Interpolant generation for EUF using Ground Horn Clauses with Explanations

José Abel Castellanos Joo
Deepak Kapur
jose.castellanosjoo@cs.unm.edu
kapur@cs.unm.edu
University of New Mexico
Albuquerque, New Mexico

ABSTRACT

TODO:

CCS CONCEPTS

• **Computing methodologies** → **Equation and inequality solving algorithms**; • **Theory of computation** → *Logic and verification.*

KEYWORDS

Craig's interpolant, free theory, congruence closure algorithms

ACM Reference Format:
José Abel Castellanos Joo and Deepak Kapur. 2020. Interpolant generation for EUF using Ground Horn Clauses with Explanations. In *UNM 16th Annual Computer Science Student Conference, March 25th, 2020, Albuquerque, NM*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Interpolation algorithms for the theory of equality with uninterpreted functions are relevant as the core part of program analysis and verification algorithms. Many useful technique in software engineering like bounded/unbounded model checking and invariant generation benefit directly from this technique. TODO. (There might be no more space to write anything else here.)

The contributions of this paper are the following:

- Performance improvement of the conditional replacement step in Kapur's algorithm for the interpolation generation of the conjunction of equalities in the theory of equality with uninterpreted functions.
- An efficient algorithm implementing the Explanation operation for testing satisfiability of grounded Horn clauses.

UNM 16th Annual Computer Science Student Conference, March 25, 2020, Albuquerque, NM

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND

2.1 Craig's Interpolant

Let α, β, γ be logical formulas in a given theory. If $\vdash \alpha \rightarrow \beta$, we say that γ is an interpolant for the pair (α, β) if the following conditions are met:

- $\vdash \alpha \rightarrow \gamma$
- $\vdash \gamma \rightarrow \beta$
- Every non-logical symbol in γ occurs both in α and β.

The *interpolantion problem* can be stated naturally as follows: given two logical formulas α, β such that $\vdash \alpha \rightarrow \beta$, find the interpolant for the pair (α, β).

In his celebrated work [5], Craig proved that for every pair (α, β) of first order formulas such that $\vdash \alpha \rightarrow \beta$, an interpolation formula exists.

Usually, we see the interpolation problem defined differently in the literature, where we consider $\beta^{'}$ to be $\neg\beta$ and the problem requires that the pair $(\alpha, \beta^{'})$ is mutually contradictory (unsatisfiable). This definition was popularized by McMillan [15]. This shift of attention explains partially the further development in interpolation generation algorithms since many of these relied on SMT solvers that provided refutation proofs in order to (re)construct interpolants for different theories (and their combination) [4, 12, 14].

Extended definitions are given to the interpolation problem when dealing with specific theories [20] in a way that interpreted function can be also part of the interpolant. The latter is justify since otherwise, many interpolation formulas might not exists in different theories (for example, lisp programs).

We also see different approaches to interpolation generation for particular theories that exploits aspects of the underlying theory [].

2.2 Equality with Uninterpreted Functions

The language of *Equality with Uninterpreted Functions* (EUF) consists of the equality symbol as the unique predicate symbol, and a countable number of functional symbols (including constants of 0-arity). Its theory consists only of the axioms defining the equality symbol to be an equivalence relation:

- *Reflexivity* $\forall x.x = x$
- *Symmetry* $\forall x, y.x = y \rightarrow y = x$
- *Transitivity* $\forall x, y, z.(x = y \wedge y = z) \rightarrow x = z$
- *Congruence* $\forall$ n-arity function symbol, $n > 0.\forall x_1, \ldots, x_n, y_1, \ldots, y_n.(x_1 = y_1 \wedge \cdots \wedge x_n = y_n) \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

We notice that the Congruence axiom is not an first-order axiom but rather an axiom scheme [17]. No other axiom is added. Because of this, the EUF theory is also called the *empty theory*.

2.3 Kapur's Algorithm for EUF Theory

Kapur's interpolation algorithm for the EUF theory uses quantifier-elimination techniques to remove symbols in the first formula of an interpolation problem instance that are not common with the second formula of the latter. Hence, the input for this algorithm is a conjunction of equalities in the EUF theory and a set of symbols to eliminate, also unknown as uncommon symbols.

