Efficient implementation of interpolation algorithms for the theories of quantifier-free equality with uninterpreted functions, unit two variable per inequality, and its combination

by

José Abel Castellanos Joo

B.Tech., Universidad de las Américas Puebla, 2015

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Science

The University of New Mexico

Albuquerque, New Mexico

July, 2020

Dedication

To my family.

Acknowledgments

TODO.

Efficient implementation of interpolation algorithms for the theories of quantifier-free equality with uninterpreted functions, unit two variable per inequality, and its combination

by

José Abel Castellanos Joo

B.Tech., Universidad de las Américas Puebla, 2015

M.S., Computer Science, University of New Mexico, 2020

Abstract

This thesis discusses theoretical aspects and an implementation for the interpolation problem of the following theories: (quantifier-free) equalitiy with uninterpreted functions (EUF), unit two variable per inequality (UTVPI), and the combination of the two previous theories. The interpolation algorithms implemented in this thesis are originally proposed in [12].

Refutational proof-based solutions are the usual approach of many interpolation algorithms [9, 18, 17]. The approach taken in [12] relies on quantifier-elimination heuristics to construct an interpolant using one of the two formulas involved in the interpolation problem. The latter makes possible to study the complexity of the algorithms obtained. On the other hand, the combination method implemented in this thesis uses a Nelson-Oppen framework, thus we still require for this particular situation a refutational proof in order to guide the construction of the interpolant for the combined theory.

The implementation uses Z3 [5] for parsing purposes and satisfiability checking in the combination component of the thesis. Minor modifications were applied to the Z3's enode data structure in order to label and distinguish formulas efficiently (i.e. distinguish A-part, B-part). Thus, the project can easily be integrated to the Z3 solver to extend its functionality for verification purposes using the Z3 plug-in module.

Contents

List of Figures

List of Tables

Glossary

TODO.

Chapter 1

Introduction

Modern society is witness of the impact computer software has done in recent years. The benefits of this massive automation is endless. On the other hand, when software fails it becomes a catostrophe to the point that some industries are severely affected and human's life are threaded at worst.

Due to strict and ambitious agendas, many software products are shipped with unseen bugs which might potential thread people's life like in critical systems. Several approaches have been used to improve software quality. However, many of these approaches offer partial coverage or might take an abyssal amount of human effort to provide such solutions and thus cannot be consideral practical. These contributions are relevant for certain applications and their proper use fit such workflows uniquely.

Formal methods aim to bring a unique combination of automation, rigor, and efficiency (whenever effecient algorithms exits for the verification task at hand). This thesis discusses a particular problem in software verification known as *the interpolantion problem* for the theories of the quantifier-free fragment of equality with uninterpreted functions (EUF) and unit two variable per inequality (UTVPI). These two theories have been studied extensively and researcher have found several applications for them.

1.1 Backgroud

An interpolant of a pair of two logical formula is a logical formula such that it is implied by the first formula in the pair and logically inconsistent with the second, having only common symbols of the two formulas in the original pair. Informally this means that the interpolant 'belongs' to the consequence of the first formula of such pair, and 'avoids' the consequence of the second formula. Not surprisingly, this intuition is behind many software verification routines where the first formula models the desirable state (termination, correctness) of a computer program, and the second formula models the set of undesirable states (non-termination, errors, crashes, etc) of such software. In Chapter 2, an extensive review of the formal concept is provided.

Interpolants find two main application in software verification:

- refinement of abstract models: TODO. explain the latter
- invariant generation: TODO. same

1.2 Related work

- (Theoretical) approaches Proof-based Color-based Brutomesso Bonachina
- (Implementation) iZ3

TODO.

1.3 Outline of the thesis

- Chapter X is about Y ...
- and so on and so on ...

TODO.

Chapter 2

Preliminaries

This chapter discusses basic concepts from first-order logic that are used in the rest of this thesis. We will pay particular attention to their language and semantics since these are fundamental concepts necessary to understand the algorithmic constructions to compute interpolants. For a comprehensive treatment on the topic, the reader is suggested the following references [19, 8].

