Compactly Representing Uniform Interpolants for $\mathcal{EUF}$ using (conditional) DAGS

Silvio Ghilardi[1], Alessandro Gianola[2], and Deepak Kapur[3]

[1] Dipartimento di Matematica, Università degli Studi di Milano (Italy)
silvio.ghilardi@unimi.it
[2] Faculty of Computer Science, Free University of Bozen-Bolzano (Italy)
gianola@inf.unibz.it
[3] Department of Computer Science, University of New Mexico (USA)
kapur@cs.unm.edu

Abstract

The concept of a uniform interpolant for a quantifier-free formula from a given formula with a list of symbols, while well-known in the logic literature, has been unknown to the formal methods and automated reasoning community. This concept is precisely defined. Two algorithms for computing the uniform interpolant of a quantifier-free formula in $\mathcal{EUF}$ endowed with a list of symbols to be eliminated are proposed. The first algorithm is non-deterministic and generates a uniform interpolant expressed as a disjunction of conjunction of literals, whereas the second algorithm gives a compact representation of a uniform interpolant as a conjunction of Horn clauses. Both algorithms exploit efficient dedicated DAG representation of terms. Correctness and completeness proofs are supplied, using arguments combining rewrite techniques with model-theoretic tools.

1 Introduction

The theory of equality over uninterpreted symbols, henceforth denoted by $\mathcal{EUF}$, is one of the simplest theories which has found numerous applications in computer science, formal methods and logic. Starting with the works of Shostak [26] and Nelson and Oppen [23] in the early 80's, some of the first algorithms were proposed in the context of developing approaches for combining decision procedures for quantifier-free theories including freely constructed data structures and linear arithmetic over the rationals. $\mathcal{EUF}$ was first exploited for hardware verification of pipelined processors by Dill [5] and more widely subsequently in formal methods and verification using model checking framework. With the popularity of SMT solvers where $\mathcal{EUF}$ serves as a glue for combining solvers for different theories, numerous new graph-based algorithms have been proposed in the literature over the last two decades for checking unsatisfiability of a conjunction of equalities and disequalities of terms built using function symbols and constants.

In [22], the use of interpolants for automatic invariant generation was proposed, leading to a mushroom of research activities to develop algorithms for generating interpolants for specific theories as well as their combination. This new application is different from the role of interpolants for analyzing proof theories of various logics starting with the pioneering work of [11, 15, 25] (for a recent survey in the SMT area, see [2, 3]). Approaches like [15, 22, 25] however assume access to a proof of $\alpha \to \beta$ for which interpolant is being generated. Given that there can in general be many interpolants including infinitely many for some theories, little is known about what the kind of interpolants are effective for different applications, even though some research has been reported on the strength and quality of interpolants.

In this paper, a different approach is taken, motivated by the insight connecting interpolating theories with those admitting quantifier-elimination as advocated in [19]. Particularly, the concept of a uniform interpolant (UI) defined by a formula α, in the context of formal methods and verification, is proposed for $\mathcal{EUF}$ which is well-known not to admit quantifier elimination. A uniform interpolant acts as a classical interpolant for any (α, β) such that $\alpha \rightarrow \beta$ (as well as a reverse interpolant for an unsatisfiable pair (α, γ)).[1] A uniform interpolant is defined for theories irrespective of whether they admit quantifier elimination; for theories admitting quantifier elimination, a uniform interpolant can be obtained using quantifier elimination, which can be prohibitively expensive. A UI is shown to exist for $\mathcal{EUF}$ and to be unique. A related concept of a cover is proposed in [14] (see also [8]).

Two different algorithms for generating UIs from a formula in $\mathcal{EUF}$ (with a list of symbols to be eliminated) are proposed with different characteristics. They share a common subpart based on concepts used in a ground congruence closure proposed in [16] which flattens the input and generates a canonical rewrite system on constants along with unique rules of the form $f(\cdots)$, where f is an uninterpreted symbol and arguments are canonical forms of constants. Further, eliminated symbols are represented as a DAG to avoid any exponential blow-up. The first algorithm is non-deterministic where undecided equalities on constants are hypothesized to be true or false, generating a branch in each case, and recursively applying the algorithm. It could also easily be formulated as an algorithm similar in spirit to the use of equality interpolants in Nelson and Oppen framework for combination, where different partitions on constants are tried, with each leading to a branch in the algorithm. New symbols are introduced along each branch to avoid exponential blow-up.

The second algorithm generalizes the concept of a DAG to conditional DAG in which subterms are replaced by new symbols under a conjunction of equality atoms, resulting in its compact and efficient representation. A fully or partially expanded form of a UI can be derived based on their use in applications. Because of their compact representation, UIs can be generated in polynomial times for a large class of formulas.

The termination, correctness and completeness of both the algorithms are proved by using results in model theory about model completions; this relies on a basic result (Lemma 7.1 below) taken from [8].

Both our algorithms are simple, intuitive and easy to understand in contrast to related algorithms in the literature. In fact, the algorithm from [8] requires full saturation in a version of superposition calculus equipped with ad hoc settings, whereas the main merit of our second algorithm is to show that a very light form of completion is sufficient, thus simplifying the whole procedure and getting better complexity results.[2] The algorithm from [14] requires some bug fixes (as pointed out in [8]) and the related completeness proof is still missing.

The use of uniform interpolants in model-checking safety problems for infinite state systems was already mentioned in [14] and further exploited in a recent research line on the verification of data-aware processes [6, 7, 9]. Model checkers need to explore the space of all reachable states of a system; a precise exploration (either forward starting from a description of the initial states or backward starting from a description of unsafe states) requires quantifier elimination. The latter is not always available or might have prohibitive complexity; in addition, it is usually preferable to make overapproximations of reachable states both to avoid divergence and to speed up convergence. One well-established technique for computing overapproximations consists in extracting interpolants from spurious traces, see e.g. [22]; interpolants are used for

[1] The third author recently learned from the first author that this concept has been used extensively in logic for decades [13, 24] to his surprise since he had the erroneous impression that he came up with the concept in 2012, which he presented in a series of talks [17, 18].

[2] Although we feel that some improvement is possible, the termination argument in [8] gives a double exponential bound, whereas we have a simple exponential bound for both algorithms (with optimal chances to keep the output polynomial in the case of the second algorithm).

various symbol elimination tasks in first-order settings [20, 21].One possible advantage of uniform interpolants over ordinary interpolants is that they do not introduce overapproximations and so abstraction/refinements cycles are not needed in case they are employed (the precise reason for that goes through the connection between uniform interpolants, model completeness and existentially closed structures, see [9] for a full account). In this sense, computing uniform interpolants have the same advantages and disadvantages as computing quantifier eliminations, with two remarkable differences. The first difference is that uniform interpolants may be available also in theories not admitting quantifier elimination ($\mathcal{EUF}$ being the typical example); the second difference is that computing uniform interpolants may be tractable when the language is suitably restricted e.g. to unary function symbols (this was already mentioned in [14], see also Remark 3.2 below). Restrictions to unary function symbols is sufficient in database driven verification to encode primary and foreign keys [9]. It is also worth noticing that, precisely by using uniform interpolants for this restricted language, in [9] *new decidability results* have been achieved for interesting classes of infinite state systems. Notably, such results also operationally mirrored in the MCMT [12] implementation since version 2.8.

The paper is structured as follows: in Section 2 we state the main problem, fix some notation, discuss DAG representations and congruence closure. In Sections 3 and 4, we give two algorithms for computing uniform interpolants in $\mathcal{EUF}$ (correctness and completeness of such algorithms are proved in Section 7).The former algorithm is tableaux-shaped and produces the output in disjunctive normal form, whereas the second algorithm is based on manipulation of Horn clauses and gives the output in (compressed) conjunctive normal form. We believe that the two algorithms are in a sense complementary to each others, especially from the point of view of applications. Model checkers typically synthesize safety invariants using conjunctions of clauses and in this sense they might better take profit from the second algorithm; however, model-checkers dually representing sets of backward reachable states ad disjunctions of cubes, would better adopt the first algorithm. Non-deterministic manipulations of cubes is also required to match certain PSPACE lower bounds, as in the case of SAS systems mentioned in [9]. On the other hand, regarding the overall complexity, it seems to be easier to avoid exponential blow-ups in concrete examples by adopting the second algorithm.

