Noname manuscript No.
(will be inserted by the editor)

NP-completeness of small conflict set generation for congruence closure

Andreas Fellner · Pascal Fontaine · Bruno Woltzenlogel Paleo

the date of receipt and acceptance should be inserted later

Abstract The efficiency of Satisfiability Modulo Theories (SMT) solvers is dependent on the capability of theory reasoners to provide small conflict sets, i.e. small unsatisfiable subsets from unsatisfiable sets of literals. Decision procedures for uninterpreted symbols (i.e. congruence closure algorithms) date back from the very early days of SMT. Nevertheless, to our best knowledge, the complexity of generating smallest conflict sets for sets of literals with uninterpreted symbols and equalities had not yet been determined, although the corresponding decision problem was believed to be NP-complete. We provide here an NP-hardness proof, using a simple reduction from SAT.[1]

Keywords Satisfiability Modulo Thories · Decision procedures · Congruence closure · Complexity

1 Introduction

Satisfiability Modulo Theory solvers are nowadays based on a cooperation between a propositional satisfiability (SAT) solver and a theory reasoner for

This work has been partially supported by the project ANR-13-IS02-0001 of the Agence Nationale de la Recherche, by the H2020-FETOPEN-2016-2017-CSA project SC^2 (712689), by an AW APART Stipendium, by FWF S11407-N23 (RiSE/SHiNE), and by ERC Start Grant (279307: Graph Games).

A. Fellner
Austrian Institute of Technology and Vienna University of Technology (Austria)

P. Fontaine
Inria, Loria, U. of Lorraine (France)

B. Woltzenlogel Paleo
Vienna University of Technology (Austria) and Australian National University (Australia)

[1] Note to the reviewers: this paper was first a short note at the SMT Workshop 2015 [5], with no formal proceedings. Besides various small improvements, we made the paper more self-contained.

the combination of theories supported by the SMT solver. The propositional structure of the problem is handled by the SAT solver, whereas the theory reasoner only has to deal with conjunctions of literals. Very schematically (we refer to [1] for more details) the Boolean abstraction of the SMT problem is repeatedly refined by adding theory conflict clauses that eliminate spurious models of the abstraction, until either unsatisfiability is reached, or a model of the SMT formula is found. Refinements can be done by refuting models of the propositional abstraction one at a time. It is, however, much more productive to refute all propositional models that are spurious for the same reason. A model of the abstraction is spurious if the set of concrete literals corresponding to the abstracted literals satisfied by this model is unsatisfiable modulo the theory. Given such an unsatisfiable set of concrete literals, the disjunction of the negations of any unsatisfiable subset (a.k.a. *core*) is a suitable conflict clause. By backtracking and asserting the conflict clause, the SAT-solver is prevented from generating the spurious model again. The smaller the clause, the stronger it is and the more spurious models it prevents. Therefore, an optimal conflict clause, corresponding to a minimal unsatisfiable subset of literals (i.e. such that all its proper subsets are satisfiable) or even a minimum one (i.e. smallest among the minimals) is desirable. This feature of the theory reasoners to *generate small conflict sets* (a name adopted in [1]) from their input is also referred to as *proof production* [9,10] or *explanation generation* [11].

Decision procedures for the theory of uninterpreted symbols and equality can be based on congruence closure [8,4,11]. The decision problem is polynomial and even quasi-linear [4] with respect to the number of terms and literals in the input set. Producing minimal conflict sets also takes polynomial time. Indeed, testing if a set S remains unsatisfiable after removal of one of its literals is also polynomial. It suffices then to repeatedly test the $|S|$ literals of S to check if they can be removed. The set S pruned of its unnecessary literals is minimal. One could also profit from the incrementality of the decision procedure [7].

It has also been common knowledge that computing minimum conflict clauses for the theory of uninterpreted symbols and equality is a difficult problem. But, to our best knowledge, the complexity of finding the smallest conflict clause generation for sets of literals with uninterpreted symbols and equalities has never been established. The complexity of the corresponding decision problem (i.e. of whether there exists a conflict clause with size smaller than a given k) is mentioned to be NP-complete in [11] — with a reference to a private communication with Ashish Tiwari — but neither the authors of [11] nor Ashish Tiwari published a written proof of this fact[2].

