A Few Exercises on the Complexity of Congruence Closure with Cardinality Constraints (Extended Abstract)

Ellen Dasnois, Pascal Fontaine

University of Liège, Belgium

Abstract
In this paper, we show various hardness results for the satisfiability of ground sets of literals in the theory of equality and uninterpreted functions (EUF) with finite cardinality constraints. It is known that this problem is NP-complete in the general case, when a domain $\mathcal{D}$ has a finite cardinality constraint $|\mathcal{D}| \leq n$, for any $n \geq 3$. We notice that this problem stays NP-complete in the case of a single domain of cardinality 2, indicating that even the Boolean sort can be hard to handle when functions are present. It is trivial that the function-free case with Boolean sorts has good complexity, but allowing a single binary function, or (an arbitrary number of) unary functions only, makes the problem NP-complete.

1. Introduction

The theory of equality and uninterpreted functions (EUF) is most central to many satisfiability modulo theories (SMT) solvers (see [1, 2] for a general background on SMT solvers). Uninterpreted functions are useful to abstractly model various objects, they are instrumental for reasoning on arrays or data-types, and, together with quantifier reasoning and a proper axiomatization, they are the key to extend a solver with custom data structures. Decision procedures for ground EUF are usually based on the congruence closure algorithm [3], which decides the satisfiability of a set of ground EUF literals in $n \log n$ time (assuming constant-time hash table operations).

In some SMT solvers, the congruence closure algorithm plays a special role as a central reasoning engine in the combination of decision procedures, keeping track of the ground terms, and dispatching relevant (dis-)equalities to the other decision procedures in the combination. Sometimes, the congruence closure algorithm itself goes a bit beyond just deciding EUF (e.g., [4]). In this short paper, we focus on cardinality constraints, and consider a few questions related to EUF and constraints on the cardinality of some domains. Cardinality constraints often appear in proof obligations stemming from verification of software or hardware systems, or more generally within any SMT-based formal task relying, for instance, on finite model finding [5] or (finite) data types [6, 7]. More marginally, since version 2.0 of the SMT-LIB standard [8] the Boolean sort can be used as the sort of function arguments. This requires attention, for instance, when checking the satisfiability of this seemingly pure EUF problem:

$$f(x) \not\approx f(y) \wedge f(y) \not\approx f(z) \wedge f(z) \not\approx f(x).$$

In fact, if the sort of the argument of uninterpreted function f is itself uninterpreted, the set is satisfiable, whereas it is unsatisfiable if the sort is Boolean.

We work in the context of ground (i.e., quantifier-free) many-sorted first-order logic with equality (see for instance [9] for a formal presentation of many-sorted first-order logic with equality). We discuss the impact of forcing one or more sorts to be interpreted as a finite domain of some given cardinality. Graph k-coloring reduces to congruence closure with a domain of cardinality k, and therefore (Section 3) this problem is NP-hard. Since two-coloring is polynomial, one could boldly believe that congruence closure on domains of cardinality 2 would have a better complexity. This is unfortunately not the case (Section 4.1): the problem stays NP-hard whenever functions are allowed, even if the functions are unary (Section 4.2). In Section 5, we discuss the related problem of finding implied disequalities.

SMT 2025: 23rd International Workshop on Satisfiability Modulo Theories, August 10–11, 2025, Glasgow, UK
Pascal.Fontaine@uliege.be (P. Fontaine)

2. Preliminaries

Satisfying a set of ground literals E in the theory of equality and uninterpreted functions boils down to extending E to a complete set E_c of equalities and disequalities over the terms in E (i.e., for each pair of terms t, t'in E, we have either $t \approx t' \in E_c$, or $t \not\approx t' \in E_c$), such that E_c satisfies the EUF axioms:

- Reflexivity: $\forall x \,.\, x \approx x$
- Symmetry: $\forall x, y \,.\, x \approx y \Rightarrow y \approx x$
- Transitivity: $\forall x, y, z \,.\, (x \approx y \wedge y \approx z) \Rightarrow x \approx z$
- Congruence (schema):
  $\forall x_1, \dots, x_n, y_1, \dots, y_n \,.\, (x_1 \approx y_1 \wedge \dots \wedge x_n \approx y_n) \Rightarrow f(x_1, \dots, x_n) \approx f(y_1, \dots, y_n)$

This is the goal of congruence closure. In extending it for finite cardinality constraints, we must make sure not to break those constraints.

