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

Model Completeness, Uniform Interpolants and Superposition Calculus

(with Applications to Verification of Data-Aware Processes)

Diego Calvanese · Silvio Ghilardi · Alessandro Gianola · Marco Montali · Andrey Rivkin

Received: date / Accepted: date

Abstract Uniform interpolants have been largely studied in non-classical propositional logics since the nineties; a successive research line within the automated reasoning community investigated uniform quantifier-free interpolants (sometimes referred to as "covers") in first-order theories. This further research line is motivated by the fact that uniform interpolants offer an effective solution to tackle quantifier elimination and symbol elimination problems, which are central in model checking infinite state systems. This was first pointed out in ESOP 2008 by Gulwani and Musuvathi, and then by the authors of the present contribution in the context of recent applications to the verification of data-aware processes. In this paper, we show how covers are strictly related to model completions, a well-known topic in model theory. We also investigate the computation of covers within the Superposition Calculus, by adopting a constrained version of the calculus and by defining appropriate settings and reduction strategies. In addition, we show that computing covers is computationally tractable for the fragment of the language used when tackling the verification of data-aware processes. This observation is confirmed by analyzing the preliminary results obtained using the MCMT tool to verify relevant examples of data-aware process. These examples can be found in the last version of the tool distribution.

D. Calvanese
Faculty of Computer Science, Free University of Bozen-Bolzano (Italy)
E-mail: calvanese@inf.unibz.it

S. Ghilardi
Dipartimento di Matematica, Università degli Studi di Milano (Italy)
E-mail: silvio.ghilardi@unimi.it

A. Gianola
Faculty of Computer Science, Free University of Bozen-Bolzano (Italy)
E-mail: gianola@inf.unibz.it

M. Montali
Faculty of Computer Science, Free University of Bozen-Bolzano (Italy)
E-mail: montali@inf.unibz.it

A. Rivkin
Faculty of Computer Science, Free University of Bozen-Bolzano (Italy)
E-mail: rivkin@inf.unibz.it

Keywords Covers · Uniform Interpolation · Model Completeness · Superposition Calculus · Verification of Data-Aware Processes

1 Introduction

Uniform interpolants were originally studied in the context of non-classical logics, starting from the pioneering work by Pitts [56]. We briefly recall what uniform interpolants are. We fix a logic or a theory T and a suitable fragment L (propositional, first-order quantifier-free, etc.) of its language. Given an L-formula $\phi(\underline{x}, \underline{y})$ (where $\underline{x}, \underline{y}$ are the variables occurring free in ϕ), a *uniform interpolant* of ϕ (w.r.t. $\underline{y}$) is an L-formula $\phi'(\underline{x})$ where only variables $\underline{x}$ occur free, and that satisfies the following two properties: *(i)* $\phi(\underline{x}, \underline{y}) \vdash_T \phi'(\underline{x})$; *(ii)* for any further L-formula $\psi(\underline{x}, \underline{z})$ such that $\phi(\underline{x}, \underline{y}) \vdash_T \psi(\underline{x}, \underline{z})$, we have $\phi'(\underline{x}) \vdash_T \psi(\underline{x}, \underline{z})$. Whenever uniform interpolants exist, one can compute an interpolant for an entailment like $\phi(\underline{x}, \underline{y}) \vdash_T \psi(\underline{x}, \underline{z})$ in such a way that this interpolant is *independent* of ψ.

The existence of uniform interpolants is an exceptional phenomenon, which is however not so infrequent; it has been extensively studied in non-classical logics starting from the nineties, as witnessed by a large literature, including for instance [59, 63, 33, 35, 34, 22, 7, 37, 44]). The main results from the above papers are that uniform interpolants exist for intuitionistic logic and for some modal systems (like the Gödel-Lob system and the S4.Grz system); they do not exist for instance in $S4$ and $K4$, whereas for the basic modal system K they exist for the local consequence relation but not for the global consequence relation.

In the last decade, also the automated reasoning community developed an increasing interest in uniform interpolants, with particular focus on quantifier-free fragments of first-order theories. This is witnessed by various talks and drafts by Kapur presented in many conferences and workshops (FloC 2010, ISCAS 2013-14, SCS 2017 [42]), as well as by the paper presented in ESOP 2008 authored by Gulwani and Musuvathi [38]. In this last paper uniform interpolants were renamed as *covers*, a terminology we shall adopt in this paper too. In these contributions, examples of cover computations were supplied and also some algorithms were sketched. The first formal *proof* about the existence of covers in $\mathcal{EUF}$ was however published only in [14] by the present authors; such a proof was equipped with powerful semantic tools (see the Cover-by-Extensions Lemma 3.1 below) obtained thanks to interesting connections with model completeness [57], and came with an algorithm for computing covers that is based on a constrained variant of the Superposition Calculus [55]. Both the model-theoretic tools and the algorithm are detailed in the present paper. Two simple additional algorithms, which exploit DAG representations of terms, are studied in [26, 27].