Our interest in this problem arose from our work on Skeptik [2], a tool for the compression of proofs generated by SAT and SMT solvers. For the sake of moving beyond the purely propositional level, we have developed an algorithm for compressing congruence closure proofs, which consists of regenerating (possibly smaller) congruence closure conflict clauses while traversing the proof.

[2] We contacted both Ashish Tiwari and the authors of [11], who confirmed this.

Congruence closure conflict clauses are typically generated from paths in the *congruence graph* maintained by the congruence closure algorithm [6,11,10]. During the replay, we use a polynomial-time algorithm for searching for short paths in the congruence graph, which is a modification of Dijkstra's shortest path algorithm [3]. This raised the question whether our algorithm could find the shortest conflict clauses, as Dijkstra's algorithm finds shortest paths. We answered this question negatively by proving that the problem of deciding whether a shorter conflict clause exists is NP-hard. The goal of this short paper is to present this proof. A preliminary version was presented at the SMT Workshop 2015 [5].

2 Preliminaries

We assume knowledge of propositional logic and quantifier-free first-order logic with equality and uninterpreted symbols, and only enumerate the notions and notations used in this paper. A literal is either a propositional variable or the negation of a propositional variable. A clause is a disjunctive set of literals. A propositional variable x appears positively (negatively) in a clause C if $x \in C$ (resp. $\neg x \in C$). The notations $\{\ell_1, \dots \ell_n\}$ and $\ell_1 \vee \dots \vee \ell_n$ will be used interchangeably. A clause is tautological if and only if it contains a variable both positively and negatively. We shall tacitly assume that clauses are non-tautological, except when explicitly stated otherwise. Clauses being sets, they cannot contain multiple occurrences of the same literal. A formula in conjunctive normal form (CNF for short) is a conjunctive set of clauses. A total (partial) assignment $\mathcal{I}$ for a formula in propositional logic assigns a value in $\{\top, \bot\}$ to each (resp. some) propositional variable(s) in the formula. An assignment $\mathcal{I}$ for a formula F is a model of F, denoted $\mathcal{I} \models F$, if it makes the formula F true. A formula is satisfiable if it has a model, it is unsatisfiable otherwise. A total or partial assignment is perfectly defined by the set of literals it makes true. By default, an assignment is total unless explicitly said to be partial. A set of formulas E entails a (set of) formula(s) E', denoted $E \models E'$, if every model of E is a model of E'.

We now define the necessary notions for quantifier-free first-order logic.

Definition 1 (Terms and equations) A *signature* Σ is a finite set of function symbols $\mathcal{F}$ equipped with an *arity* function $\mathcal{F} \to \mathbb{N}$. A *constant* is a function with null arity. A *unary* function has arity one. Given a signature Σ, the set of *terms* $\mathcal{T}^\Sigma$ is the smallest set containing all constants in $\mathcal{F}$ and all terms of the form $g(t_1, \dots, t_n)$, where g is a function symbol of arity n in $\mathcal{F}$ and $t_1, \dots, t_n$ are terms in $\mathcal{T}^\Sigma$. An *equation* between two terms s, t in $\mathcal{T}^\Sigma$ is noted as $s = t$.

Signatures commonly include predicate symbols. Everything extends smoothly to signatures with predicates, but to simplify, a quantifier-free first-order logic formula is here just a Boolean combination of equalities between terms; a literal is either an equation or the negation of an equation.

The terms $t_1, \ldots, t_n$ are *direct subterms* of $g(t_1, \ldots, t_n)$. The *subterm* relation is the reflexive and transitive closure of the direct subterm relation.

An assignment $\mathcal{I}$ on some signature maps each constant to an element in a universe $\mathcal{U}$, and each function symbol to a function of appropriate arity on $\mathcal{U}$. By extension, it assigns an element in $\mathcal{U}$ to every term, and a value to every equation $s = t$, namely $\top$ if $\mathcal{I}(s) = \mathcal{I}(t)$ and $\bot$ otherwise. Like in propositional logic, an assignment on some signature thus gives a truth value to every formula on this signature.