2 Preliminaries

We adopt the usual first-order syntactic notions of signature, term, atom, (ground) formula, and so on; our signatures are always *finite* or *countable* and include equality. For simplicity, we only consider functional signatures, i.e. signatures whose only predicate symbol is equality. We compactly represent a tuple $\langle x_1, \dots, x_n \rangle$ of variables as $\underline{x}$. The notation $t(\underline{x}), \phi(\underline{x})$ means that the term t, the formula ϕ has free variables included in the tuple $\underline{x}$. This tuple is assumed to be formed by *distinct variables*, thus we underline that, when we write e.g. $\phi(\underline{x}, \underline{y})$, we mean that the tuples $\underline{x}, \underline{y}$ are made of distinct variables that are also disjoint from each other. A formula is said to be *universal* (resp., *existential*) if it has the form $\forall \underline{x}(\phi(\underline{x}))$ (resp., $\exists \underline{x}(\phi(\underline{x}))$), where ϕ is quantifier-free. Formulae with no free variables are called *sentences*.

From the semantic side, we use the standard notion of Σ-structure $\mathcal{M}$: this is a pair formed of a set (the 'support set', indicated as $|\mathcal{M}|$) and of an interpretation function. The interpretation function maps n-ary function symbols to n-ary operations on $|\mathcal{M}|$ (in particular, constants symbols are mapped to elements of $|\mathcal{M}|$). A free variables assignment $\mathcal{I}$ on $\mathcal{M}$ extends the interpretation function by mapping also variables to elements of $|\mathcal{M}|$; the notion of truth of a formula in a Σ-structure under a free variables assignment $\mathcal{I}$ is the standard one.

It may happen that we need to expand a signature Σ with a fresh name for every $a \in |\mathcal{M}|$:

such expanded signature is named $\Sigma^{|\mathcal{M}|}$ and $\mathcal{M}$ is by abuse seen as a $\Sigma^{|\mathcal{M}|}$-structure itself by interpreting the name of $a \in |\mathcal{M}|$ as a (the name of a is directly indicated as a for simplicity).

A Σ-*theory* T is a set of Σ-sentences; a *model* of T is a Σ-structure $\mathcal{M}$ where all sentences in T are true. We use the standard notation $T \models \phi$ to say that ϕ is true in all models of T for every assignment to the variables occurring free in ϕ. We say that ϕ is T-*satisfiable* iff there is a model $\mathcal{M}$ of T and an assignment to the variables occurring free in ϕ making ϕ true in $\mathcal{M}$.

2.1 Uniform Interpolants

Fix a theory T and an existential formula $\exists \underline{e}\, \phi(\underline{e}, \underline{z})$; call a *residue* of $\exists \underline{e}\, \phi(\underline{e}, \underline{z})$ any quantifier-free formula belonging to the set of quantifier-free formulae

$$Res(\exists \underline{e}\, \phi) \;=\; \{\theta(\underline{z}, \underline{y}) \mid T \models \exists \underline{e}\, \phi(\underline{e}, \underline{z}) \to \theta(\underline{z}, \underline{y})\} \;=\; \{\theta(\underline{z}, \underline{y}) \mid T \models \phi(\underline{e}, \underline{z}) \to \theta(\underline{z}, \underline{y})\}.$$

A quantifier-free formula $\psi(\underline{y})$ is said to be a T-*uniform interpolant*[3] (or, simply, a *uniform interpolant*, abbreviated UI) of $\exists \underline{e}\, \phi(\underline{e}, \underline{z})$ iff $\psi(\underline{z}) \in Res(\exists \underline{e}\, \phi)$ and $\psi(\underline{z})$ implies (modulo T) all the other formulae in $Res(\exists \underline{e}\, \phi)$. It is immediately seen that UI are unique (modulo T-equivalence). We say that a theory T has *uniform quantifier-free interpolation* iff every existential formula $\exists \underline{e}\, \phi(\underline{e}, \underline{z})$ has a UI.

It is clear that if T has uniform quantifier-free interpolation, then it has ordinary quantifier-free interpolation [4], in the sense that if we have $T \models \phi(\underline{e}, \underline{z}) \to \phi'(\underline{z}, \underline{y})$ (for quantifier-free formulae ϕ, ϕ'), then there is a quantifier-free formula $\theta(\underline{y})$ such that $T \models \phi(\underline{e}, \underline{z}) \to \theta(\underline{z})$ and $T \models \theta(\underline{z}) \to \phi'(\underline{z}, \underline{y})$. In fact, if T has uniform quantifier-free interpolation, then the interpolant θ is independent on ϕ' (the same $\theta(\underline{z})$ can be used as interpolant for all entailments $T \models \phi(\underline{e}, \underline{z}) \to \phi'(\underline{z}, \underline{y})$, varying ϕ'). Uniform quantifier-free interpolation has a direct connection to an important notion from classical model theory, namely model completeness (see [8] for more information).

2.2 Problem Statement

In this paper we deal about the problem of *computing UI for the case in which T is pure identity theory in a functional signature* Σ; this theory is called $\mathcal{EUF}(\Sigma)$ or just $\mathcal{EUF}$ in the SMT-LIB2 terminology. We shall provide two different algorithms for that (while proving correctness and completeness of such algorithms, we simultaneously also show that UI exist in $\mathcal{EUF}$). The first algorithm computes a UI in disjunctive normal form format, whereas the second algorithm supplies a UI in conjunctive normal form format. Both algorithm use suitable DAG-compressed representation of formulae.

We use the following notation throughout the paper. Since it is easily seen that UI commute with disjunctions, it is sufficient to compute UI for *primitive* formulae, i.e. for formulae of the kind $\exists \underline{e}\, \phi(\underline{e}, \underline{z})$, where ϕ is a *constraint*, i.e. a conjunction of literals. We partition all the constant symbols from the input as well as symbols newly introduced into disjoint sets. We use the following conventions:

- $\underline{e} = e_0, \ldots, e_N$ are the symbols to be eliminated, called *variables*,
- $\underline{z} = z_0, \ldots, z_M$ are the symbols not to be eliminated, called *parameters*,
- letters $a, b, \ldots$ stand for both variables and parameters.

[3] In some literature [8, 14] uniform interpolants are called *covers*.

Variables $\underline{e}$ are usually skolemized during the manipulations of our algorithms and proofs below, so that they have to be considered as fresh individual constants.

Remark 2.1. *UI computations eliminate symbols which are existentially quantified variables (or skolemized constants).* Elimination of function symbols *can be reduced to elimination of variables in the following way. Consider a formula $\exists f\, \phi(f, \underline{z})$, where ϕ is quantifier-free. Successively abstracting out functional terms, we get that $\exists f\, \phi(f, \underline{z})$ is equivalent to a formula of the kind $\exists \underline{e}\, \exists f (\bigwedge_i (f(\underline{t}_i) = e_i) \wedge \psi)$, where the $\underline{e}$ are fresh variables, f does not occur in $\underline{t}_i, e_i, \psi$ and ψ is quantifier-free. The latter is semantically equivalent to $\exists \underline{e} (\bigwedge_{i \neq j} (\underline{t}_i = \underline{t}_j \to e_i = e_j) \wedge \psi)$, where $\underline{t}_i = \underline{t}_j$ is the conjunction of the component-wise equalities of the tuples $\underline{t}_i$ and $\underline{t}_j$.*

2.3 Flat Literals, DAGs and Congruence Closure

A *flat literal* is a literal of one of the following kinds

$$f(a_1, \dots, a_n) = b, \quad a_1 = a_2, \quad a_1 \neq a_2 \tag{1}$$

where $a_1, \dots, a_n$ are (not necessarily distinct) variables or constants. A formula is flat iff all literals occurring in it are flat; flat terms are terms that may occur in a flat literal (i.e. terms like those appearing in (1)).

We call a *DAG-definition* (or simply a DAG) any formula $\delta(\underline{y}, \underline{z})$ of the following form (let $\underline{y} := y_1 \dots, y_n$)

$$\bigwedge_{i=1}^{n} (y_i = f_i(y_1, \dots, y_{i-1}, \underline{z})) \ .$$

Thus, $\delta(\underline{y}, \underline{z})$ provides in fact an *explicit definition* of the $\underline{y}$ in terms of the parameters $\underline{z}$. To such a DAG δ, is in fact associated the substitution σ_δ recursively defined by the mapping

$$(y_i)\sigma_\delta \ := \ f_i((y_1)\sigma_\delta, \dots, (y_{i-1})\sigma_\delta, \underline{a}).$$

We may sometimes confuse a DAG δ like above with its associated substitution σ_δ. DAGs are commonly used to represent formulae and substitution in compressed form: in fact a formula like

$$\exists \underline{y}\ (\delta(\underline{y}, \underline{a}) \wedge \Phi(\underline{y}, \underline{a})) \tag{2}$$

is equivalent to $\Phi((\underline{y})\sigma_\delta, \underline{a})$, however the full unravelling of such an equivalence causes an exponential blow-up. This is why we shall systematically prefer *DAG-representations* like (2) to their uncompressed forms.