Reflexivity is implicitly assumed and, therefore, automatically satisfied. Symmetry and transitivity are enforced by the use of the union-find data structure, i.e., by working with congruence classes rather than with the equalities directly. Two terms are supposed to be equal if and only if they are in the same congruence class. The equalities in E are satisfied by merging the necessary classes: if $t \approx t' \in E$, the classes of t and t' are merged. Finally, the congruence axioms are satisfied by merging the classes of functional terms $f(t_1, \dots t_n)$ and $f(t'_1, \dots t'_n)$ whenever their top symbols and arguments match. The result is the minimal set of equalities that satisfies both the EUF axioms and the equalities in E. Every possible disequality holds, and therefore E is satisfied if and only if none of the disequalities in E is violated in the resulting set of congruence classes.

Since this is the minimal set of equalities, congruence classes cannot be split to satisfy cardinality constraints. The only possible move is merging congruence classes until there are no more classes than the requested number of elements in the domain. The difficulty lies in doing this without violating the disequalities in E (or showing that this task is impossible).

3. (Dis-)Equalities with Finite Cardinality Constraints

In the absence of functions, the satisfiability problem for a set of ground equalities and disequalities is equivalent to graph coloring, as suggested in [5]. To be self-contained, we here briefly explain this equivalence again.

First, notice that it is not important whether we are considering the many-sorted case or the one-sorted case, the many-sorted case being simply the juxtaposition of independent one-sorted problems. Indeed, since there are no functions, each literal is of the form $a \approx b$, or $a \not\approx b$, where a and b are constants. There is no way for elements of different sorts to interact, so each sort can be treated independently.

Given a set of equalities and disequalities E in a sort S with some cardinality constraint stating that the sort S is of cardinality at most k, a congruence closure-like procedure can be applied to obtain an equivalence relation $\equiv_E$ over the terms in E. In absence of functions, the congruence rule plays no role, but a congruence closure algorithm can still be used to compute this equivalence relation.

Once this equivalence relation is known, it remains to assign one of at most k elements to each equivalence class, while making sure not to assign the same element to two classes that have a disequality between them. The classes that are assigned the same element can subsequently be merged. This is exactly the problem of graph k-coloring: the vertices are the equivalence classes, the edges are the disequalities between those classes, and the k colors are the k elements of the domain. Therefore, determining the satisfiability of a set of ground equalities and disequalities in presence of cardinality constraints reduces to graph coloring. Conversely, any graph coloring problem can be trivially encoded as a satisfiability problem in the theory of finite cardinality constraints. Graph coloring is NP-complete for any $k \geq 3$, and thus, the same applies to the satisfiability of a set of ground equalities and disequalities with finite cardinality constraints. For $k \leq 2$, however, graph coloring can be solved in linear time, yielding a linear-time procedure for the satisfiability of a set of (dis-)equalities with cardinality constraints only imposing domains of at most two elements.

Incidentally, note that graph coloring can be instrumental to prove the NP-hardness of many first-order theories [10].

4. Uninterpreted Functions and Finite Cardinality Constraints

We now consider the complexity when the theory of uninterpreted functions is combined with finite cardinality constraints. For cardinality constraints requiring domains to contain at most k elements with $k \geq 3$, the problem is NP-complete, since the hardness is directly inherited from the case without functions above. In the degenerate case $k = 1$, the problem is no harder than standalone EUF. Within the sorts which have a cardinality 1 constraint, all elements must be equal. The problem can thus be solved by applying regular congruence closure, and merging all congruence classes within the cardinality-restricted sorts. The polynomial complexity of congruence closure is preserved. For $k \leq 2$, one could hope to still have polynomial-time algorithms, as in the function-free case. The case $k = 2$ would be particularly helpful in integrating Boolean cardinality constraints with EUF reasoning. Unfortunately, we show that functions do actually make the problem significantly harder.