Definition 2 (Congruence relation) Given a set of terms $\mathcal{T}$ closed under the subterm relation, a relation $R \subseteq \mathcal{T} \times \mathcal{T}$ is a congruence if it is

- reflexive: $(t, t) \in R$ for each $t \in \mathcal{T}$;
- symmetric: $(s, t) \in R$ if $(t, s) \in R$;
- transitive: $(r, t) \in R$ if $(r, s) \in R$ and $(s, t) \in R$;
- compatible: $(g(t_1, \ldots, t_n), g(s_1, \ldots, s_n)) \in R$ if g is a n-ary function symbol and $(t_i, s_i) \in R$ for all $i = 1, \ldots, n$.

A congruence relation is also an equivalence relation, since it is reflexive, transitive and symmetric. Therefore a congruence relation partitions its underlying set of terms $\mathcal{T}$ into congruence classes, such that two terms (s, t) belong to the same class if and only if $(s, t) \in R$. The relations $\emptyset$ and $\mathcal{T} \times \mathcal{T}$ are trivial congruence relations. An assignment $\mathcal{I}$ on a signature Σ defines a congruence relation on any subset $\mathcal{T} \subseteq \mathcal{T}^\Sigma$, that is, $R = \{(s, t) \mid \mathcal{I}(s = t) = \top\}$.

An equation $s = t$ on terms in a set $\mathcal{T}$ can be seen as a singleton relation $\{(s, t)\} \subseteq \mathcal{T} \times \mathcal{T}$. By extension, a set of equations can also be seen as a relation, i.e., the union of the singleton relations.

Definition 3 (Congruence closure) The congruence closure E^* of a set of equations E on a set of terms $\mathcal{T}$ closed under the subterm relation is the smallest congruence relation on $\mathcal{T}$ containing E.

Since congruence relations are closed under intersection, the congruence closure of a set of equations always exists. Also notice that, if $(s, t) \in E^*$, then $E \models s = t$. We say that E is an *explanation* for $s = t$.

An algorithm computing the congruence closure of a relation is also a decision procedure for the problem of satisfiability of sets of equalities and disequalities in quantifier-free first-order logic with uninterpreted (predicates and) functions. It suffices indeed to compute the congruence closure of all equalities on the terms and subterms occurring in the literals. Then, the set of literals is satisfiable if and only if there is no disequality with both terms in the same class. A model can be built from the congruence closure, on a universe with cardinality equal to the number of classes in the congruence.

3 Congruence Closure in Practice

The algorithms we consider in the following take as input a set of literals E. Considering complexity, not only the cardinality of the set is important,

but also the number of terms and subterms as well as the number of their occurrences. Congruence closure algorithms in modern SMT solvers typically represent terms with Directed Acyclic Graphs (DAGs) using maximal sharing, and not trees. The number of term and subterm occurrences does not matter, but only the number of distinct (sub)terms. The input is also typically not a set, but successive calls to an assertion function with a literal as argument: every repetition of the same literal then matters for complexity. Let us assume, however, that the input is a set E, terms are DAGs with maximal sharing (identity of two terms can be checked in constant time), and identity of two function symbols can be checked in constant time. Therefore, we characterize complexity results in terms of number of literals, terms and subterms of the input set, i.e. $|E|$ and $|\mathcal{T}(E)|$.

Since congruence relations are basically partitions of equivalent terms that additionally satisfy the compatibility property, it is unsurprising that practical congruence closure algorithms, or decision procedures for ground sets of first-order logic literals, are based on some kind of union-find data-structure. Terms (and subterms) are put into equivalence classes, according to the equalities in the input. The algorithms furthermore check, every time two classes of the partition are merged, whether any new equality induced by compatibility has to be taken into account. Also, it checks that the congruence is consistent with the set of disequalities. We refer the reader to [8,4,11] for more details. The complexity of those algorithms depend on the internal data-structures and on the representation of terms [4]. Algorithms typically implemented in SMT solvers have complexity $\mathcal{O}(|E| + |\mathcal{T}(E)|\ \log|\mathcal{T}(E)|)$ on average, a hash table being used to detect new equalities induced by compatibility.