As above stated, our main aim is to compute the UI of a primitive formula $\exists \underline{e}\, \phi(\underline{e}, \underline{z})$; using trivial logical manipulations (that have just linear complexity costs), it can be easily seen that *the constraint ϕ can be assumed to be flat.* In order to do that, it is sufficient to apply well-known Congruence Closure Transformations: the reader is referred to [16] for a full account.

3 The Tableaux Algorithm

The algorithm proposed in this section is tableaux-like. It manipulates formulae in the following *DAG-primitive* format

$$\exists \underline{y}\ (\delta(\underline{y}, \underline{z}) \wedge \Phi(\underline{y}, \underline{z}) \wedge \exists \underline{e}\ \Psi(\underline{e}, \underline{y}, \underline{z})) \tag{3}$$

where $\delta(\underline{y}, \underline{z})$ is a DAG and Φ, Ψ are flat constraints (notice that the $\underline{e}$ do not occur in Φ). To make reading easier, we shall omit in (3) the existential quantifiers, so as (3) will be written simply as

$$\delta(\underline{y}, \underline{z}) \wedge \Phi(\underline{y}, \underline{z}) \wedge \Psi(\underline{e}, \underline{y}, \underline{z}) \quad . \tag{4}$$

Initially the DAG δ and the constraint Φ are the empty conjunction. In the DAG-primitive formula (4), variables $\underline{z}$ are called *parameter* variables, variables $\underline{y}$ are called *(explicitly) defined* variables and variables $\underline{e}$ are called *(truly) quantified* variables. Variables $\underline{z}$ are never modified; in contrast, during the execution of the algorithm it could happen that some quantified variables may disappear or become defined variables (in the latter case they are renamed: a quantified variables e_i becoming defined is renamed as y_j, for a fresh y_j). Below, letters $a, b, \ldots$ range over $\underline{e} \cup \underline{y} \cup \underline{z}$.

Definition 3.1. *A term t (resp. a literal L) is $\underline{e}$-free when there is no occurrence of any of the variables $\underline{e}$ in t (resp. in L). Two flat terms t, u of the kinds*

$$t := f(a_1, \ldots, a_n) \qquad u := f(b_1, \ldots, b_n) \tag{5}$$

are said to be compatible *iff for every $i = 1, \ldots, n$, either a_i is identical to b_i or both a_i and b_i are $\underline{e}$-free. The* difference set *of two compatible terms like above is the set of disequalities $a_i \neq b_i$ such that a_i is not identical to b_i.*

3.1 The Algorithm

Our algorithm applies the transformation below (except the last one) in a "don't care" non-deterministic way. The last transformation has lower priority and splits the execution of the algorithm in several branches: each branch will produce a different disjunct in the output formula. Each state of the algorithm is a dag-primitive formula like (4). We now provide the rules that constitute our 'tableaux-like' algorithm.

(1) $\boxed{\textit{Simplification Rules}:}$

(1.0) if an atom like $t = t$ belongs to Ψ, just remove it; if a literal like $t \neq t$ occurs somewhere, delete Ψ, replace Φ with $\perp$ and stop;

(1.i) If t is not a variable and Ψ contains both $t = a$ and $t = b$, remove the latter and replace it with $a = b$.

(1.ii) If Ψ contains $e_i = e_j$ with $i > j$, remove it and replace everywhere e_i by e_j.

(2) $\boxed{\textit{DAG Update Rule}:}$ if Ψ contains $e_i = t(\underline{y}, \underline{z})$, remove it, rename everywhere e_i as y_j (for fresh y_j) and add $y_j = t(\underline{y})$ to $\delta(\underline{y}, \underline{z})$. More formally:

$$\delta(\underline{y}, \underline{z}) \wedge \Phi(\underline{y}, \underline{z}) \wedge \big(\Psi(\underline{e}, e_i, \underline{y}, \underline{z}) \wedge e_i = t(\underline{y}, \underline{z})\big)$$

$$\Downarrow$$

$$\big(\delta(\underline{y}, \underline{z}) \wedge y_j = t(\underline{y}, \underline{z})\big) \wedge \Phi(\underline{y}, \underline{z}) \wedge \Psi(\underline{e}, y_j, \underline{y}, \underline{z})$$

(3) $\boxed{\underline{e}\textit{-Free Literal Rule}:}$ if Ψ contains a literal $L(\underline{y}, \underline{z})$, move it to $\Phi(\underline{y}, \underline{z})$. More formally:

$$\delta(\underline{y}, \underline{z}) \wedge \Phi(\underline{y}, \underline{z}) \wedge \big(\Psi(\underline{e}, \underline{y}, \underline{z}) \wedge L(\underline{y}, \underline{z})\big)$$

$$\Downarrow$$

$$\delta(\underline{y}, \underline{z}) \wedge \big(\Phi(\underline{y}, \underline{z}) \wedge L(\underline{y}, \underline{z})\big) \wedge \Psi(\underline{e}, \underline{y}, \underline{z})$$

(4) $\boxed{\textit{Splitting Rule}:}$ If Ψ contains a pair of atoms $t = a$ and $u = b$, where t and u are compatible flat terms like in (5), and no disequality from the difference set of t, u belongs to Φ, then non-deterministically apply one of the following alternatives:

- (4.0) remove from Ψ the atom $f(b_1, \dots, b_n) = b$, add it the atom $a = b$ and add to Φ all equalities $a_i = b_i$ such that $a_i \neq b_i$ is in the difference set of t, u;
- (4.1) add to Φ one of the disequalities from the difference set of t, u (notice that the difference set cannot be empty, otherwise Rule (1.i) applies).

When no rule is still applicable, delete $\Psi(\underline{e}, \underline{y}, \underline{z})$ from the resulting formula

$$\delta(\underline{y}, \underline{z}) \wedge \Phi(\underline{y}, \underline{z}) \wedge \Psi(\underline{e}, \underline{y}, \underline{z})$$

so as to obtain for any branch an output formula in DAG-representation of the kind

$$\exists \underline{y}\ (\delta(\underline{y}, \underline{z}) \wedge \Phi(\underline{y}, \underline{z})) \quad .$$

The following proposition states that, by applying the previous rules, termination is always guaranteed.

Proposition 3.1. *The non-deterministic procedure presented above always terminates.*

Proof. It is sufficient to show that every branch of the algorithm must terminate. In order to prove that, first observe that the total number of the variables involved never increases and it decreases if (1.ii) is applied (it might decrease also by the effect of (1.0)). Whenever such a number does not decrease, there is a bound on the number of inequalities that can occur in Ψ, Φ. Now transformation (4.1) decreases the number of inequalities that are actually *missing*; the other transformations do not increase this number. Finally, all transformations except (4.1) reduce the length of Ψ. $\square$

The following remark will be useful to prove the correctness of our algorithm, since it gives a description of the kind of literals contained in a state triple that is *terminal* (i.e., when no rule applies).