2.1 First-Order Predicate Logic

2.1.1 Language

A language is a collection of symbols of different sorts equipped with a rule of composition that effectively tells us how to recognize elements that belong to the language [26]. In particular, a first-order language is a language that expresses boolean combinations of predicates using terms (constant symbols and function applications). In mathematical terms,

Definition 2.1.1. *A first-order language (also denoted signature) is a triple* $\langle \mathfrak{C}, \mathfrak{P}, \mathfrak{F} \rangle$

of non-logical symbols where:

- $\mathfrak{C}$ *is a collection of constant symbols*
- $\mathfrak{P}$ *is a collection of n-place predicate symbols*
- $\mathfrak{F}$ *is a collection of n-place function symbols*

including logical symbols like quantifiers(universal ($\forall$), existential($\exists$)) logical symbols like parenthesis, propositional connectives (implication ($\rightarrow$), conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$)), and a countably number of variables Vars *(i.e.* $\mathrm{Vars} = \{v_1, v_2, \dots\}$*).*

The rules of composition distinguish two objects, terms *and* formulas*, which are defined recursively as follows:*

- *Any variable symbol or constant symbol is a term.*
- *If $t_1, \dots, t_n$ are terms and f is an n-ary function symbol, then $f(t_1, \dots, t_n)$ is also a term.*
- *If $t_1, \dots, t_n$ are terms and P is an n-ary predicate symbol, then $P(t_1, \dots, t_n)$ is formula.*
- *If x is a variable and ψ, φ are formulas, then $\neg\psi, \forall x.\psi, \exists x.\psi$ and $\psi \Box \varphi$ are formulas where $\Box \in \{\rightarrow, \wedge, \vee\}$*

No other expression in the language can be considered terms nor formulas if such expressions are not obtained by the previous rules.

2.1.2 Semantics

In order to define a notion of truth in a first-order language is necessary to associate for each non-logical symbol (since logical symbols have established semantics from propositional logic) a denotation or mathematical object and an *assignment* to the collection of variables. The two previous components are part of an *structure* [8] (or interpretation [26]) for a first-order language.

Definition 2.1.2. *Given a first-order language $\mathfrak{L}$, an interpretation $\mathfrak{I}$ is a pair $(\mathfrak{A}, \mathfrak{J})$, where $\mathfrak{A}$ is a non-empty domain (set of elements) and $\mathfrak{J}$ is a map that associates to the non-logical symbols from $\mathfrak{L}$ the following elements:*

- $c^{\mathfrak{J}} \in \mathfrak{A}$ *for each* $c \in \mathfrak{C}$
- $f^{\mathfrak{J}} \in \{\mathfrak{A}^n \to \mathfrak{A}\}$ *for each n-ary function symbol* $f \in \mathfrak{F}$
- $P^{\mathfrak{J}} \in \{\mathfrak{A}^n\}$ *for each n-ary predicate symbol* $P \in \mathfrak{P}$

An assignment $s : Vars \to \mathfrak{A}$ is a map between Vars to elements from the domain of the interpretation.

With the definition of interpretation $\mathfrak{I}$ and assignment s, we can recursively define a notion of *satisfiability* (denoted by the symbol $\models_{\mathfrak{I},s}$) as a free extension from atomic predicates (function application of predicates) to general formulas as described in [8]. For the latter, we need to extend the assignment function to all terms in the language.

Definition 2.1.3. *Let $\mathfrak{I} = (\mathfrak{A}, \mathfrak{J})$ be an interpretation and s an assignment for a given language, Let $\bar{s} : Terms \to A$ be defined recursively as follows:*

- $\bar{s}(c) = c^{\mathfrak{J}}$
- $\bar{s}(f(t_1, \ldots, t_n)) = f^{\mathfrak{J}}(\bar{s}(t_1), \ldots, \bar{s}(t_n))$

Notice that the extension of s depends on the interpretation used.