The generation of conflict sets or explanations is based on the congruence graph: its nodes are the terms and subterms considered by the algorithm. An edge in the graph is either a full edge, linking two nodes s and t and labeled by an input equality $s = t$, or a congruence edge (a dotted edge in the figures in this paper), linking two terms with the same leading function symbol and labeled by the compatibility-deduced equality between both terms. The graph has a path between two terms if and only if they belong to the same congruence class. The equality between two terms in the same class is a logic consequence of the set of equalities labeling the path. To get an explanation for the equality of two terms in the same class, that is, a set of input equalities implying the equality of the two terms, it thus suffices to collect the set of equalities labeling a path, and recursively replace any compatibility equality $g(t_1, \ldots, t_n) = g(s_1, \ldots, s_n)$ by the explanations of $t_1 = s_1, \ldots, t_n = s_n$.

Practical congruence closure algorithms with explanation build a congruence graph while computing the congruence closure. Every time the decision procedure merges two classes, either because of an input equality or because an equality was deduced due to compatibility, a full- or congruence- edge is added to the graph. Since edges between nodes are only added when their respective congruence classes are merged, the path between two terms in the same class is unique. The explanation that two terms are equal is also unique, but there is no guarantee that this explanation is the smallest one. Indeed,

it may happen that the algorithm considers, e.g. equations $a = b$ and $b = c$ before $a = c$, merging a, b and c before getting the last equality, and thus discarding $a = c$ as redundant: in that case, $a = c$ would have been the smallest proof that a and c are equal, but the congruence graph would only consider the two other equalities. There is not even a guarantee that the explanation is minimal. Again, the congruence closure algorithm can prove that $a = f(b)$ from the input equalities $b = f(a)$, $a = b$ and $f(a) = f(b)$, but the redundant equality $f(a) = f(b)$ would only be recorded in the congruence graph if it were considered before $a = b$.

In practice, the congruence closure procedures implemented in SMT solvers produce explanations efficiently: the complexity of the explanation production is linear with respect to the explanation size, which is at most equal to the number of input equalities. But the explanations are not optimal, i.e. they are not always the smallest or even minimal. It is possible to compute minimal explanations in polynomial time; it suffices for instance to compute again the congruence closure successively removing every equality in the explanation, to see if it is redundant or not. One could hope to conceive a different congruence closure algorithm generating the smallest explanation in polynomial time, for instance on the basis of shortest path algorithms applied to congruence graphs containing even redundant equalities. But, the next section proves that the corresponding decision problem is NP-hard.

4 NP-Completeness of the Small Conflict Set Problem

The function problem of *generating* the smallest conflict set corresponds to the decision problem of *deciding* whether a conflict set with size smaller than a given k exists.

Definition 4 (Small conflict set problem) Given an unsatisfiable set E of literals in quantifier-free first-order logic with equality and $k \in \mathbb{N}$, the *small conflict set generation problem* is the problem of deciding whether there exists an unsatisfiable set $E' \subseteq E$ with $|E'| \leq k$.

If we had a polynomial-time algorithm α capable of generating the smallest conflict set for any unsatisfiable set E, then we could decide in polynomial-time any instance of the small conflict set problem by applying α to E and checking whether $\alpha(E)$ has size smaller than k. But, as proven below, the small conflict set problem is NP-complete and, therefore, polynomial-time generation of conflict sets with minimum size is not possible (unless P = NP). Our proof reduces the problem of deciding the satisfiability of a propositional logic formula in conjunctive normal form (SAT) to the small explanation problem.

Definition 5 (Small explanation problem) Given a set of equations $E = \{s_1 = t_1, \ldots s_n = t_n\}$, $k \in \mathbb{N}$ and a target equation $s = t$, the *small explanation problem* is the problem of answering whether there exists a set E' such that $E' \subseteq E$, $E' \models s = t$ and $|E'| \leq k$.

The small explanation problem and the small conflict set problem are closely related: there is a small explanation of size k of $s = t$ from E if and only if there is a small conflict of size $k+1$ for $E \cup \{s \neq t\}$. The proof of hardness is based on a (polynomial) translation of the propositional satisfiability problem to the small explanation problem.

The translation of the propositional satisfiability problem consists of two parts: a translation of propositional formulas, here assumed, without loss of generality, to be in CNF (as shown in Definition 6), and a translation of assignments (as shown in Definition 7).