Remark 3.1. *Notice that if no transformation applies to (3), the set Ψ can only contain inequalities of the kind $e_i \neq a$, together with equalities of the kind $f(a_1, \dots, a_n) = a$. However, when it contains $f(a_1, \dots, a_n) = a$, one of the a_i must belong to $\underline{e}$ (otherwise (2) or (3) applies). Moreover, if $f(a_1, \dots, a_n) = a$ and $f(b_1, \dots, b_n) = b$ are both in Ψ, then either they are not compatible or $a_i \neq b_i$ belongs to Φ for some i and for some variables a_i, b_i not in $\underline{e}$ (otherwise (4) or (1.i) applies).*

Remark 3.2. *The complexity of the above algorithm is exponential, however the complexity of producing a single branch is quadratic. Notice that if functions symbols are all unary, there is no need to apply Rule 4, hence for this restricted case computing UI is a tractable problem. The case of unary functions has relevant applications in database driven verification [6, 7, 9] (where unary function symbols are used to encode primary and foreign keys).*

Example 3.1. *Let us compute the UI of the formula $\exists e_0\, (g(z_4, e_0) = z_0 \wedge f(z_2, e_0) = g(z_3, e_0) \wedge h(f(z_1, e_0))) = z_0$. Flattening gives the set of literals*

$$g(z_4, e_0) = z_0 \wedge e_1 = f(z_2, e_0) \wedge e_1 = g(z_3, e_0) \wedge e_2 = f(z_1, e_0) \wedge h(e_2) = z_0 \tag{6}$$

where the newly introduced variables e_1, e_2 need to be eliminated too. Applying (4.0) removes $g(z_3, e_0) = e_1$ and introduces the new equalities $z_3 = z_4$, $e_1 = z_0$. This causes e_1 to be renamed as y_1 by (2). Applying again (4.0) removes $f(z_1, e_0) = e_2$ and adds the equalities $z_1 = z_2$, $e_2 = y_1$; moreover, e_2 is renamed as y_2. To the literal $h(y_2) = z_0$ we can apply (3). The branch terminates with $y_1 = z_0 \wedge y_2 = y_1 \wedge z_1 = z_2 \wedge z_3 = z_4 \wedge h(y_2) = z_0 \wedge f(z_2, e_0) = y_1 \wedge g(z_4, e_0) = z_0$. This produces $z_1 = z_2 \wedge z_3 = z_4 \wedge h(z_0) = z_0$ as a first disjunct of the uniform interpolant. The other branches produce $z_1 = z_2 \wedge z_3 \neq z_4$, $z_1 \neq z_2 \wedge z_3 = z_4$ and $z_1 \neq z_2 \wedge z_3 \neq z_4$ as further disjuncts, so that the UI turns out to be equivalent to $z_1 = z_2 \wedge z_3 = z_4 \to h(z_0) = z_0$.

4 The Conditional Algorithm

This section discusses a new algorithm with the objective of generating a compact representation of the UI in $\mathcal{EUF}$ by avoiding having to split based on conditions in Horn clauses generated from literals whose left sides have the same function symbol. A by-product of this approach is that often the UI can be computed in polynomial time for a large class of formulas. Further, the output of this algorithm generates the UI of $\exists \underline{e}\, \phi(\underline{e}, \underline{z})$ (where $\phi(\underline{e}, \underline{z})$ is a conjunction of literals and $\underline{e} = e_0, \ldots, e_N$, $\underline{z} = z_0, \ldots, z_M$, as usual) in conjunctive normal form with literals $f(a_1, \ldots, a_h) = a$, $a = b$, $a \neq b$ and conditional Horn equations. Toward this goal, a new data structure of a conditional DAG, a generalization of DAG, is introduced so as to maximize sharing of sub-formulas.

Using the core preprocessing procedure explained in Subsection 2.3, it is assumed that ϕ is the conjunction $\bigwedge S_1$ of a set of literals S_1 containing only literals of the following two kinds:

$$f(a_1, \ldots, a_h) = a \tag{7}$$

$$a \neq b \tag{8}$$

(recall that we use letters $a, b, \ldots$ for elements of $\underline{e} \cup \underline{z}$). In addition we can assume that variables in $\underline{e}$ must occur in (8) and in the left member of (7).We do not include equalities like $a = b$ because they can be eliminated by replacement.

4.1 The Algorithm

The algorithm requires two steps in order to get a set of clauses representing the output in a suitably compressed format.

$\boxed{\textbf{Step 1}}$. Out of every pair of literals $f(a_1, \ldots, a_h) = a$ and $f(a'_1, \ldots, a'_h) = a'$ of the kind (7) we produce the Horn clause

$$a_1 = a'_1, \ldots, a_h = a'_h \to a = a' \tag{9}$$

Let us call S_2 the set of clauses obtained from S_1 by adding to it these new Horn clauses.

$\boxed{\textbf{Step 2}}$. We saturate S_2 with respect to the following rewriting rule

$$\frac{\Gamma \to e_j = e_i \qquad C}{\Gamma \to C[e_i]_p}$$

where $j > i$, $C[e_i]_p$ means the result of the replacement of e_j by e_i in the position p of the clause C and $\Gamma \to C[e_i]_p$ is the clause obtained by merging Γ with the antecedent of the clause $C[e_i]_p$.

Notice that we apply the rewriting rule only to conditional equalities of the kind $\Gamma \to e_j = e_i$: this is because clauses like $\Gamma \to e_j = z_i$ are considered 'conditional definitions' (and the clauses like $\Gamma \to z_j = z_i$ as 'conditional facts').

We let S_3 be the set of clauses obtained from S_2 by saturating it with respect to the above rewriting rule, by removing from antecedents identical literals of the kind $a = a$ and by removing subsumed clauses.

Example 4.1. *Let S_1 be the set of the following literals*

$$\begin{array}{lll} f_1(e_0, z_1) = e_1, & f_1(e_0, z_2) = z_3, & f_2(e_0, z_4) = e_2, \\ f_2(e_0, z_5) = z_6, & g_1(e_0, e_1) = e_2, & g_1(e_0, z_1') = z_2', \\ g_2(e_0, e_2) = e_1, & g_2(e_0, z_1'') = z_2'' & h(e_1, e_2) = z_0 \end{array}$$

Step 1 produces the following set S_2 of Horn clauses

$$\begin{array}{ll} z_1 = z_2 \to e_1 = z_3, & z_4 = z_5 \to e_2 = z_6, \\ e_1 = z_1' \to e_2 = z_2', & e_2 = z_1'' \to e_1 = z_2'' \end{array}$$

Since there are no Horn clauses whose consequent is an equality of the kind $e_i = e_j$, Step 2 does not produce further clauses and we have $S_3 = S_2$.

4.2 Conditional DAGs

In order to be able to extract the output UI in a plain (uncompressed) format out of the above set of clauses S_3, we must identify all the 'implicit conditional definitions' it contains.

Let $\underline{w}$ be an ordered subset of the $\underline{e} = \{e_1, \dots, e_N\}$: that is, in order to specify $\underline{w}$ we must take a subset of the $\underline{e}$ and an ordering of this subset. Intuitively, these $\underline{w}$ will play the role of *placeholders* inside a conditional definition.

If we let $\underline{w}$ be $w_1, \dots, w_s$ (where, say, w_i is some e_{k_i} with $k_i \in \{1, \dots, N\}$), we let L_i be the language restricted to $\underline{z}$ and $w_1, \dots, w_i$ (for $i \le s$): in other words, an L_i-term or an L_i-clause may contain only terms built up from $\underline{z}, w_1, \dots, w_i$ by applying them function symbols. In particular, L_s (also called $L_{\underline{w}}$) is the language restricted to $\underline{z} \cup \underline{w}$. We let L_0 be the language restricted to $\underline{z}$.

Given a set S of clauses and $\underline{w}$ as above, a $\underline{w}$*-conditional* DAG δ (or simply a conditional DAG δ) built out of S is a set of Horn clauses from S

$$\Gamma_1 \to w_1 = t_1, \ \dots, \ \Gamma_s \to w_s = t_s \tag{10}$$

where Γ_i is a finite tuple of L_{i-1}-atoms and t_i is a L_{i-1}-term. Given a $\underline{w}$-conditional DAG δ we can define the formulae ϕ^i_δ (for $i = 1, \dots, s+1$) as follows:

- ϕ^{s+1}_δ is the conjunction of all $L_{\underline{w}}$-clauses belonging to S;
- for $i \le s$, the formula ϕ^i_δ is $\Gamma_i \to \forall w_i \, (w_i = t_i \to \phi^{i+1}_\delta)$.

It is clear that ϕ^i_δ is equivalent to a quantifier-free L_{i-1} formula,[4] in particular ϕ^1_δ (abbreviated as ϕ_δ) is equivalent to an L_0-quantifier-free formula. The explicit computation of such quantifier-free formulae may however produce an exponential blow-up.

[4]It can be easily seen that such a formula can be turned, again up to equivalence, into a conjunction of Horn clauses.