The steps/ideas in Kapur's algorithm for interpolant generation for the EUF theory are the following:

- **Elimination of uncommon terms using Congruence Closure**. This step builds an equivalence relation using the input of the algorithm such that the representatives are common terms whenever possible. Let *answer* be a variable denoting the conjunction of all the input formulas which uncommon subterms are replaced by their representatives. If all the representatives in the equivalence relation are common terms, return *answer* as the interpolant. Otherwise, continue with the following step.
- **Horn clause generation by exposure**. This step uses Ackermann's reduction [13] to produce Horn clauses to eliminate uncommon terms identifying two cases:
  - The term is uncommon because the function symbol is uncommon.
  - The term is uncommon because at least one of its arguments is an uncommon term.

  Conjunct to *answer* the conjunction of all these Horn Clauses. If all the Horn Clauses above are common, return *answer* as the interpolant. Otherwise, continue with the following step.
- **Conditional elimination**. Since some of the Horn clauses produced by the previous step are not common, we identify the Horn clauses that have *common antecedents* and head equation with at least one uncommon term. For each of these Horn clauses, replace the uncommon term in its head by the common term in its head in the rest of the Horn Clauses, and include its antecedent to the antecedent of such Horn clauses. We can see that this step reduces the number of uncommon terms in the equalities of the Horn Clauses. We repeat this step until it cannot be performed. At the end, we take only the set of Horn Clauses that have common antecedents and have one uncommon term in its head. The call these Horn clauses *useful Horn clauses*. Continue with the next step.
- **Conditional replacement**. Using the *useful Horn clauses* generated in the previous step, update *answer* to be the formula resulting after *conditionally replacing* uncommon terms in each equation of *answer* by an appropriate common term in the head of a *useful Horn clauses*. To be more precise, let $\bigwedge_i a_i = b_i \rightarrow u \mapsto c$ be a useful Horn clause, where the antecedent is a conjunction of common grounded equations, u is an uncommon term, and c is a common term. Then for every instance of u in each equation of *answer*, conditionally replace u by c under $\bigwedge_i a_i = b_i$. We notice that equations in *answer* of the previous step will become Horn Clauses with less uncommon terms. For completeness, we perform these replacements zero or more times (up to the maximal number of instances per equation) in order to leave space for other *useful Horn Clauses* to replace the uncommon term in their heads as well. Remove all the literals in the current *answer* that contain uncommon terms and return this as the interpolant.

If the user is not interested in an explicit interpolant, we can present the *usable Horn clauses* in a proper order such that the replacements can be done without exponentially increasing the size of the interpolant. This representation is useful because it provides a more compact representation of the interpolant that the user might be able quicker to obtain. Additionally, the user might be just interested in a particular subformula of the interpolant, so the latter representation offers feature.

This algorithm allows a flexible implementation which can lead several optimizations based on the nature and applications of the interpolants. TODO.

2.4 Ground Horn Clauses unsatisfiability testing

In [11] it was proposed an algorithm for testing the unsatisfiability of ground Horn clauses with equality. The main idea was to interleave two algorithms: *implicational propagation* (propositional satisfiability of Horn clauses) that updates the truth value of equations in the antecedent of the input Horn clauses [8]; and *equational propagation* (congruence closure for grounded equations) to update the state of a Union-Find data structure [10] that keeps the minimal equivalence relation defined by grounded equations in the input Horn clauses.

The author in [11] defined two variations of his algorithms by adapting the Congruence Closure algorithms in [9, 17]. Additionally, modifications in the data structures used by the original algorithms were needed to make the interleaving mechanism more efficient.

2.5 Congruence Closure with Explanations

In [19], the authors introduced a Union-Find data structure that supports the Explanation operation. This operation receives as input an equation between constants. If the input equation is a consequence of the current equivalence relation defined in the Union Find data structure, the Explanation operation returns the minimal sequence of equations used to build such equivalence relation, otherwise it returns 'Not provable'. A proper implementation of this algorithm extends the traditional Union-Find data structure with a *proof-forest*, which consists of an additional representation of the underlying equivalence relation that does not compress paths whenever a call to the Find operation is made. For efficient reasons, the Find operation uses the path compression and weighted union.