Definition 6 (CNF congruence translation) Let $\mathcal{C}$ be a set of propositional clauses $\{C_1, \dots C_n\}$ using variables $x_1, \dots, x_m$. The *congruence translation* $E_{\mathcal{C}}$ of $\mathcal{C}$ is defined as the set of equations

$$E_{\mathcal{C}} = \mathit{Connect} \cup \bigcup_{1 \leq i \leq n} \mathit{Clause}_i$$

with

$$\begin{aligned} \mathit{Connect} &= \{c'_i = c_{i+1} \mid 1 \leq i < n\} \\ \mathit{Clause}_i &= \{c_i = t_i(\hat{x}_j) \mid x_j \text{ appears in } C_i\} \\ &\quad \cup \{t_i(\top_j) = c'_i \mid x_j \text{ appears positively in } C_i\} \\ &\quad \cup \{t_i(\bot_j) = c'_i \mid x_j \text{ appears negatively in } C_i\} \end{aligned}$$

where $c_1, \dots c_n, c'_1, \dots c'_n, \hat{x}_1, \dots \hat{x}_m, \top_1, \dots \top_m, \bot_1, \dots \bot_m$ are distinct constants, and $t_1, \dots t_n$ are distinct unary functions.[3]

Remark 1 Note that the constants $\top_i$ and $\bot_i$ (for $1 \leq i \leq m$)) should not be confused with the Boolean values $\top$ and $\bot$. The intuitive relationship between these constants and the boolean values is established in Definition 7.

This translation is illustrated by the following example.

Example 1 Consider the set of clauses $\mathcal{C}$

$$\big\{C_1 = x_1 \vee x_2 \vee \neg x_3, C_2 = \neg x_2 \vee x_3, C_3 = \neg x_1 \vee \neg x_2\big\}.$$

Figure 1 represents the congruence translation of $\mathcal{C}$ graphically, an edge between two nodes meaning that the set contains an equation between the terms labeling the two nodes.

Definition 7 (Assignment congruence translation) The translation $E_{\mathcal{I}}$ of an assignment $\mathcal{I}$ on propositional variables $x_1, \dots, x_m$ is the set of equations

$$\begin{aligned} E_{\mathcal{I}} = \quad & \{\hat{x}_j = \top_j \mid 1 \leq j \leq m \text{ and } \mathcal{I} \models x_j\} \\ & \cup \{\hat{x}_j = \bot_j \mid 1 \leq j \leq m \text{ and } \mathcal{I} \models \neg x_j\} \end{aligned}$$

For convenience, we also define the set

$$\mathit{Assignment}^\star = \{\hat{x}_j = \top_j, \hat{x}_j = \bot_j \mid 1 \leq j \leq m\}.$$

[3] It would be possible to define a translation without the c'_i constants, but they ease the presentation.

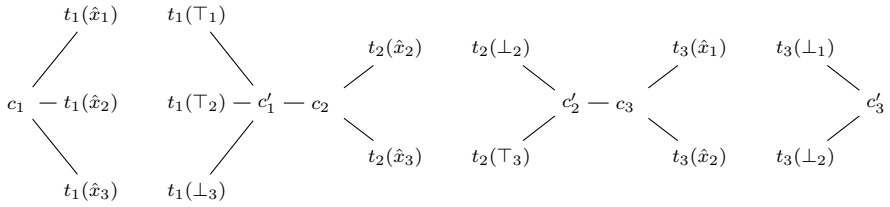

Fig. 1 The congruence translation $E_{\mathcal{C}} = \mathit{Connect} \cup \bigcup_{1 \leq i \leq n} \mathit{Clause}_i$ of $\mathcal{C}$.

An assignment congruence translation is always a subset of *Assignment**. By extension, a subset of *Assignment** is said to be an assignment if it is the congruence translation of an assignment, that is, if it does not contain both $\hat{x}_j = \top_j$ and $\hat{x}_j = \bot_j$ for some j.