Example 4.2. *Consider the set S_3 of the Horn clauses mentioned in Example 4.1. We can get not logically equivalent formulae for ϕ_{δ_1} and ϕ_{δ_2} considering δ_1 with $\underline{w}_1 = e_1, e_2$ and conditional definitions $z_1 = z_2 \to e_1 = z_3, \quad e_1 = z'_1 \to e_2 = z'_2$ or δ_2 with $\underline{w}_2 = e_2, e_1$ and conditional definitions $z_4 = z_5 \to e_2 = z_6, \quad e_2 = z''_1 \to e_1 = z''_2$ In fact, ϕ_{δ_1} is logically equivalent to*

$$z_1 = z_2 \wedge z_3 = z'_1 \to \bigwedge S_3^{-0}[z_3/e_1, z'_2/e_2] \quad . \tag{11}$$

whereas ϕ_{δ_2} is logically equivalent to

$$z_4 = z_5 \wedge z_6 = z''_1 \to \bigwedge S_3^{-0}[z_6/e_2, z''_2/e_1] \tag{12}$$

where we used the notation $\bigwedge S_3^{-0}[z_3/e_1, z'_2/e_2]$ to mean the result of the substitution in the conjunction of S_3-clauses not involving e_0 of z_3 for e_1 and of z'_2 for e_2 (a similar notation is used for $S_3^{-0}[z_6/e_2, z''_2/e_1]$). A third possibility is to use the conditional definitions $z_1 = z_2 \to e_1 = z_3$ and $z_4 = z_5 \to e_2 = z_6$ with (equivalently) either $\underline{w}_1$ or $\underline{w}_2$ resulting in a conditional dag δ_3 with ϕ_{δ_3} logically equivalent to

$$z_1 = z_2 \wedge z_4 = z_5 \to \bigwedge S_3^{-0}[z_3/e_1, z_6/e_2] \quad . \tag{13}$$

Next lemma shows the relevant property of the formula ϕ_δ:

Lemma 4.1. *For every set of clauses S and for every $\underline{w}$-conditional DAG δ built out of S, the formula*

$$\bigwedge S \to \phi_\delta$$

is logically valid.

Proof. We prove that $\bigwedge S \to \phi^i_\delta$ is valid by induction on i. The base case is clear. For the case $i \leq s$, proceed e.g. in natural deduction as follows: assume S, Γ_i and $\tilde{w}_i = t_i$ in order to prove $\phi^{i+1}_\delta(\tilde{w}_i/w_i)$. Since $\Gamma_i \to w_i = t_i \in S$, then by implication elimination you get $w_i = t_i$ and also $w_i = \tilde{w}_i$ by transitivity of equality. Now you get what you need from induction hypothesis and equality replacement. □

Notice that it is not true that the conjunction of all possible ϕ_δ (varying δ and w) implies $\bigwedge S$: in fact, such a conjunction can be empty (hence $\top$) in case there is no conditional DAG built up from S at all (this happens for instance if S is just $e_1 = e_2$).

4.3 Extraction of UI's

We shall prove below that in order to get a UI of $\exists \underline{e}\, \phi(\underline{e}, \underline{a})$, one can take *the conjunction of all possible ϕ_δ, varying δ among the conditional DAGs that can be built out of the set of clauses* S_3 from Step 2 of the above algorithm.

Example 4.3. *If ϕ is the conjunction of the literals of Example 4.1, then the conjunction of* (11), (12) *and* (13) *is a UI of $\exists \underline{e}\, \phi$; in fact, no further non-trivial conditional dag δ can be extracted (if we take $\underline{w} = e_1$ or $\underline{w} = e_2$ or $\underline{w} = \emptyset$ to extract δ, then it happens that ϕ_δ is the empty conjunction $\top$).*

Example 4.4. *Let us turn to the literals* (6) *of Example 3.1. Step 1 produces out of them the conditional clauses*

$$z_3 = z_4 \to e_1 = z_0, \qquad z_1 = z_2 \to e_2 = e_1 \quad . \tag{14}$$

Step 2 produces by rewriting the further clauses $z_1 = z_2 \to f(z_1, e_0) = e_1$ *and* $z_1 = z_2 \to h(e_1) = z_0$*. We can extract two conditional dags* δ *(using both the conditional definitions* (14) *or just the first one); in both cases* ϕ_δ *is* $z_1 = z_2 \wedge z_3 = z_4 \to h(z_0) = z_0$*, which is the UI.*

As should be evident from the two examples above, the conditional DAGs representation of the output considerably reduces computational complexity in many cases; this is a clear advantage of the present algorithm over the algorithm from Section 3 and over other approaches like e.g. [8]. Still, next example shows that in some cases the overall complexity remains exponential.

Example 4.5. *Let* $\underline{e}$ *be* $e_0, \dots, e_N$ *and let* $\underline{z}$ *be* $\{z_0, z_0'\} \cup \{z_{i,j}, z_{i,j}' \mid 1 \leq i < j \leq N\}$*. Let* $\phi(\underline{e}, \underline{z})$ *be the conjunction of the identities* $f(e_0, e_1) = z_0$*,* $f(e_0, e_N) = z_0'$ *and the set of identities* $h_{ij}(e_0, z_{ij}) = e_i$*,* $\quad h_{ij}(e_0, z_{ij}') = e_j$*, varying* i, j *such that* $1 \leq i < j \leq N$*. After applying Step 1 of the algorithm presented in Subsection 4.1, we get the Horn clauses* $z_{ij} = z_{ij}' \to e_i = e_j$*, as well as the clause* $e_1 = e_N \to z_0 = z_0'$*. If we now apply Step 2, it is clear that we can never produce a conditional clause of the kind* $\Gamma \to e_i = t$ *with* t *being* $\underline{e}$*-free (because we can only rewrite some* e_i *into some* e_j*). Thus no sequence of clauses like* (10) *can be extracted from* S_3*: notice in fact that the term* t_1 *from such a sequence must not contain the* $\underline{e}$*. In other words, the only* $\underline{w}$*-conditional DAG* δ *that can be extracted is based on the empty* $\underline{w} \subseteq \underline{e}$ *and is empty itself. However such* δ *produces a formula* ϕ_δ*, in fact quite big: it is the conjunction of the exponentially many clauses from* S_3 *where the* $\underline{e}$ *do not occur.*

5 Correctness and Completeness Proofs

In this section we prove correctness and completeness of our two algorithms. To this aim, we need some elementary background, both from model theory and from term rewriting.

For model theory, we refer to [10]. We just recall few definitions. A Σ-*embedding* (or, simply, an embedding) between two Σ-structures $\mathcal{M}$ and $\mathcal{N}$ is a map $\mu : |\mathcal{M}| \longrightarrow |\mathcal{N}|$ among the support sets $|\mathcal{M}|$ of $\mathcal{M}$ and $|\mathcal{N}|$ of $\mathcal{N}$ satisfying the condition ($\mathcal{M} \models \varphi \quad \Rightarrow \quad \mathcal{N} \models \varphi$) for all $\Sigma^{|\mathcal{M}|}$-literals φ ($\mathcal{M}$ is regarded as a $\Sigma^{|\mathcal{M}|}$-structure, by interpreting each additional constant $a \in |\mathcal{M}|$ into itself and $\mathcal{N}$ is regarded as a $\Sigma^{|\mathcal{M}|}$-structure by interpreting each additional constant $a \in |\mathcal{M}|$ into $\mu(a)$). If $\mu : \mathcal{M} \longrightarrow \mathcal{N}$ is an embedding which is just the identity inclusion $|\mathcal{M}| \subseteq |\mathcal{N}|$, we say that $\mathcal{M}$ is a *substructure* of $\mathcal{N}$ or that $\mathcal{N}$ is an *extension* of $\mathcal{M}$.