The main observation in [19] is that, in order to recover an explanation between two terms, by traversing the path between the two nodes in the proof tree, the last edge in the path guarantees to be part of the explanation. Intuitively, this follows because only the last Union operation was responsible of merging the two classes into one. Hence, we can recursively recover the rest of the explanation by recursively traversing the subpaths found.

Additionally, the authors in [19] extended the Congruence Closure algorithm [18] using the above data structure to provide Explanations for EUF theory. The congruence closure algorithm is a simplification of the congruence closure algorithm in [9]. The latter combines the traditional *pending* and *combine* list into one single list, hence removing the initial *combination* loop in the algorithm in [9].

3 MAIN CONTRIBUTION

First, we explain a high level ideal on how we improve the *conditional elimination* step in Kapur's algorithm. We notice that this step *propagates equationally* the head equations of grounded Horn clauses with common antecedents. Initially we employ the unsatisfiability algorithm for Horn clauses to achieve such propagation. However, the original algorithm will not be enough because it will only propagate the head equation when all the antecedents have truth value equal to true. To fix that problem, we modify two steps in Gallier's algorithms:

- When we build the data structure *numargs* that keeps track of the number of unproven equations in the antecedent of each Horn clause, we change this number by the number of unproven uncommon equations in the antecedent of each Horn clause. This will be useful because we only introduce head equations intro the queue data structure in Gallier's algorithm when all the antecedents are true. With this modification, our algorithm introduces head equations when all the antecedent equations are common. Additionally the algorithm can still update correctly the truth value of common equations, but these are not relevant for our propagation purposes.
- To guarantee that *numargs* keeps the right number of uncommon equations yet to be proven, we also modify the update mechanism for *numargs* in the main while loop of the algorithm. The original algorithm reduces by one the corresponding entry in *numargs* whenever a recently popped element from the queue matches the antecedent of a Horn clause. We only decrease this value if such popped equation is uncommon. This prevents the algorithm from accidentally reducing the number of uncommon equations yet to be proven, which can cause that we propagate the uncommon head equation when the antecedent of a Horn clause only consists of common equations.

At the end of this algorithm we can identify *usable Horn clauses* by checking the Horn clauses with *numargs* entries equal to 0. Nonetheless, these Horn clauses are not the desired *usable Horn clauses* because the unsatisfiability testing algorithm did not update the antecedents of the Horn clauses. The main difficulty to design a data structure for the latter to work inside the unsatisfiability testing algorithm was the queue data structure only adds grounded equation whenever the truth value of the literal changes to true, which happens during *equational propagation* or during the *implicational propagation* steps. For the *implicational propagation* the task is easy because we can know the clause where the just new proven ground equation comes, but it cannot be the same situation for the *equational propagation* since this step relies on the Congruence Closure.

To remedy this issue, we equip our Congruence Closure algorithm with the Explanation operator, so we can recover the grounded equations needed to entail any particular grounded equation. Additionally, this will require a data structure to maintain the Horn clauses for each grounded equation that it is the head equation of. With the latter we can recover the Horn Clauses where each grounded equation came from to update the antecedents and obtain *usable Horn clauses*.

3.1 New optimized conditional elimination step in Kapur's algorithm

The algorithm appears below in pseudo code notation:

Algorithm 1 Modified Unsatisfiability Testing for Ground Horn Clauses

```
1:  procedure SATISFIABLE(var H : Hornclause; var queue, combine: queuetype; var GT(H) : Graph; var consistent : boolean)
2:      while queue not empty and consistent do
3:          node := pop(queue);
4:          for clause1 in H[node].clauselist do
5:              if ¬ clause1.isCommon() then
6:                  numargs[clause1] := numargs[clause1] - 1
7:              end if
8:              if numargs[clause1] = 0 then
9:                  nextnode := poslitlist[clause1];
10:                 if ¬ H[nextnode].val then
11:                     if nextnode ≠ ⊥ then
12:                         queue := push(nextnode, queue);
13:                         H[nextnode].val := true;
14:                         u := left(H[nextnode].atom); v := right(H[nextnode].atom);
15:                         if FIND(R, u) ≠ FIND(R, v) then
16:                             combine := push((u, v), combine);
17:                         end if
18:                     else
19:                         consistent := false;
20:                     end if
21:                 end if
22:             end if
23:         end for
24:         if queue is empty and consistent then
25:             closure(combine, queue, R);
26:         end if
27:     end while
28: end procedure

29: procedure CLOSURE(var combine, queue : queuetype; var R : partition)
30:     while combine is not empty do
31:         (u, v) = pop(combine)
32:         MERGE(R, u, v, queue)
33:     end while
34: end procedure
```

Algorithm 2 Modified Congruence Closure with Explanation Algorithms

```
procedure MERGE(R : partition, u, v : node; queue, combine : queuetype)
    if u then and v are constants a and b
        add a = b to Pending; Propagate();
    else                                   ▷ u=v is of the form f(a1, a2)=a
        if L thenookup(Representative(a1), Representative(a2)) is some f(b1, b2)=b
            add (f(a1, a2)=a, f(b1, b2) = b) to Pending; Propagate();
        else
            set Lookup(Representative(a1), Representative(a2)) to f(a1, a2)=a;
            add f(a1, a2)=a to UseList(Representative(a1)) and to UseList(Representative(a2));
        end if
    end if
end procedure

procedure PROPAGATE( )
    while P doending is non-empty
        Remove E of the form a=b or (f(a1, a2) = a, f(b1, b2) = b) from Pending
        if thenRepresentative(a) ≠ Representative(b) and w.l.o.g. |ClassList(Representative(a))| ≤ |ClassList(Representative(b))|
            oldReprA := Representative(a);
            Insert edge a ⟶ b labelled with E into the proof forest;
            for e doach c in ClassList(oldReprA)
                set Representative(c) to Representative(b)
                move c from ClassList(oldReprA) to ClassList(Representative(b))
                for e doach pointer L in ClassList(u)
                    if H then[L].val = false
                        set the field H[L].lclass or H[L].rclass pointed to by p to Representative(b)
                        if H then[L].lclass = H[L].rclass
                            queue := push(L, queue);
                            H[L].val := true
                        end if
                    end if
                end for
            end for
            for e doach f(c1, c2) = c in UseList(oldReprA)
                if L thenookup(Representative(c1), Representative(c2)) is some f(d1, d2) = d
                    add (f(c1, c2) = c, f(d1, d2) = d) to Pending;
                    remove f(c1, c2) = c from UseList(oldReprA);
                else
                    set Lookup(Representative(c1), Representative(c2)) to f(c1, c2) = c;
                    move f(c1, c2) = c from UseList(oldReprA) to UseList(Representative(b));
                end if
            end for
        end if
    end while
end procedure
```

3.2 Ground Horn Clauses with Explanations

We notice that, by removing our changes to the unsatisfiability testing for grounded Horn clauses regarding uncommon symbols, we effectively combine the congruence closure with explanations to the original unsatisfiability testing algorithm. With the latter, we can query the membership of a Horn clauses in a given user-defined theory and additionally obtain a proof of the latter. This approach works by introducing the antecedent equations of a grounded Horn clause as part of the user-defined theory in order to prove its head equation. By the Deduction Theorem [16], we can recover a proof of the original queried Horn clause by removing the antecedent equations appearing the proof given by the Explain operation.

4 CONCLUSIONS

In this paper we have presented a new optimized step for conditional elimination in Kapur's algorithm for EUF interpolant generation. As a by-product, we discover an efficient algorithm to implement the Explain operator in grounded Horn clauses. To our knowledge, we are the first authors to contribute with such operation for grounded Horn clauses in particular. However, we can see that also an Explain operator can be encoded using the proof-producing capabilities of SMT solvers [1–3, 6, 7]. Nonetheless, the latter requires a refutational proof, our method might be able to outperform these approaches. For future work we plan to implement this algorithm to incorporate it into an implementation of Kapur's algorithm and compare the proof-producing feature with state of the art SMT solvers.