Example 2 (Example 1 continued) Consider the model $\mathcal{I} = \{x_1, \neg x_2, x_3\}$ of $\mathcal{C}$. Figure 2 gives a graphical representation of $E_{\mathcal{I}}$, whereas *Assignment** is represented in Figure 3. Notice that $E_{\mathcal{C}} \cup E_{\mathcal{I}} \models c_1 = c'_3$, and c_1 and c'_3 are connected in the congruence graph of $E_{\mathcal{C}} \cup E_{\mathcal{I}}$ (Figure 4), the path containing both full edges corresponding to equalities in $E_{\mathcal{C}} \cup E_{\mathcal{I}}$, and dotted edged corresponding to equalities due to the compatibility property of the congruence relation.

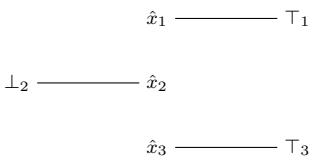

Fig. 2 Congruence translation of $\mathcal{I}$

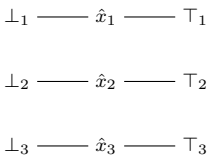

Fig. 3 *Assignment**

Lemma 1 *Consider a (partial or total) assignment $\mathcal{I}$ for non-tautological clauses $\mathcal{C} = \{C_1, \ldots C_n\}$. Then $\mathcal{I} \models \mathcal{C}$ if and only if $E_{\mathcal{I}} \cup E_{\mathcal{C}} \models c_1 = c'_n$.*

Proof. Let the propositional variables in $\mathcal{C}$ be $x_1, \ldots, x_m$.

($\Leftarrow$) Consider the congruence graph induced by $E_{\mathcal{I}} \cup E_{\mathcal{C}}$. Besides edges directly associated to equalities in the set, the only edges are congruence edges

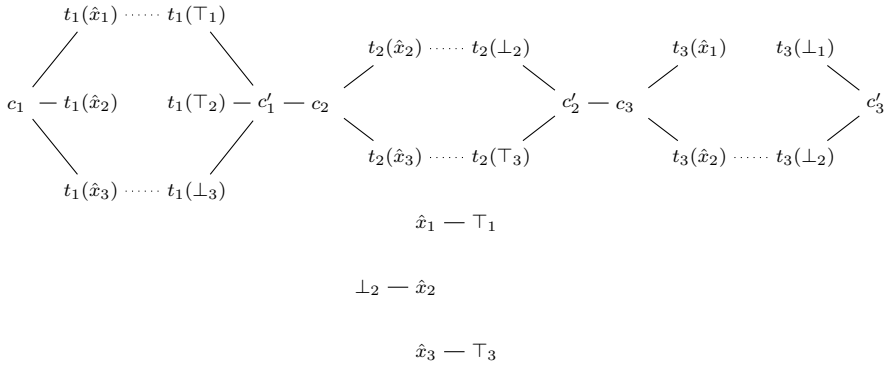

Fig. 4 The congruence graph for $E_{\mathcal{C}} \cup E_{\mathcal{I}}$

between terms $t_i(\hat{x}_j)$ and either $t_i(\top_j)$ or $t_i(\bot_j)$. So any path from c_1 to c'_n would go through such a congruence edge for each i. And such an edge exists for i if and only if the clause i is satisfied by $\mathcal{I}$.

($\Rightarrow$) If $\mathcal{I} \models \mathcal{C}$, then $\mathcal{I} \models C_i$ for each clause $C_i \in \mathcal{C}$. Assume $\mathcal{I}$ makes true a variable x_j, literal of C_i (the case of the negation of a variable is handled similarly). Then $E_{\mathcal{I}} \models t_i(\hat{x}_j) = t_i(\top_j)$, and $E_{\mathcal{I}} \cup \mathit{Clause}_i \models c_i = c'_i$. This is true for each i, and thanks to the equations in *Connect*, one can deduce using transitivity that $E_{\mathcal{I}} \cup E_{\mathcal{C}} \models c_1 = c'_n$.