Definition 2.1.4. *Given an interpretation $\mathfrak{I} = (\mathfrak{A}, \mathfrak{J})$, an assignment s, and ψ a formula, we define $\mathfrak{I} \models_s \psi$ (read ψ is satisfiable under interpretation $\mathfrak{I}$ and assignment s) recursively as follows:*

- $\models_{\mathfrak{I},s} P(t_1, \ldots, t_n)$ *if and only if* $P^{\mathfrak{I}}(\bar{s}(t_1), \ldots, \bar{s}(t_n))$
- $\models_{\mathfrak{I},s} \neg\psi$ *if and only if it is not the case that* $\models_{\mathfrak{I},s} \psi$
- $\models_{\mathfrak{I},s} \psi \wedge \varphi$ *if and only if* $\models_{\mathfrak{I},s} \psi$ *and* $\models_{\mathfrak{I},s} \varphi$
- $\models_{\mathfrak{I},s} \psi \vee \varphi$ *if and only if* $\models_{\mathfrak{I},s} \psi$ *or* $\models_{\mathfrak{I},s} \varphi$
- $\models_{\mathfrak{I},s} \psi \rightarrow \varphi$ *if and only if* $\models_{\mathfrak{I},s} \neg\psi$ *or* $\models_{\mathfrak{I},s} \varphi$
- $\models_{\mathfrak{I},s} \forall x.\psi$ *if and only if for every* $d \in \mathfrak{A}, \models_{\mathfrak{I},s_{x\mapsto d}} \psi$*, where* $s_{x\mapsto d} : Vars \rightarrow \mathfrak{A}$ *is reduct of* s *under* $Vars \setminus \{x\}$ *and* $s_{x\mapsto d}(x) = d$
- $\models_{\mathfrak{I},s} \exists x.\psi$ *if and only if exits* $d \in \mathfrak{A}, \models_{\mathfrak{I},s_{x\mapsto d}} \psi$*, where* $s_{x\mapsto d}$ *is defined as in the previous item.*

If an interpretation and assignment satisfies a formula, then we say that the interpretation and the assignment are a model for the respective formula. A collection of formulas are satified by an interpretation and assignment if these model each formula in the collection.

Additionally, if all the models $(\mathfrak{I}, s)$ in a language of a collection of formulas Γ satisfy a formula ψ, then we say that Γ logically implies ψ (written $\Gamma \models \psi$).

2.2 Mathematical Theories

A theory $\mathcal{T}$ is a collection of formulas that are closed under logical implication, i.e. if $\mathcal{T} \models \psi$ then $\psi \in \mathcal{T}$. This concept is quite relevant for our thesis work since we will focus on two theories, the quantifier-free fragment of the theory of equality with uninterpreted functions (EUF), and the theory of unit two variable per inequality (UTVPI).

For some theories it is enough to provide a collection of formulas (known as the axioms of the theory). For the case of the theories of interest for the thesis, the axiomatization if the following:

2.2.1 Equality with uninterpreted functions

Definition 2.2.1. *Let $\mathfrak{L}_{EUF} = \{\{\}, \{=\}, \{f_1, \dots, f_n\}\}$ be the language of EUF. The axioms of the theory are:*

- *(Reflexivity)* $\forall x.x = x$
- *(Symmetry)* $\forall x.\forall y.x = y \rightarrow y = x$
- *(Transitivity)* $\forall x.\forall y.\forall z.(x = y \wedge y = z) \rightarrow x = z$
- *(Congruence)* $\forall x_1 \dots \forall x_n.\forall y_1 \dots \forall y_n.(x_1 = y_1 \wedge \cdots \wedge x_n = y_n) \rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$

We notice that the congruence axiom is not a first-order logic axiom, but rather an axiom-scheme since it is necessary to *instantiate* such axiom for every arity of the function symbols in a given language.

2.2.2 Ordered commutative rings

In order to describe the UTVPI theory we will first introduce the language and theory of an ordered commutative ring.

Definition 2.2.2. *Let* $\mathfrak{L}_{Ord-R} = \{, \{0,1\}, \{=, \leq\}, \{+, -, *, /\}, \}$ *be the language of an ordered ring* R*. The axioms of the theory are:*

- $\forall x.\forall y.\forall z.x + (y + z) = (x + y) + z$
- $\forall x.\forall y.x + y = y + x$
- $\forall x.x + 0 = x$
- $\forall x.x + (-x) = 0$
- $\forall x.\forall y.\forall z.x * (y * z) = (x * y) * z$
- $\forall x.x * 1 = x$
- $\forall x.\forall y.\forall z.x * (y + z) = x * y + x * z$
- $\forall x.\forall y.\forall z.(y + z) * x = y * x + z * x$
- $\forall x.\forall y.\forall z.x \leq y \rightarrow x + z \leq y + z$
- $\forall x.\forall y.(0 \leq x \wedge 0 \leq y) \rightarrow 0 \leq x * y.$
- $0 \neq 1 \wedge 0 \leq 1$

In the next section we will provide further computability aspects for the theories of interest that are relevant for verification.