4.1. 2-Element Domain with Functions of Arbitrary Arity

With $k = 2$, the problem is NP-hard: there is a simple polynomial reduction of 3-SAT to the satisfiability of a set of literals in the (quantifier-free) theory of equality with uninterpreted functions, where at least one sort is constrained to a domain with at most two elements.

First, introduce two constants for the sort constrained to at most two elements, and impose that they are different, by asserting a disequality between those constants. This results in the sort having at least two congruence classes. Those two congruence classes will represent elements which are true and false respectively, so the constants are named accordingly.

$$\text{true} \not\approx \text{false}$$

The cardinality constraint on the sort imposes that any element in the sort is either equal to true or false. Next, for each proposition in the 3-SAT instance, introduce two constants representing the proposition and its negation. The two constants must be different, since they have different truth values. For example, with propositions a, b, c, introduce constants $a, b, c, \bar{a}, \bar{b}, \bar{c}$, and add the constraints:

$$a \not\approx \bar{a}$$
$$b \not\approx \bar{b}$$
$$c \not\approx \bar{c}$$

Finally, introduce a function $f : \text{Bool}^3 \to S$, where Bool is the sort of cardinality two, and S is an arbitrary sort (with cardinality 2, finite cardinality larger than 2, or arbitrary cardinality). For each clause in the 3-SAT instance, e.g. $a \vee b \vee \neg c$, introduce a corresponding constraint of the form:

$$f(a, b, \bar{c}) \not\approx f(\text{false}, \text{false}, \text{false})$$

Since the top symbol is the same, this constraint imposes that one of the arguments of f differs, i.e., that $a \not\approx$ false, $b \not\approx$ false, or $\bar{c} \not\approx$ false, which is equivalent to the constraint imposed by the clause.

If a model is found that satisfies all these constraints, it can trivially be translated to a model of the original 3-SAT formula. The converse is also true, meaning that this set of disequalities is satisfiable if and only if the original 3-SAT problem is.

Note that S *can* be the same sort as Bool. An example of a satisfying f would be a function that maps a clause to its truth value. This means that the proof holds even if the Boolean sort is the only sort in the problem. Furthermore, if f returns Boolean values, a single binary function is enough; clauses can be encoded in the following way:

$$f(f(a,b),\bar{c}) \not\approx f(f(\text{false},\text{false}),\text{false})$$

The problem is also in NP: it can be solved in polynomial time by non-deterministically partitioning the set of terms of each cardinality-restricted sort into (at most) two classes, then running the usual congruence closure algorithm. This can be done by first selecting two terms to be the representatives of the two classes (or a single representative term for the case where one class is empty), then adding an equality with one of the representatives for each term. This operation takes linear time, and linearly increases the size of the set of literals. Since congruence closure takes polynomial time, the complete runtime is also polynomial.

4.2. 2-Element Domain with Unary Functions Only

Even in the case where all functions are unary, the satisfiability problem for a set of EUF literals where at least one sort must be interpreted as a domain of cardinality two remains NP-complete. This is due to the interaction between transitivity and congruence. Disequalities stemming from transitivity alone are easy to find by reasoning about equivalence classes rather than individual (dis-)equalities. For congruence, in the unary case, the rule manifests as a set of binary clauses, each with one positive and one negative literal (e.g., $a \not\approx b \vee f(a) \approx f(b)$). Thus, exhaustively propagating all consequences of the congruence rule is easy. Taken together, however, transitivity and congruence can interact in complex ways and allow encoding arbitrary clauses. Consider the literals:

$$f(a_1) \not\approx g(a_3)$$

$$f(a_2) \approx g(a_4)$$

Adding equalities $a_1 \approx a_2$ and $a_3 \approx a_4$ leads to a conflict, but neither equality leads to a conflict on its own. In other words, the disjunction $a_1 \not\approx a_2 \vee a_3 \not\approx a_4$ is a logical consequence of these literals, and neither of its disjuncts is. A ternary disjunction can be built by applying this encoding twice: $a_1 \not\approx a_2 \vee a_3 \not\approx a_4 \vee a_5 \not\approx a_6$ becomes:

$$f(a_1) \not\approx g(a_3) \vee a_5 \not\approx a_6$$

$$f(a_2) \approx g(a_4)$$

then,

$$f(f(a_1)) \not\approx g(a_5)$$
$$f(g(a_3)) \approx g(a_6)$$
$$f(a_2) \approx g(a_4)$$