Extensions and UI are related to each other by the following result we take from [8]:

Lemma 5.1 (Cover-by-Extensions). *A formula* $\psi(\underline{y})$ *is a UI in* T *of* $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ *iff it satisfies the following two conditions:*

(i) $T \models \forall \underline{y}\, (\exists \underline{e}\, \phi(\underline{e}, \underline{y}) \to \psi(\underline{y}))$*;*

(ii) *for every model* $\mathcal{M}$ *of* T*, for every tuple of elements* $\underline{a}$ *from the support of* $\mathcal{M}$ *such that* $\mathcal{M} \models \psi(\underline{a})$ *it is possible to find another model* $\mathcal{N}$ *of* T *such that* $\mathcal{M}$ *embeds into* $\mathcal{N}$ *and* $\mathcal{N} \models \exists \underline{e}\, \phi(\underline{e}, \underline{a})$*.*

To conveniently handle extensions, we need diagrams. Let $\mathcal{M}$ be a Σ-structure. The *diagram* of $\mathcal{M}$ [10], written $\Delta_\Sigma(\mathcal{M})$ (or just $\Delta(\mathcal{M})$), is the set of ground $\Sigma^{|\mathcal{M}|}$-literals that are true in $\mathcal{M}$. An easy but important result, called *Robinson Diagram Lemma* [10], says that, given any Σ-structure $\mathcal{N}$, the embeddings $\mu : \mathcal{M} \longrightarrow \mathcal{N}$ are in bijective correspondence with expansions of $\mathcal{N}$ to $\Sigma^{|\mathcal{M}|}$-structures which are models of $\Delta_\Sigma(\mathcal{M})$. The expansions and the embeddings are related in the obvious way: the name of a is interpreted as $\mu(a)$. It is convenient to see

$\Delta_\Sigma(\mathcal{M})$ as a set of flat literals as follows: the positive part of $\Delta_\Sigma(\mathcal{M})$ contains the $\Sigma^{|\mathcal{M}|}$-equalities $f(a_1, \dots, a_n) = b$ which are true in $\mathcal{M}$ and the negative part of $\Delta_\Sigma(\mathcal{M})$ contains the $\Sigma^{|\mathcal{M}|}$-inequalities $a \neq b$, varying a, b among the pairs of different elements of $|\mathcal{M}|$.

For term rewriting we refer to a textbook like [1]; we only recall the following classical result:

Lemma 5.2. *Let R be a canonical ground rewrite system over a signature Σ. Then there is a Σ-structure $\mathcal{M}$ such that for every pair of ground terms t, u we have that $\mathcal{M} \models t = u$ iff the R-normal form of t is the same as the R-normal form of u. Consequently R is consistent with a set of negative literals S iff for every $t \neq u \in S$ the R-normal forms of t and u are different.*

We are now ready to prove correctness and completeness of our algorithms. We first give the relevant intuitions for the proof technique, which is the same for both cases. By Lemma 7.1 above, what we need to show is that if a model $\mathcal{M}$ satisfies the output formula of the algorithm, then it can be extended to a superstructure $\mathcal{M}'$ satisfying the input formula of the algorithm. By the Diagram Lemma, this is achieved if we show that $\Delta(\mathcal{M})$ is consistent with the output formula of the algorithm. The output formula is equivalent to a disjunction of constraints and the diagram $\Delta(\mathcal{M})$ is also a constraint (albeit infinitary). The positive part of $\Delta(\mathcal{M})$ is a canonical rewriting system (equalities like $f(a_1, \dots, a_n) = a$ are obviously oriented from left-to-right) and every term occurring in $\Delta(\mathcal{M})$ is in normal form. If an algorithm does a good job, it will be easy to see that the completion of the union of $\Delta(\mathcal{M})$ with the relevant disjunct constraint is trivial and does not produce inconsistencies.

5.1 Correctness and Completeness of the Tableaux Algorithm

Theorem 5.1. *Suppose that we apply the algorithm of Subsection 3.1 to the primitive formula $\exists \underline{e}(\phi(\underline{e}, \underline{z}))$ and that the algorithm terminates with its branches in the states*

$$\delta_1(\underline{y}_1, \underline{z}) \wedge \Phi_1(\underline{y}_1, \underline{z}) \wedge \Psi_1(\underline{e}_1, \underline{y}_1, \underline{z}), \qquad \dots, \qquad \delta_k(\underline{y}_k, \underline{z}) \wedge \Phi_k(\underline{y}_k, \underline{z}) \wedge \Psi_k(\underline{e}_k, \underline{y}_k, \underline{z})$$

then the UI of $\exists \underline{e}(\phi(\underline{e}, \underline{z}))$ in $\mathcal{EUF}$ is the DAG-unravelling *of the formula*

$$\bigvee_{i=1}^{k} \exists \underline{y}_i \ (\delta_i(\underline{y}_i, \underline{z}) \wedge \Phi_i(\underline{y}_i, \underline{z})) \quad . \tag{15}$$

Proof. Since $\exists \underline{e}(\phi(\underline{e}, \underline{z}))$ is logically equivalent to $\bigvee_{i=1}^{k} \exists \underline{y}_i \ (\delta_i(\underline{y}_i, \underline{z}) \wedge \Phi_i(\underline{y}_i, \underline{z}) \wedge \exists \underline{e}_i \Psi_i(\underline{e}_1, \underline{y}_1, \underline{z}))$, it is sufficient to check that if a formula like (3) is terminal (i.e. no rule applies to it) then its UI is $\exists \underline{y} \ (\delta(\underline{y}, \underline{z}) \wedge \Phi(\underline{y}, \underline{z}))$. To this aim, we apply Lemma 7.1: we pick a model $\mathcal{M}$ satisfying $\delta(\underline{y}, \underline{z}) \wedge \Phi(\underline{y}, \underline{z})$ via an assignment $\mathcal{I}$ to the variables $\underline{y}, \underline{z}$[5] and we show that $\mathcal{M}$ can be embedded into a model $\mathcal{M}'$ such that, for a suitable extensions $\mathcal{I}'$ of $\mathcal{I}$ to the variables $\underline{e}$, we have that $(\mathcal{M}', \mathcal{I}')$ satisfies also $\Psi(\underline{e}, \underline{y}, \underline{z})$. What we need (by Robinson Diagram Lemma) is to find a model for the following set of literals

$$\Delta(\mathcal{M}) \cup \Psi \cup \{a = \tilde{a}\}_{a \in \underline{y} \cup \underline{z}} \tag{16}$$

where $\tilde{a}$ is the value of a under the assignment $\mathcal{I}$ (here all variables in (23) are seen as constants, so (23) is a set of ground literals). We can orient the equalities in (23) by letting function symbols having bigger precedence over constants and by letting a having bigger precedence over $\tilde{a}$. Normalizing (23) replaces a with $\tilde{a}$ in Ψ, call $\tilde{\Psi}$ the resulting set of literals. Now

[5]Actually the values of the assignment $\mathcal{I}$ to the $\underline{z}$ uniquely determines the values of $\mathcal{I}$ to the $\underline{y}$.

we claim that all oriented equalities in $\Delta(\mathcal{M}) \cup \tilde{\Psi} \cup \{a = \tilde{a}\}_{a \in \underline{y} \cup \underline{z}}$ form a canonical rewriting system. This is due to Remark 3.1: in fact (let $\tilde{e}_i$ be e_i for all $e_i \in \underline{e}$) if $f(a_1, \ldots, a_n)$ and $f(b_1, \ldots, b_n)$ both occur in Ψ, it cannot happen that $f(\tilde{a}_1, \ldots, \tilde{a}_n)$ and $f(\tilde{b}_1, \ldots, \tilde{b}_n)$ are the same term because the $\underline{e}$ are already in normal form and because $\mathcal{M} \models \Phi$ (and hence also $\mathcal{M} \models \tilde{\Phi}$). In addition, if $f(a_1, \ldots, a_n)$ occurs in Ψ, then one of the a_i belongs to $\underline{e}$, hence rules from $\tilde{\Psi}$ and $\Delta(\mathcal{M})$ cannot superpose. Since all oriented equalities in $\Delta(\mathcal{M}) \cup \tilde{\Psi} \cup \{a = \tilde{a}\}_{a \in \underline{y} \cup \underline{z}}$ form a canonical rewriting system, the inequalities in $\Delta(\mathcal{M}) \cup \tilde{\Psi}$ are in normal form and we can apply Lemma 7.2 to get the desired $\mathcal{M}'$. ☐

5.2 Correctness and Completeness of the Conditional Algorithm

To justify the algorithm of Subsection 4.1, we first need a Lemma:

Lemma 5.3. *If the clauses $\Gamma \to f(a_1, \ldots, a_h) = b$ and $\Gamma' \to f(a'_1, \ldots, a'_h) = b'$ both belong to S_3 and b is not the same term as b', then S_3 contains also a clause subsuming the clause*

$$\Gamma, \Gamma', a_1 = a'_1, \ldots, a_h = a'_h \to b = b'$$

Proof. By induction on the number K of applications of the rewriting rule of Step 2 needed to derive $\Gamma \to f(a_1, \ldots, a_h) = b$ and $\Gamma' \to f(a'_1, \ldots, a'_h) = b'$. If K is 0, the claim is clear by the instruction of Step 1. Suppose that $K > 0$ and let say $\Gamma' \to f(a'_1, \ldots, a'_h) = b'$ be obtained from $\Gamma_1 \to e_i = e_j$ by rewriting e_j to e_i from some clause C. We need to distinguish cases depending on the position p of the rewriting. All cases being treated in the same way,[6] suppose for instance that p is in the antecedent, so that C is $\Gamma_2 \to f(a'_1, \ldots, a'_h) = b'$ and that $\Gamma' \to f(a'_1, \ldots, a'_h) = b'$ is $\Gamma_1, \Gamma_2[e_i]_p \to f(a'_1, \ldots, a'_h) = b'$. Then by induction hypothesis S_3 contains a clause subsuming

$$\Gamma, \Gamma_2, a_1 = a'_1, \ldots, a_h = a'_h \to b = b'$$

and rewriting with $\Gamma_1 \to e_i = e_j$ produces

$$\Gamma, \Gamma_1, \Gamma_2[e_i]_p, a_1 = a'_1, \ldots, a_h = a'_h \to b = b'$$

as required. ☐

Theorem 5.2. *Let S_3 be obtained as in Steps 1-2 from $\exists \underline{e}\, \phi(\underline{e}, \underline{z})$. Then the conjunction of all possible ϕ_δ (varying δ among the conditional DAGs that can be built out of S_3) is a UI of $\exists \underline{e}\, \phi(\underline{e}, \underline{z})$ in $\mathcal{EUF}$.*

Proof. We use Lemma 7.1. Condition (i) of that Lemma is ensured by Lemma 4.1 above because $\bigwedge S_3$ is logically equivalent to ϕ. So let us take a model $\mathcal{M}$ and elements $\underline{\tilde{a}}$ from its support such that we have $\mathcal{M} \models \bigwedge_\delta \phi_\delta$ under the assignment of the $\underline{\tilde{a}}$ to the parameters $\underline{z}$. We need to expand it to a superstructure $\mathcal{N}$ in such a way that we have $\mathcal{N} \models \bigwedge S_1$, under some assignment to $\underline{z}, \underline{e}$ extending the assignment $\underline{z} \mapsto \underline{\tilde{a}}$ (recall that $\bigwedge S_1$ is logically equivalent to ϕ too). From now on, we consider the assignment $\underline{z} \mapsto \underline{\tilde{a}}$ fixed, so that when we write $\mathcal{M} \models C$ for a clause $C(\underline{z})$ we mean that $\mathcal{M} \models C$ holds under the assignment $\underline{z} \mapsto \underline{\tilde{a}}$.

Notice that every $\underline{w}$-conditional DAG δ extracted from S_3 (let it be given by the clauses (10)) is naturally equipped with a substitution σ_δ which is given in DAG form by $w_1 \mapsto t_1, \ \ldots, \ w_s \mapsto t_s$. We say that δ is *realized* in $\mathcal{M}$ iff we have that

$$\mathcal{M} \models \bigwedge \Gamma_1 \sigma_\delta, \ldots, \mathcal{M} \models \bigwedge \Gamma_s \sigma_\delta$$

[6] If b' is the same term as b and we rewrite it, then b' is e_j and $\Gamma_1 \to e_i = e_j$ is the clause we are looking for.

Let δ be a $\underline{w}$-conditional DAG which is realized in $\mathcal{M}$ and let it be maximal with this property (a $\underline{w}$-conditional DAG δ is said to be bigger than a $\underline{w}'$-conditional DAG δ' iff $\underline{w}$ includes $\underline{w}'$ - the inclusion is as sets, the order is disregarded). Since $\mathcal{M} \models \phi_\delta$ and δ is realized in $\mathcal{M}$, it is clear that all $L_{\underline{w}}$-clauses from S_3 (hence also all $L_{\underline{w}}$-literals from S_1) are true in $\mathcal{M}$. Let $\underline{u} = u_1, \dots, u_k$ be the variables from $\underline{e} \setminus \underline{w}$ and let $S_{\underline{u}}$ be the literals from S_1 which are not $L_{\underline{w}}$-literals. What we need (by Robinson Diagram Lemma) is to find a model for the following set of literals

$$\Delta(\mathcal{M}) \cup S_{\underline{u}} \cup \{w_i = \tilde{b}_i \mid i = 1, \dots, s\} \cup \{z_i = \tilde{a}_i \mid i = 1, \dots, M\} \tag{17}$$

where $\tilde{b}_i$ is the value of $w_i\sigma_\delta$ under the assignment $\underline{z} \mapsto \underline{\tilde{a}}$. We orient the functional equalities in (24) from left to right and the equalities $w_i = \tilde{b}_i$ also from left to right. The $\underline{e}$ are ordered as $e_1 > \cdots > e_N$ and are bigger than the constants naming the elements of $|\mathcal{M}|$; function symbols are bigger than constant symbols. We show that the ground Knuth-Bendix completion of (24) cannot produce any inconsistent literal of the kind $t \neq t$ (this completes the proof of the Theorem).

First notice that the rules $\{w_i = \tilde{b}_i \mid i = 1, \dots, s\} \cup \{z_i = \tilde{a}_i \mid i = 1, \dots, M\}$ simply eliminates the $\underline{w}$ and the $\underline{z}$ from $S_{\underline{u}}$ (they become inactive after such normalization steps). Let $\tilde{S}_{\underline{u}}$ be the set of equalities resulting after this elimination. It turns out that $\tilde{S}_{\underline{u}}$ can only contains equalities of the kinds

$$f(a_1, \dots, a_h) = a \tag{18}$$

$$u_j \neq a \tag{19}$$

where $a_1, \dots a_h, a$ can be either among the $\underline{u}$ or constants naming elements of $|\mathcal{M}|$. However some of the $\underline{u}$ must be among $a_1, \dots, a_h$ for each equality of the kind (25) because atoms not containing the $\underline{u}$ are removed by $\Delta(\mathcal{M})$ and atoms like $u_i = t$ (where t does not contain any of the $\underline{u}$) cannot be there because δ is maximal. During completion, in addition to these kinds of atoms, only atoms of the kind

$$u_i = u_j \tag{20}$$

can possibly be produced. This is a consequence of next Lemma. Below we say that a tuple of atoms Γ is *realized in* $\mathcal{M}$ iff the Γ are $L_{\underline{w}}$-atoms and $\mathcal{M} \models \bigwedge \Gamma\sigma_\delta$; similarly we say that a literal Θ is *conditionally realized in* $\mathcal{M}$ if there exists Γ realized in $\mathcal{M}$ with $\Gamma \to \Theta \in S_3$ (if Θ is a negative literal, $\Gamma \to \Theta$ stands for $\Gamma, \neg\Theta \to$).

Lemma 5.4. *Suppose that a literal Λ is produced during the completion of $\tilde{S}_{\underline{u}}$. Then it must be of the kinds* (25)*,* (26)*,* (27)*. Moreover there exists a literal Λ' such that (i) Λ' is conditionally realized in $\mathcal{M}$; (ii) Λ is obtained from Λ' by rewriting $\underline{z}, \underline{w}$ respectively to $\underline{\tilde{a}}, \underline{\tilde{b}}$.*

Proof. By straightforward case analysis; we analyze the most interesting case given by the superposition of two rules of the kind (25). Suppose that

$$f(a_1, \dots, a_h) = a \;\text{ and }\; f(a_1, \dots, a_h) = b$$

produce the equality $a = b$. Then by induction hypothesis, there are in S_3 two clauses like

$$\Gamma' \to f(a'_1, \dots, a'_h) = a', \quad \Gamma'' \to f(a''_1, \dots, a''_h) = a''$$

with Γ', Γ'' realized in $\mathcal{M}$, with $a'_1, \dots, a'_h, a'$ rewritable (using $\underline{z}, \underline{w} \mapsto \underline{\tilde{a}}, \underline{\tilde{b}}$) to $a_1, \dots, a_h, a$, respectively, and with $a''_1, \dots, a''_h, a''$ also rewritable (using $\underline{z}, \underline{w} \mapsto \underline{\tilde{a}}, \underline{\tilde{b}}$) to $a_1, \dots, a_h, b$, respectively. By Lemma 7.3, S_3 contains a clause subsuming

$$\Gamma', \Gamma'', a'_1 = a''_1, \dots, a'_h = a''_h \to a' = a'' \tag{21}$$

which is as required because $\Gamma', \Gamma'', a'_1 = a''_1, \dots, a'_h = a''_h$ is realized in $\mathcal{M}$ and $a' = a''$ rewrites (using $\underline{z}, \underline{w} \mapsto \underline{\tilde{a}}, \underline{\tilde{b}}$) to $a = b$. It remains to check that $a = b$ is of the kind (27). If both a', a'' taken from the consequent of (28) belong to $\underline{z} \cup \underline{w}$, then since the antecedent of (28) is realized in $\mathcal{M}$ and (28) belongs to S_3, a and b must be the same element from $|\mathcal{M}|$, so that $a = b$ is a trivial identity (which does not enter into the completion). It cannot be that only one between a' and a'' belongs to $\underline{z} \cup \underline{w}$ (the other one being from $\underline{u}$) because δ is maximal among conditional DAGs realized by $\mathcal{M}$ and thus it cannot be properly enlarged by adding it the additional conditional definition which would be supplied by (28). Thus it must be the case that both a', a'' are from $\underline{u}$, which implies that they cannot be rewritten (using $\underline{z}, \underline{w} \mapsto \underline{\tilde{a}}, \underline{\tilde{b}}$), so that a' is a, a'' is b and $a = b$ is of the kind (27). $\square$