2.3 Interpolants

TODO.

2.3.1 Craig interpolation theorem

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

- $\models_{\mathcal{T}} \alpha \rightarrow \gamma$
- $\models_{\mathcal{T}} \gamma \rightarrow \beta$
- 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 $\models_{\mathcal{T}} \alpha \rightarrow \beta$, find the interpolant for the pair (α, β).

In his celebrated result [4], Craig proved that for every pair (α, β) of formulas in first-order logic such that $\models \alpha \rightarrow \beta$, an interpolation formula exists. Nonetheless, there are many logics and theories that this result does not hold [13].

Usually, we see the interpolation problem defined differently in the literature, where it is considered β' to be $\neg\beta$ and the problem requires that the pair (α, β') is mutually contradictory (unsatisfiable). This definition was popularized by McMillan [17]. 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) [14, 3, 18].

Extended definitions are given to the interpolation problem when dealing with specific theories [28] 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 [23, 25].

TODO.

2.3.2 Interpolants in verification (iZ3)

TODO.

2.4 Decision Procedures

Given a theory $\mathcal{T}$ in a formula ψ in the language of the theory, is it possible to know $\models_{\mathcal{T}} \psi$? The last question is known as the verification problem for the respective theory $\mathcal{T}$. This question has been studied extensively for many theories of interest [1].

The theories mentioned in Section 2.2 are actually undecidable [1, 8]. Nonetheless, the quantifier-free fragment of EUF and the restriction imposed in the decision problem for the UTVPI theory allow efficient algorithms to decide validity in their respective theories [20, 7, 16].

In the rest of this section we review some decision procedures used in the implementation work of the thesis.

2.4.1 Satisfiability and Satisfiability Modulo Theories

The naive solution to satisfiability checking of a formula consistent of enumerating all possible interpretations and assignments in a theory and test whether the last two are models for the formula. For the case of propositional logic we can see that such procedure with be $\mathcal{O}(2^n)$.

2.4.2 Congruence Closure

TODO.

2.4.3 Satisfiability of Horn clauses of propositions and grounded equations

TODO.

2.4.4 Nelson-Oppen framework for theory combination

TODO.

2.5 Algorithms

TODO.

2.5.1 DPLL, DPLL(T) and extensions

TODO.

2.5.2 Congruence Closure algorithms (DST, NO, with explanations)

In [22], 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 [22] 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 [22] extended the Congruence Closure algorithm [21] using the above data structure to provide Explanations for EUF theory. The congruence closure algorithm is a simplification of the congruence closure algorithm in [7]. The latter combines the traditional *pending* and *combine* list into one single list, hence removing the initial *combination* loop in the algorithm in [7].

TODO.

2.5.3 Gallier

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 [6]; 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 [7, 20]. Additionally, modifications in the data structures used by the original algorithms were needed to make the interleaving mechanism more efficient.

TODO.

2.6 Available tools

TODO.

2.6.1 Z3 and my minor modifications

TODO.

2.6.2 zChaff and my minor modifications

TODO.

2.6.3 DIMACS

TODO.

Chapter 3

The Theory of EUF

Interpolation algorithms for the theory of equality with uninterpreted functions are relevant as the core component of verification algorithms. Many useful techniques in software engineering like bounded/unbounded model checking and invariant generation benefit directly from this technique. In [2], the authors introduced a methodology to debug/verify the control logic of pipelined microprocessors by encoding its specification and a logical formula denoting the implementation of the circuit into a EUF solver.

Previous work addressing the interpolation problem for EUF has involved techniques ranging from interpolant-extraction from refutation proof trees [17, 18, 27], and colored congruence closure graphs [9]. Kapur's algorithm uses a different approach by using quantifier-elimination, a procedure that given a formula, it produces a logically equivalent formula without a variable in particular [8].

TODO.

3.1 Algorithm

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 [15] 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 $\bigwedge_i a_i = b_i \to u \mapsto c$ 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 such feature.