This can be used to encode 3-SAT. Using a similar encoding as in Section 4.1, a clause $a \vee b \vee \neg c$ can be translated into EUF literals using two unary functions:

$$f(f(a)) \not\approx g(\bar{c})$$

$$f(g(b)) \approx g(\text{false})$$

$$f(\text{false}) \approx g(\text{false})$$

One can easily verify by case analysis on the truth values of a, b, and $\bar{c}$, that this is satisfiable (i.e., there exist appropriate functions f and g over the 2-element domain) if and only if a, b, and $\bar{c}$ are not all equal to false. Additional clauses can be encoded in a similar way, introducing two fresh functions instead of f and g for each clause. This SAT encoding is polynomial, therefore the satisfiability of sets of EUF literals with unary functions only is NP-hard, if the domain is constrained to two elements.

4.3. No Functions from Cardinality-Restricted Sorts

In general, EUF problems in SMT are expressed in many-sorted logic, and have to be solved for multiple domains at once. Some domains might have cardinality constraints, and some others might not. When there exists a function that takes arguments of multiple different sorts, or that returns values of a sort different from the argument sort(s), the corresponding domains have to be treated together.

Sections 4.1 and 4.2 suggest that functions are hard to handle in the case of a 2-element domain, but this does not mean that all functions need to be avoided in order to maintain a polynomial complexity. If none of the functions take arguments of a cardinality-restricted sort, the reduction of Section 3 can still be applied. This time, the congruence rule needs to be used, since functions are present, so the first step of the reduction corresponds to regular congruence closure.

After applying congruence closure, notice that merging congruence classes of a cardinality-restricted sort does not require merging any other classes; since no functions take arguments of this sort, the congruence rule cannot apply. This means that merging two congruence classes of this sort will lead to a conflict if and only if there is an explicit disequality in E between elements of those classes. Thus, the problem once again reduces to graph 2-coloring: two classes can have the same color (i.e., correspond to the same element) if and only if there is no explicit disequality between them.

5. Disequality Propagation: Discussion and Future Work

The above theoretical bounds have a practical consequence: any procedure for congruence closure with finite cardinality constraints, except in very degenerate cases, will require some kind of SAT solver-like reasoning capacity, since the problem is NP-complete. Given a partition of terms in congruence classes, taking into account the cardinality constraints essentially boils down to iteratively merging classes, until the target cardinality upper bounds are met for the sorts with cardinality restrictions. It would therefore be highly valuable to know beforehand, at each point, which pairs of classes not to merge, i.e., finding out which disequalities are consequences of the current set of literals. This is not totally trivial: if two congruence classes do not have an explicit disequality between them, merging them can lead to other class merges, leading to a "hidden" conflict. For example, with three Boolean constants a, b, c, and given the disequality $f(a) \not\approx f(b)$, where f returns values of an arbitrary unconstrained sort, merging the classes of a and b is not possible. There is no explicit disequality between their classes, but merging them leads to merging the classes of $f(a)$ and $f(b)$, triggering a conflict.

To avoid this, one solution is to propagate disequalities "backwards" through the congruence rule. Taking the contrapositive of the congruence rule gives $f(a) \not\approx f(b) \Rightarrow a \not\approx b$. This allows propagating $a \not\approx b$, this explicit disequality preventing the merge of the classes of a and b. With binary functions and above, this becomes more complicated since the propagation yields a disjunction of disequalities. For example, we have $f(a_1, a_2) \not\approx f(b_1, b_2) \Rightarrow (a_1 \not\approx b_1 \lor a_2 \not\approx b_2)$.