Lemma 2 *Consider a (partial or total) assignment $\mathcal{I}$ for non-tautological clauses $\mathcal{C} = \{C_1, \dots C_n\}$ on variables $x_1, \dots, x_m$. $|E_{\mathcal{I}} \cup E_{\mathcal{C}}|$ and $|\mathcal{T}(E_{\mathcal{I}} \cup E_{\mathcal{C}})|$ are polynomial in n and m.*

Proof. $E_{\mathcal{I}}$ contains at most m equations, since for no j both $\mathcal{I} \models x_j$ and $\mathcal{I} \models \neg x_j$. The set *Connect* contains exactly $n-1$ equations. For every i, the set Clause_i contains at most $2m$ equations, resulting in $2mn$ equations for all clauses. In total, we thus have $|E_{\mathcal{I}} \cup E_{\mathcal{C}}| \leq n - 1 + m + 2mn$.

$E_{\mathcal{I}} \cup E_{\mathcal{C}}$ contains at most $2n + 3m + 3mn$ terms: $2n$ for c_i, c'_i, $3m$ for $\hat{x}_j, \top_j, \bot_j$ and $3mn$ for all possible combinations of $t_i(\hat{x}_j), t_i(\top_j), t_i(\bot_j)$.

Considering again Example 2, and particularly Figure 4, any transitivity chain from c_1 to c'_3 will pass through c'_1, c_2, c'_2 and c_3. Any acyclic path from c_1 to c'_3 will contain 11 edges: 3 congruence edges, $3 * 2$ edges in Clause_i for $i = 1, 2, 3$ and 2 edges from *Connect*.

Since every interpretation $\mathcal{I}$ is such that $E_{\mathcal{I}} \subset \mathit{Assignment}^\star$, one can try to relate the propositional satisfiability problem for a set of clauses $\mathcal{C} = \{C_1, \dots C_n\}$ to finding an explanation of $c_1 = c'_n$ in $\mathit{Assignment}^\star \cup E_{\mathcal{C}}$. However, it is necessary that this explanation does not set $\hat{x}_j$ equal both to $\top_j$ and $\bot_j$, i.e. at most one of the two equations $\hat{x}_j = \top_j$ and $\hat{x}_j = \bot_j$ should be in the explanation. By restricting assignments to total ones, i.e. by enforcing that at least one of the two equations $\hat{x}_j = \top_j$ and $\hat{x}_j = \bot_j$ belongs to the explanation, it is also possible, with a single cardinality constraint on the explanation, to require that at most one of them belong to the explanation.

Lemma 3 *A set of non-tautological clauses $\mathcal{C} = \{C_1, \dots C_n\}$ using variables $x_1, \dots, x_m$ is satisfiable if and only if there is a subset $E' \subseteq Assignment^\star \cup E_{\mathcal{C}'}$ such that $E' \models c_1 = c'_{n+m}$ and $|E'| \leq 3n + 4m - 1$, where $\mathcal{C}'$ is $\mathcal{C}$ augmented with the tautological clauses $C_{n+i} = x_i \vee \neg x_i$ for $i = 1, \dots m$.*

Proof. ($\Rightarrow$) Consider a total model $\mathcal{I}$ for $\mathcal{C}$: $E_\mathcal{I} \subset Assignment^\star$ contains exactly m elements. For each clause i ($i = 1 \dots n + m$), collect in $E \subset E_{\mathcal{C}'}$ both equations relative to one satisfied literal (that is $2(n+m)$ equations in total), as well as the $n + m - 1$ equations from *Connect*. The equation $c_1 = c'_{n+m}$ is a consequence of the disjoint union $E \cup E_\mathcal{I}$, which contains $3n + 4m - 1$ elements.

($\Leftarrow$) An explanation of $c_1 = c'_{n+m}$ has to contain $2(n+m)$ equations from $Clause_i$ ($i = 1 \dots n+m$) and $n+m-1$ equations from *Connect*. Thanks to the tautological clauses, any explanation also has to contain at least $\hat{x}_j = \top_j$ or $\hat{x}_j = \bot_j$ for each $j \in \{1 \dots m\}$. Therefore, the cardinality constraint requires that the explanation contains at most one $\hat{x}_j = \top_j$ or $\hat{x}_j = \bot_j$ for each $j \in \{1 \dots m\}$. If such an explanation exists, Lemma 1 guarantees the existence of a model for $\mathcal{C}'$, or equivalently for the original set of clauses $\mathcal{C}$.