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

3.1.1 Contributions

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.*

3.1.2 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

```
procedure SATISFIABLE(var H : Hornclause; var queue, combine: queuetype;
var GT(H) : Graph; var consistent : boolean)
    while queue not empty and consistent do
        node := pop(queue);
        for clause1 in H[node].clauselist do
            if $\neg$ clause1.isCommon() then
                numargs[clause1] := numargs[clause1] - 1
            end if
            if numargs[clause1] = 0 then
                nextnode := poslitlist[clause1];
                if $\neg$ H[nextnode].val then
                    if nextnode $\neq \perp$ then
                        queue := push(nextnode, queue);
                        H[nextnode].val := true;
                        u := left(H[nextnode].atom); v := right(H[nextnode].atom);
                        if FIND(R, u) $\neq$ FIND(R, v) then
                            combine := push((u, v), combine);
                        end if
                    else
                        consistent := false;
                    end if
                end if
            end if
        end for
        if queue is empty and consistent then
            closure(combine, queue, R);
        end if
    end while
end procedure

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

Algorithm 2 Modified Congruence Closure with Explanation Algorithms - Merge

```
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
```

3.1.3 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 [19], we can recover a proof of the original queried Horn clause by removing the antecedent equations appearing the proof given by the Explain operation.

TODO.

Theorem 3.1.1. *Let t be an uncommon term and let H be a collection of Horn equations. Assume antecedent(H) $\neq \{\}$. If consequent(t) $= \{\}$, then we cannot conditionality eliminate t from any Horn equation $h \in H$.*

Proof. Suppose, by contradiction, that we can conditionally eliminate the term t from H. Then, there exists $h^{'} \in H$ such that t appears in the consequent of $h^{'}$. But

$consequent(t) = \{\}$, contradiction. □

Corollary 3.1.1.1. *We cannot eliminate an uncommont term* t *unconditionally once* $consequent(t) = \{\}$.

3.2 Implementation

TODO.

3.3 Evaluation

TODO.

Algorithm 3 Modified Congruence Closure with Explanation Algorithms - Propagate