Proof of Theorem 7.2 (continued). Once $\tilde{S}_{\underline{u}}$ (standing alone) is completed, only literals of the kinds (25), (26), (27) are produced. No completion inference is possible between literals of the kinds (25), (26), (27) on one side and literals from $\Delta(\mathcal{M}) \cup \{w_i = \tilde{b}_i \mid i = 1, \dots, s\} \cup \{z_i = \tilde{a}_i \mid i = 1, \dots, M\}$ on the other side; hence the completion of $\tilde{S}_{\underline{u}}$ alone, once joined to $\Delta(\mathcal{M}) \cup \{w_i = \tilde{b}_i \mid i = 1, \dots, s\} \cup \{z_i = \tilde{a}_i \mid i = 1, \dots, M\}$ yields a completion of (24). The only possible inconsistencies that can arise are given by literals of the kind $u_i \neq u_i$. Suppose that indeed one such a literal $u_i \neq u_i$ is produced during the completion of $\tilde{S}_{\underline{u}}$. Applying the above lemma, there should be in S_3 a clause like $\Gamma, u_i = u_i \to$ (i.e. after simplification, a clause like $\Gamma \to$) with Γ being realized in $\mathcal{M}$. The latter means that $\mathcal{M} \models \bigwedge \Gamma\sigma_\delta$. This cannot be, because $\Gamma \to$ is a $L_{\underline{w}}$-clause from S_3: in fact, we have that $\mathcal{M} \models \phi_\delta$ and that δ is realized in $\mathcal{M}$, which imply that $\mathcal{M} \models C\sigma_\delta$ holds for every $L_{\underline{w}}$-clause from S_3 by the definition of ϕ_δ. In particular, we should have $\mathcal{M} \models \neg \bigwedge \Gamma\sigma_\delta$, taking $\Gamma \to$ as C. $\square$

6 Conclusions

Two different algorithms for computing uniform interpolants (UI) from a formula in $\mathcal{EUF}$ with a list of symbols to be eliminated are presented. They share a common subpart as well as they are different in their overall objectives. The first algorithm is non-deterministic and generates a UI expressed as a disjunction of conjunction of literals, whereas the second algorithm gives a compact representation of a UI as a conjunction of Horn clauses. The output of both algorithms needs to be expanded if a fully (or partially) expanded uniform interpolant is needed for an application. This restriction/feature is similar in spirit for syntactic unification where also, efficient unification algorithms never produce output in fully expanded form to avoid an exponential blow-up.

For generating a compact representation of the UI, both algorithms make use of DAG representation of terms by introducing new symbols to stand for subterms arising the full expansion of the UI. In addition, the second algorithm uses a conditional DAG, a new data structure introduced in the paper, to represent subterms under conditions.

The complexity of the algorithms is also analyzed. It is shown that while the first algorithm generates exponentially many branches with each branch of at most quadratic length, the complexity of the second algorithm is typically polynomial, even though its worse case is exponential. A fully expanded UI however can be of exponential size thus needing exponentially many steps. An implementation of both the algorithms, along with a comparative study are planned as a future activity. In parallel with the implementation, a characterization of classes of formulae for which computation of UI requires polynomial time in our algorithms (especially in the second one) needs further investigation.

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, United Kingdom, 1998.

[2] M. P. Bonacina and M. Johansson. Interpolation systems for ground proofs in automated deduction: a survey. *J. Autom. Reasoning*, 54(4):353–390, 2015.

[3] M. P. Bonacina and M. Johansson. On interpolation in automated theorem proving. *J. Autom. Reasoning*, 54(1):69–97, 2015.

[4] R. Bruttomesso, S. Ghilardi, and S. Ranise. Quantifier-free interpolation in combinations of equality interpolating theories. *ACM Trans. Comput. Log.*, 15(1):5:1–5:34, 2014.

[5] J. R. Burch and D. L. Dill. Automatic verification of pipelined microprocessor control. In D. L. Dill, editor, *Computer Aided Verification, 6th International Conference, CAV '94, Stanford, California, USA, June 21-23, 1994, Proceedings*, volume 818 of *Lecture Notes in Computer Science*, pages 68–80. Springer, 1994.

[6] D. Calvanese, S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin. Formal modeling and SMT-based parameterized verification of data-aware BPMN. In *Proc. of BPM*, volume 11675 of *LNCS*, pages 157–175. Springer, 2019.

[7] D. Calvanese, S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin. From model completeness to verification of data aware processes. In *Description Logic, Theory Combination, and All That*, volume 11560 of *LNCS*, pages 212–239. Springer, 2019.

[8] D. Calvanese, S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin. Model completeness, covers and superposition. In *Proc. of CADE*, volume 11716 of *LNCS (LNAI)*, pages 142–160. Springer, 2019.

[9] D. Calvanese, S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin. SMT-based verification of data-aware processes: a model-theoretic approach. *Mathematical Structures in Computer Science*, 2020.

[10] C.-C. Chang and J. H. Keisler. *Model Theory.* North-Holland Publishing Co., Amsterdam-London, third edition, 1990.

[11] W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symbolic Logic*, 22:269–285, 1957.

[12] S. Ghilardi and S. Ranise. MCMT: A model checker modulo theories. In *Proc. of IJCAR*, pages 22–29, 2010.

[13] S. Ghilardi and M. Zawadowski. *Sheaves, games, and model completions*, volume 14 of *Trends in Logic—Studia Logica Library.* Kluwer Academic Publishers, Dordrecht, 2002. A categorical approach to nonclassical propositional logics.

[14] S. Gulwani and M. Musuvathi. Cover algorithms and their combination. In *Proc. of ESOP, Held as Part of ETAPS*, pages 193–207, 2008.

[15] G. Huang. Constructing craig interpolation formulas. In *Computing and Combinatorics* COCOON, pages 181–190. LNCS, 959, 1995.

[16] D. Kapur. Shostak's congruence closure as completion. In *Rewriting Techniques and Applications, 8th International Conference, RTA-97, Sitges, Spain, June 2-5, 1997, Proceedings*, pages 23–37, 1997.

[17] D. Kapur. Nonlinear polynomials, interpolants and invariant generation for system analysis. In *Proc. of the 2nd International Workshop on Satisfiability Checking and Symbolic Computation co-located with ISSAC*, 2017.

[18] D. Kapur. Conditional congruence closure over uninterpreted and interpreted symbols. *J. Systems Science & Complexity*, 32(1):317–355, 2019.

[19] D. Kapur, R. Majumdar, and C. G. Zarba. Interpolation for data structures. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2006, Portland, Oregon, USA, November 5-11, 2006*, pages 105–116, 2006.

[20] L. Kovács and A. Voronkov. Interpolation and symbol elimination. In *Proc. of CADE*, pages 199–213, 2009.

[21] L. Kovács and A. Voronkov. First-order interpolation and interpolating proof systems. In *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, pages 49–64, 2017.

[22] K. L. McMillan. Lazy abstraction with interpolants. In *Proc. of CAV*, pages 123–136, 2006.

[23] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.

[24] A. M. Pitts. On an interpretation of second order quantification in first order intuitionistic propositional logic. *J. Symb. Log.*, 57(1):33–52, 1992.

[25] P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.

[26] R. E. Shostak. Deciding combinations of theories. *J. ACM*, 31(1):1–12, 1984.