Corollary 1 (NP-hardness) *The small explanation problem is NP-hard.*

Proof. Propositional satisfiability is NP-hard, and can be reduced in polynomial time to the small explanation problem.

Lemma 4 (NP) *The small explanation problem is in NP.*

Proof. Let E be a set of equations and $s = t$ be a target equation. A solution to the explanation problem for some $k \in \mathbb{N}$ is a subset $E' \subseteq E$, such that $|E'| \leq k$. Let $n = |\mathcal{T}(E)| + |E|$ and $n' = |\mathcal{T}(E')| + |E'|$. We have $n' \leq n$, since $E' \subseteq E$ and every term in E' appears also in E. Checking whether E' is an explanation of $s = t$ can be done by computing its congruence closure, which is possible in polynomial time in n' [8] and thereby also in n.

Theorem 1 (Small explanation NP-completeness) *The small explanation problem is NP-complete.*

Proof. By corollary 1 and lemma 4.

Theorem 2 (Small conflict NP-completeness) *The small conflict set problem is NP-complete.*

Proof. The small conflict set problem is at least as hard as the small explanation problem since the small explanation problem has been showed to be reducible to the small conflict set problem. It is also in NP for exactly the same reason that the small explanation problem is.

5 Conclusion

The conflict set generation feature of congruence algorithms is essential for practical SMT solving. Although one could argue that the important property of the generated conflicts is minimality (i.e. no useless literal is in the conflict), it is also interesting to consider producing the smallest conflict. We have shown that the problem of deciding whether a conflict of a given size exists is NP-complete. Therefore, it is generally intractable to obtain the smallest conflict.

In [7,9,10], methods to obtain small conflicts, but not necessarily the smallest, are discussed. In practice, it pays off to prioritize speed of the congruence closure algorithm and conflict generation over succinctness of conflicts. However, other applications sensitive to proof size may benefit from other methods prioritizing small conflict size, at a cost of less efficient solving. Thanks to the NP-completeness, one option could be to iteratively encode the small conflict problem into SAT, and use a SAT-solver to find successively smaller conflicts, until the smallest is found. Perhaps a Max-SAT solver could be used instead, to avoid repeated calls to the SAT-solver and find the smallest conflict in one run.

Acknowledgment. We would like to thank Roberto Nieuwenhuis and Ashish Tiwari for discussions and some preliminary ideas that led us to this proof. We are grateful to the anonymous reviewers of [5] for their comments.

References

1. Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, February 2009.
2. Joseph Boudou, Andreas Fellner, and Bruno Woltzenlogel Paleo. Skeptik: A proof compression system. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *International Joint Conference on Automated Reasoning (IJCAR)*, volume 8562 of *Lecture Notes in Computer Science*, pages 374–380. Springer, 2014.
3. Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
4. Peter J. Downey, Ravi Sethi, and Robert E. Tarjan. Variations on the common subexpressions problem. *Journal of the ACM*, 27(4):758–771, October 1980.
5. Andreas Fellner, Pascal Fontaine, Georg Hofferek, and Bruno Woltzenlogel Paleo. NP-completeness of small conflict set generation for congruence closure. In Vijay Ganesh and Dejan Jovanovi, editors, *International Workshop on Satisfiability Modulo Theories (SMT)*, 2015.
6. Pascal Fontaine. *Techniques for verification of concurrent systems with invariants*. PhD thesis, PhD thesis, Institut Montefiore, Université de Liege, Belgium, 2004.
7. Pascal Fontaine and E. Pascal Gribomont. Using BDDs with combinations of theories. In Matthias Baaz and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 2514 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 2002.
8. Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, April 1980.
9. Robert Nieuwenhuis and Albert Oliveras. Union-find and congruence closure algorithms that produce proofs. In Cesare Tinelli and Silvio Ranise, editors, *Pragmatics of Decision Procedures in Automated Reasoning (PDPAR)*, 2004.

10. Robert Nieuwenhuis and Albert Oliveras. Proof-producing congruence closure. In Jürgen Giesl, editor, *Rewriting Techniques and Applications (RTA)*, volume 3467 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 2005.
11. Robert Nieuwenhuis and Albert Oliveras. Fast congruence closure and extensions. *Information and Computation*, 205(4):557–580, 2007.