Exhaustive disequality propagation is obviously polynomial. Indeed, if there are n terms, it suffices to run $n \times (n-1)/2$ times the quasi-linear satisfiability checking procedure on $E \cup (t \approx t')$ for any pair t, t' of terms to check whether $t \not\approx t'$ is a logical consequence of E. Exhaustive disequality propagation is therefore upper bounded by $n^3 \log n$. In practice, however, deducing all implied equalities might be subtle. Considering the literal

$$f(x_1, \ldots x_n) \not\approx f(y_1, \ldots y_n),$$

it is clear that the disjunction $x_1 \not\approx y_1 \lor \ldots x_n \not\approx y_n$ holds. The set of constraints might contain equalities $x_i \approx g_i(t)$ and $y_i \approx g_i(u)$ for each i, and therefore $t \not\approx u$ would in turn be a consequence of each disjunct. This however requires to consider each $x_i \not\approx y_i$ successively, before concluding that $t \not\approx u$ holds.

For future work, it would be interesting to either prove that the time complexity of disequality propagation has an impractical lower bound, e.g., n^2, or to design an efficient algorithm (i.e., with quasi-linear time complexity).

6. Conclusion

Our objective is to understand how to best tune a practical congruence closure algorithm to accommodate cardinality constraints. It appears that checking the satisfiability of sets of EUF literals with cardinality constraints is NP-hard, even with strong restrictions on the use of functions. A bit surprisingly (the graph 2-coloring problem is linear) this high complexity even occurs for cardinality constraints with two elements. This tells us that a complete algorithm will necessarily embed reasoning capabilities similar to those of a SAT solver. Future work will need to determine whether efficient algorithms can simply embed off-the-shelf SAT solvers, or if they need to be tailored while getting inspiration from state-of-the-art SAT solving techniques.

Heuristics are often crucial for practically solving NP-complete problems. Here, we believe that disequality propagation is worth further investigation.

Acknowledgments The authors are grateful to the anonymous reviewers for their comments. Ellen Dasnois gratefully acknowledges the financial support of the Wallonia-Brussels Federation for her FRIA grant. Research reported in this paper was supported in part by an Amazon Research Award, Fall 2022 CFP. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of Amazon. This work was also partially supported by the Fonds de la Recherche Scientifique - FNRS WEAVE under Grant number T.W028.23.

Declaration on Generative AI The authors have not employed any Generative AI tools.

References

[1] C. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: A. Biere, M. J. H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2009, pp. 825–885.

[2] C. Barrett, D. Kroening, T. Melham, Problem Solving for the 21st Century: Efficient Solvers for Satisfiability Modulo Theories, Technical Report 3, London Mathematical Society and Smith Institute for Industrial Mathematics and System Engineering, 2014. URL: http://www.cs.nyu.edu/~barrett/pubs/BKM14.pdf, knowledge Transfer Report.

[3] P. J. Downey, R. Sethi, R. E. Tarjan, Variations on the common subexpressions problem, Journal of the ACM 27 (1980) 758–771.

[4] R. Nieuwenhuis, A. Oliveras, Congruence closure with integer offsets, in: M. Y. Vardi, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning (LPAR), volume 2850 of *Lecture Notes in Computer Science*, Springer, Almaty, Kazakhstan, 2003, pp. 78–90.

[5] A. Reynolds, C. Tinelli, A. Goel, S. Krstic, Finite model finding in SMT, in: N. Sharygina, H. Veith (Eds.), Computer Aided Verification (CAV), volume 8044 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 640–655.

[6] D. C. Oppen, Reasoning about recursively defined data structures, Journal of the ACM 27 (1980) 403–411.

[7] C. Barrett, I. Shikanian, C. Tinelli, An abstract decision procedure for a theory of inductive data types, Journal on Satisfiability, Boolean Modeling and Computation 3 (2007) 21–46.

[8] C. Barrett, A. Stump, C. Tinelli, The SMT-LIB standard : Version 2.0, 2010. First official release of Version 2.0 of the SMT-LIB standard.

[9] H. B. Enderton, A Mathematical Introduction to Logic, Academic Press, Inc., Orlando, Florida, 1972.

[10] R. Sebastiani, Colors make theories hard, in: N. Olivetti, A. Tiwari (Eds.), International Joint Conference on Automated Reasoning (IJCAR), volume 9706 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 152–170.