```
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 then$Representative(a) \neq Representative(b)$ and w.l.o.g. $|ClassList(Representative(a))| \leq |ClassList(Representative(b))|$
            oldReprA := Representative(a);
            Insert edge $a \rightarrow b$ 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
```

Chapter 4

The Theory of Octagonal Formulas

TODO.

4.1 Algorithm

TODO.

Theorem 4.1.1. *Given a mutually contradictory pair* (α, β)*, where* α, β *are finite conjunctions of octagonal atoms, the above algorithm terminates with an interpolant* I_α *that is a finite conjunction of octagonal atoms and is equivalent to* $\exists \vec{x}.\alpha$*, where* $\vec{x}$ *is the symbols in* α *which are not in* β*. Further* I_α *is he strongest interpolant for* (α, β)*.*

Proof. First we will prove that $\models \alpha \rightarrow I_\alpha$. The latter follows since the algorithm produces a conjunction of octagonal formulas using the Elim rule, which is a truth-preserving rule of inference, eliminating conjuncts with uncommon symbols.

Now, we will prove that $\not\models I_\alpha \wedge \beta$. We will prove the latter by induction on the number of variables to eliminate (k).

- Base case: $k = 0$. Then the algorithm outputs α. Then the statement holds since (α, β) is unsatisfiable.

- Inductive case: $k = n + 1$. Since the set S of variables to eliminate is non-empty, we just take any variable $x \in S$ and apply the above algorithm to eliminate such variable. Let X be the set of octagonal inequalities of (α, β) and $X^{'}$ be the set of octagonal inequalities $(\alpha^{'}, \beta)$ where $\alpha^{'}$ is the conjunct obtained after removing the variable x. We know X and $X^{'}$ are equisatisfiable using a similar argument as in the Fourier-Motzkin elimination method [24]. Let us suppose $(\alpha^{'}, \beta)$ is satisfiable, hence (α, β) is unsatisfiable as well. But the latter entails a contradiction since (α, β) is assumed to be unsatisfiable. Hence, $(\alpha^{'}, \beta)$ is unsatisfiable. Since $(\alpha^{'}, \beta)$ is an unsatisfiable formula with n variables to eliminate, using the Inductive Hypothesis we conclude that (I_α, β) is unsatisfiable as well.

□

4.2 Implementation

TODO.

4.3 Evaluation

TODO.

Chapter 5

Combining Theories: EUF and Octagonal Formulas

A relevant comment from the paper A combination method for generating interpolants.

The question is how to split it inot two formulas, A' and B'. The condition for splitting is that the common symbols for ¡A', B'¿ should be (a subset of) AB-common symbols, because we would like to use the resultant interpolant for ¡A', B'¿ as a part of an interpolant for the original A and B.

Suppose that Eq contains only AB-pure equalities ...

The latter captures the idea behind equality-interpolating theories. The core definition guarantees that the Nelson-Oppen combination framework is able to propagate AB-pure equalities so the splitting part is not a concern for the algorithm.

TODO.

5.1 Algorithm

TODO.

5.2 Implementation

TODO.

5.3 Evaluation

TODO.

Chapter 6

Future Work

TODO.

Appendices

Bibliography

[1] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* Universitext. Springer Berlin Heidelberg, 2001.

[2] Jerry R. Burch and David L. Dill. Automatic verification of pipelined microprocessor control. In David L. Dill, editor, *Computer Aided Verification*, pages 68–80, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[3] Jürgen Christ, Jochen Hoenicke, and Alexander Nutz. Proof tree preserving interpolation. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 124–138, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[4] William Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *The Journal of Symbolic Logic*, 22(3):269–285, 1957.

[5] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[6] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *The Journal of Logic Programming*, 1(3):267 – 284, 1984.

[7] Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, October 1980.

[8] Herbert B. Enderton. *A mathematical introduction to logic.* Academic Press, 1972.

[9] Alexander Fuchs, Amit Goel, Jim Grundy, Sava Krstić, and Cesare Tinelli. Ground interpolation for the theory of equality. In Stefan Kowalewski and Anna

Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 413–427, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[10] Bernard A. Galler and Michael J. Fisher. An improved equivalence algorithm. *Commun. ACM*, 7(5):301–303, May 1964.

[11] Jean H. Gallier. Fast algorithms for testing unsatisfiability of ground horn clauses with equations. *Journal of Symbolic Computation*, 4(2):233 – 254, 1987.

[12] Deepak Kapur. A new algorithm for computing (strongest) interpolants over quantifier-free theory of equality over uninterpreted symbols. *Manuscript*, 2017.

[13] Yuichi Komori. Logics without craig's interpolation property. *Proc. Japan Acad. Ser. A Math. Sci.*, 54(2):46–48, 1978.

[14] Laura Kovács and Andrei Voronkov. Interpolation and symbol elimination. In Renate A. Schmidt, editor, *Automated Deduction – CADE-22*, pages 199–213, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

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

[16] Shuvendu K. Lahiri and Madanlal Musuvathi. An efficient decision procedure for utvpi constraints. In Bernhard Gramlich, editor, *Frontiers of Combining Systems*, pages 168–183, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[17] K. L. McMillan. An interpolating theorem prover. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 16–30, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[18] Kenneth McMillan. Interpolants from z3 proofs. In *Formal Methods in Computer-Aided Design*, October 2011.

[19] Elliott Mendelson. *Introduction to Mathematical Logic.* Chapman and Hall-CRC, 5th edition, 2009.

[20] Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356–364, April 1980.

[21] Robert Nieuwenhuis and Albert Oliveras. Congruence closure with integer offsets. In Moshe Y. Vardi and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 78–90, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[22] Robert Nieuwenhuis and Albert Oliveras. Proof-producing congruence closure. In Jürgen Giesl, editor, *Term Rewriting and Applications*, pages 453–468, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[23] Andrey Rybalchenko and Viorica Sofronie-Stokkermans. Constraint solving for interpolation. In Byron Cook and Andreas Podelski, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 346–362, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[24] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, Inc., USA, 1986.

[25] Viorica Sofronie-Stokkermans. Interpolation in local theory extensions. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, pages 235–250, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[26] Dirk van Dalen. *Logic and structure (3. ed.).* Universitext. Springer, 1994.

[27] Georg Weissenbacher. Interpolant strength revisited. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, pages 312–326, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[28] Greta Yorsh and Madanlal Musuvathi. A combination method for generating interpolants. In Robert Nieuwenhuis, editor, *Automated Deduction – CADE-20*, pages 353–368, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.