ACKNOWLEDGMENTS

To Prof. Kapur, for his patience and coping with me all these years.

REFERENCES

[1] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanoviundefined, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAVâĂŹ11)*. Springer-Verlag, Berlin, Heidelberg, 171âĂŞ177.

[2] Clark Barrett, Leonardo de Moura, and Pascal Fontaine. 2014. Proofs in satisfiability modulo theories. (07 2014).

[3] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. 2009. veriT: An Open, Trustable and Efficient SMT-Solver. In *Automated Deduction – CADE-22*, Renate A. Schmidt (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 151–156.

[4] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. 2013. Proof Tree Preserving Interpolation. In *Tools and Algorithms for the Construction and Analysis of Systems*, Nir Piterman and Scott A. Smolka (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 124–138.

[5] William Craig. 1957. Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory. *The Journal of Symbolic Logic* 22, 3 (1957), 269–285. http://www.jstor.org/stable/2963594

[6] Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 337–340.

[7] Leonardo de Moura and Nikolaj BjÃÿrner. 2008. Proofs and Refutations, and Z3. *CEUR Workshop Proceedings* 418.

[8] William F. Dowling and Jean H. Gallier. 1984. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *The Journal of Logic Programming* 1, 3 (1984), 267 – 284. https://doi.org/10.1016/0743-1066(84)90014-1

[9] Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. 1980. Variations on the Common Subexpression Problem. *J. ACM* 27, 4 (Oct. 1980), 758âĂŞ771. https://doi.org/10.1145/322217.322228

[10] Bernard A. Galler and Michael J. Fisher. 1964. An Improved Equivalence Algorithm. *Commun. ACM* 7, 5 (May 1964), 301âĂŞ303. https://doi.org/10.1145/364099.364331

[11] Jean H. Gallier. 1987. Fast algorithms for testing unsatisfiability of ground horn clauses with equations. *Journal of Symbolic Computation* 4, 2 (1987), 233 – 254. https://doi.org/10.1016/S0747-7171(87)80067-6

[12] Laura Kovács and Andrei Voronkov. 2009. Interpolation and Symbol Elimination. In *Automated Deduction – CADE-22*, Renate A. Schmidt (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 199–213.

[13] Daniel Kroening and Ofer Strichman. 2008. *Decision Procedures: An Algorithmic Point of View* (1 ed.). Springer Publishing Company, Incorporated.

[14] Kenneth McMillan. 2011. Interpolants from Z3 proofs. In *Formal Methods in Computer-Aided Design* (formal methods in computer-aided design ed.). https://www.microsoft.com/en-us/research/publication/interpolants-from-z3-proofs/

[15] K. L. McMillan. 2004. An Interpolating Theorem Prover. In *Tools and Algorithms for the Construction and Analysis of Systems*, Kurt Jensen and Andreas Podelski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 16–30.

[16] Elliott Mendelson. 2009. *Introduction to Mathematical Logic* (5th ed.). Chapman and Hall-CRC.

[17] Greg Nelson and Derek C. Oppen. 1980. Fast Decision Procedures Based on Congruence Closure. *J. ACM* 27, 2 (April 1980), 356âĂŞ364. https://doi.org/10.1145/322186.322198

[18] Robert Nieuwenhuis and Albert Oliveras. 2003. Congruence Closure with Integer Offsets. In *Logic for Programming, Artificial Intelligence, and Reasoning*, Moshe Y. Vardi and Andrei Voronkov (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 78–90.

[19] Robert Nieuwenhuis and Albert Oliveras. 2005. Proof-Producing Congruence Closure. In *Term Rewriting and Applications*, Jürgen Giesl (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 453–468.

[20] Greta Yorsh and Madanlal Musuvathi. 2005. A Combination Method for Generating Interpolants. In *Automated Deduction – CADE-20*, Robert Nieuwenhuis (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 353–368.