCS. Fomally it can said that:

1. A subset U of φ is an MUS if and only if U is an irreducible hitting set of MCSes(φ).
2. A subset M of φ is an MUS if and only if M is an irreducible hitting set of MUSes(φ).

$$dualityPicAkteHobe$$

To show duality we use our example formula $\varphi = (x) \wedge (\neg x) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y)$. First, take the set MCSes of φ and it is $\{\{C_1\}, \{C_2, C_3\}, \{C_2, C_4\}\}$. For this, the set of minimal and irreducible hitting sets is $\{\{C_1, C_2\}, \{C_1, C_{3,} C_4\}\}$ which is exactly like the set of MUSes (given in Table *blabla*). Similarly, we can show that the set of MCSes is the set of irreducible hitting sets of the set of MUSes. So, every MCS has at least one clause from every MUS and vice-versa. The duality is illustrated in Figure(blabla).

2.7 AtMost Constraints

AtMost Constraint is a type of clause that can be generated from many types of clauses. If $\{l_1, l_2, l_3, \cdots, l_\mathrm{n}\}$ is a set of n literals and k is a positive integer, an AtMost constraint is defines as,

$$AtMosterFormulaLikhteHobe$$

where, $val(l_\mathrm{i})$ is 1 if l_i is assigned *true* and otherwise 0. k is a bound which tells that maximum how many literals can be assigned *true*. Authors used an implementation of the AtMost constraint where the variables are watched and propagates the negation of each literal when k of them are assigned *true*. This is implemented with exactly n watched literals and a counter that is incremented or decremented whenever on of them is assigned or unassigned.

3 Algorithms for Computing Minimal Unsatisfiable Subset

The approach of computing all MUSes of φ is to first find all MCSes(φ) and then to compute all irreducible hitting sets of MCSes(φ), which are all MUSes(φ). So, there are two phases to generate all MUSes of a unsatisfiable CNF formula. The first phase is to compute MCSes by using an algorithm and the second phase is to compute MUSes from the MCSes by using a recursive algorithm which authors have developed to compute irreducible hitting sets.
In the first phase, authors use a SAT solver (produces an satisfying assignment, if exists) to work with the input given and provide hitting sets of MUSes without revealing the underlying MUSes. For the second phase, they get all the information with MCSes and they use an recursive algorithm to generate minimal and irreducible hitting sets.

$$algorithmMCSLikhteHobe$$

3.1 Computing MCSes

In Algorithm 1, there is a method named **MCSes** which takes a CNF formula φ as input. The goal of this is to find minimal set of clauses whose removal makes φ satisfiable. In Line 1, φ is augmented with clause-selector variables by creating φ'. The augmentation increases the number of variables. Also the search space grows for finding a satisfying assignment. Initially, the variable $MCSes$ is empty and at the end all MCSes is going to be stored in this variable. There is a counter variable K which is initialized by 1.
Each iteration of the outer while loop (lines 4-12) finds an MCS of size k, which is incremented by 1 after each iteration. In Line 5, a clause is added of the form $atMostFormula$♣♣ to φ' and a new formula φ'_{k} is created. K number of variables w_i are assigned to $false$ so that K number of clauses are disabled. The set of variables w_i are assigned to $false$ indicates the clauses as an MCS.
The inner while loop (lines 6-10) searches all the satisfiable assignments of φ'_{k} and also finds all MCSes of size k. In line 6, IncrementalSAT(φ'_{k}) method is called which uses MiniSAT (a modern open-source SAT solver which has incremental solving ability) for finding a solution of φ'_{k}. In this method the solving function is invoked multiple times, but each time with a different set of assumption literals and then the solver checks the satisfiability of all the clauses provided with the current assumptions only [4]. Each satisfying assignment produces an MCS stored in a variable $newMCS$ which contains the clauses having w_{i} assigned $false$. Each $newMCS$ is recorded in $MCSes$ (Line 7). In Line 8 and 9 a blocking clause is added to φ'_{k} and φ', respectively to block that solution. Blocking clause is a clause containing the clause-selector variables which consist in the clauses of that MCS. It means at least one of the clauses of MCS will be enabled for future solution. For example, an MCS has the clauses C_2 and C_3 which means w_2 and w_3 are assigned false in satisfying assignment. Then the blocking clause will be $(w_2 \vee w_3)$. $True$ is assigned to at least one w_{i} excluding the MCS and any of its supersets from any future solution.
All MCSes founded are irreducible as the MCSes were found in increasing size and all supersets are excluded form the future solutions. After evaluating all the solutions of with bound K enforces to find solutions with bound $K+1$. Also, $K+1$ disabled clause to be irreducible as potential subsets would have found and blocked.
The outer while loop checks if φ' is still satisfiable without any bound on w_{i}. Note that, φ' is augmented with all collected blocking clauses. Finding no satisfying assignments indicates that we have found all MCSes and there is no other way to remove clauses for generating satisfying formula. Finally, the algorithm terminates and method $MCSes$ con-

taining all MCSes to caller.
Example♣♣:
Let us consider, our example formula $\varphi = (x) \wedge (\neg x) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y)$. So, $\varphi' = (\neg w_1 \vee x) \wedge (\neg w_2 \vee \neg x) \wedge (\neg w_3 \vee \neg x \vee y) \wedge (\neg w_4 \vee \neg x \vee \neg y)$. Initially, $MCSes = \emptyset$ and $k = 1$.
In the first iteration, it will add an AtMost bound with $K = 1$ by creating $\varphi'_1 = \varphi' \wedge AtMost$♣♣. At first $false$ is assigned to w_1 and the incremental solver find a solution for φ'_{k}. Now, we get the followings:

$$MCSes = \{C_1\}$$

$$\varphi'_1 = \varphi'_1 \wedge w_1$$
$$\varphi' = \varphi' \wedge w_1$$

After adding the blocking clause, (w_1) the incremental solver will be unable to any further solution with $K = 1$.There is no other way to remove one clause to satisfy φ. It means no single clause covers all of its MUSes. The inner while loop is exited
As φ' is still satisfiable, the bound is incremented to 2 and the search for soluitons of $\varphi'_2 = \varphi' \wedge AtMost$♣♣ is continued.For $K = 2$, we get the followings, respectively: $bulletDiteHobe$ $w_2 = false$ and $w_3 = false$

$$MCSes = \{\{C_1\}, \{C_2, C_3\}\}$$

$$\varphi'_2 = \varphi'_2 \wedge (w_2 \vee w_3)$$
$$\varphi' = \varphi' \wedge (w_2 \vee w_3)$$

$bulletDiteHobe$ $w_2 = false$ and $w_4 = false$

$$MCSes = \{\{C_1\}, \{C_2, C_3\}, \{C_2, C_4\}\}$$

$$\varphi'_2 = \varphi'_2 \wedge (w_2 \vee w_4)$$
$$\varphi' = \varphi' \wedge (w_2 \vee w_4)$$

We will not get any other MCSes for $K = 2$. Also, the outer while loop will exit as φ' with all the blocking clauses added is no longer satisfiable. So, final MCSes are $\{C_1\}, \{C_2, C_3\}$ and $\{C_2, C_4\}$.

$$tableAkteHobe$$

3.2 Compuitng MUSes

Once the entire collection of MCSes has been computed, the second phase produces all MUSes of the given instances and this phase is independent of first phase as it does not depend on the semantics of the input of Algorithm 1. The second phase can be applied to any minimal hitting set. Authors presented a recursive algorithm which takes two inputs, one is set of MCSes and another the MUS currently being constructed in each branch of the recursion. The outputs are all MUSes.

$$algorithmSMUSLikhteHobe$$

Algorithm 2 is the recursive algorithm where the inputs are $\boldsymbol{MCSes}$ and $\boldsymbol{currentMUS}$. The goal of this is to find a irreducible set of clauses that hits each set of $\boldsymbol{MCSes}$. The terminating condition of the recursion is on Line 1 that is if $\boldsymbol{MCSes}$ is empty. If it is,prints

the MUS generated in current recursion branch and returns to look for other MUSes in another branches on Line 2 and in Line 3,respectively. Lines $5 - 12$ is iterated through all possible choices of clause selected on line 5 and recorded in a variable, $\boldsymbol{selClause}$. In Line 6, $\boldsymbol{selClause}$ is stored in a growing MUS, $\boldsymbol{newMUS}$. For all possible choices of MCSes (selected on line 7) recorded in $\boldsymbol{selMCS}$ such that $\boldsymbol{selClause}$ consists in $\boldsymbol{selMCS}$, iteration occurs from Lines $7 - 11$. Then, $\boldsymbol{MCSes}$ is recorded in a variable, $\boldsymbol{newMCSes}$.
For every $\boldsymbol{selMCS}$ a method, **PropagateChoice** (Algorithm 3) is called with $\boldsymbol{newMCSes}$, $\boldsymbol{selClause}$ and $\boldsymbol{selMCS}$. Algorithm 3 has the goal to alter a copy of the current MCSes, $\boldsymbol{newMCSes}$. It alters by removing the MCSes from $\boldsymbol{MCSes}$ hit by $\boldsymbol{thisClause}$. In the last line **MaintainNoSupersets** is invoked which removes any MCS from $\boldsymbol{MCSes}$ which is a superset of others. This is needed as we are dealing with the minimal set of MCSes and no MCS cannot be a superset of others. For example, let we have $MCSes = \{\{C_2\}\{C_1, C_3\}\{C_2, C_4\}\}$. Here, $\{C_2, C_4\}$ is a superset of $\{C_2\}$. If we pass this MCSes through **MaintainNoSupersets**, $\{C_2, C_4\}$ will be removed.
The recursion of Algorithm 2 continues by calling **AllMUSes** with $\boldsymbol{newMCSes}$ and $\boldsymbol{newMUS}$. Example♣♣:
Let us consider, our example formula φ. We already have found $MCSes(\varphi) = \{\{C_1\}, \{C_2, C_3\}, \{C_2, C_4\}\}$. Let, C_1 is chosen as $\boldsymbol{selClause}$ in the first iteration. So, C_1 is added in the growing MUS. Now, for each set of MCSes another iteration will occur which calls **PropagateChoice** method by creating a modified copy of current MCSes and makes a recursive call to itself. Modified MCSes contain $\{C_2, C_3\}\{C_2, C_4\}$. The recursive call continues until empty MCSes is not found. Empty MCSes is found by producing a MUS, $\{C_2, C_3\}$. Now, it will look for other MUSes in other branches. For our example the whole procedure to generate all MUSes is illustrated in the Figure 1. Finally we get all MUSes, $\{C_1, C_2\}$ and $\{C_1, C_3, C_4\}$.

4 Quantifier Elimination with the Virtual Substitution

5 Conclusion

We have described the solving technique of non-linear equalities with virtual substitution step by step. It has two steps. One is to construct test candidates with side conditions and another one is to replace a variable by a test candidate in a formula. It has already said that the replacement is based on some substitution rules.Finally, we can have the solutions for which the formula is satisfied.

References

[1] V. Weispfenning, *Quantifier elimination for real algebra - the quadratic case and beyond.* Appl. Algebra Eng. Commun. Comput, 1997.

[2] R. Loss, V. Weispfenning, *Applying linear quantifier elimination.* The computer Journal 36 (1993), pp. 450-462.