The usefulness of covers in model checking was already stressed in [38] and further motivated by our recent line of research on the verification of data-aware processes [13, 12, 17, 15]. Notably, this is also operationally mirrored in the MCMT model checker [32] starting from version 2.8. The need for incorporating this algorithm within MCMT is due to the following reason. Declarative approaches to infinite state model checking [58] need to manipulate logical formulae in order to represent sets of reachable states. To prevent divergence, various abstraction strategies have been adopted, ranging from interpolation-based [48] to sophisticated search via counterexample elimination [39]. Precise computations of the set

of reachable states require some form of quantifier elimination and hence are subject to two problems, namely that quantifier elimination might not be available at all and that, when available, it is computationally very expensive. To cope with the first problem, Gulwani and Musuvathi [38] introduced the notion of *cover* and showed that covers can be used as an alternative to quantifier elimination and yield a precise computation of reachable states. Concerning the second problem, again in [38] it was observed (as a side remark) that computing the cover of a conjunction of literals becomes tractable when only free unary function symbols occur in the signature. We show here (see Section 6 below) that the same observation applies when also free relational symbols occur.

In [11,13] we propose a new formalism for representing *read-only database schemas* towards the verification of integrated models of processes and data [10] (called data-aware processes henceforth), in particular so-called *artifact systems* [62, 20,45,8]; this formalism (briefly recalled in Section 4.1 below) precisely uses signatures comprising unary function symbols and free n-ary relations. In [11,13,17] we apply model completeness techniques for verifying transition systems based on read-only databases, in a framework where such systems employ both individual and higher order variables.

In this paper we show (see Section 3 below) that covers (alias uniform interpolants) are strictly related to *model completions*, thus creating a bridge that links different research areas. In particular, we prove that computing covers for a theory is *equivalent* to eliminating quantifiers in its model completion. This connection reproduces, in a first-order setting, an analogous well-known connection for propositional logics: the connection between propositional uniform interpolants and model completions of equational theories axiomatizing the varieties corresponding to propositional logics, which was first stated in [36] and further developed in [33,37,44]. Interestingly, model completeness has other well-known applications in computer science. It has been applied: *(i)* to reveal interesting connections between temporal logic and monadic second order logic [29,30]; *(ii)* in automated reasoning to design complete algorithms for constraint satisfiability in combined theories over non-disjoint signatures [23,1,31,52,50,51]; *(iii)* again in automated reasoning in relationship with interpolation and symbol elimination [60,61]; *(iv)* in modal logic and in software verification theories [24,25], to obtain combined interpolation results.

This paper is organized as follows. After some preliminaries in Section 2, we first state the formal connection between uniform quantifier-free interpolation and model completions in Section 3. Then, in Section 4 we report our applications (mostly taken from [17]) concerning verification of data-aware processes. We begin the second part of the paper by proving (Section 5 below) that covers for $\mathcal{EUF}$ can be computed through a constrained version of the *Superposition Calculus* [55] equipped with appropriate settings and reduction strategies; the related completeness proof requires a careful analysis of the constrained literals generated during the saturation process. Complexity bounds for the fragment used in data-aware processes verification are investigated in Section 6; an extension of our constrained Superposition Calculus that handles a schema of additional constraints (useful for our applications) is provided in Section 7; in Section 8 we give some details about our first implementation in our tool MCMT. This paper is the extended version of [14]: apart from containing more basic preliminary material, a thorough account of model-checking applications, full proofs and detailed exam-

ples, in Sections 6 and 7 this paper covers additional new results on complexity analysis and extensions.

2 Preliminaries

We adopt the usual first-order syntactic notions of signature, term, atom, (ground) formula, and so on; our signatures are multi-sorted and include equality for every sort. This implies that variables are sorted as well. For simplicity, most basic definitions in this section will be supplied for single-sorted languages only. However, the adaptation to multi-sorted languages is straightforward: for example, a multi-sorted signature Σ must contain not only constant, function and relation symbols, but also sorts. We compactly represent a tuple $\langle x_1, \ldots, x_n \rangle$ 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 $\underline{x}$. Our tuples are assumed to be formed by *distinct variables*, thus we underline that, writing 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.

We assume that a function arity can be deduced from the context. Whenever we build terms and formulae, we always assume that they are well-typed, in the sense that the sorts of variables, constants, and function sources/targets match. 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 a quantifier-free formula. Formulae with no free variables are called *sentences*.

From the semantic side, we use the standard notion of a Σ-structure $\mathcal{M}$ and of truth of a formula in a Σ-structure under a free variables assignment. The *support* of a structure $\mathcal{M}$ is the disjoint union of the interpretations of the Σ-sorts in $\mathcal{M}$ and is indicated with $|\mathcal{M}|$.

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 the 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 $\mathcal{M}$.

A Σ-formula ϕ is a *Σ-constraint* (or just a constraint) iff it is a conjunction of literals, i.e. of atomic formulae and their negations. The *constraint satisfiability problem* for T is the following: we are given a constraint (equivalently, a quantifier-free formula) $\phi(\underline{x})$ and we are asked whether there exist a model $\mathcal{M}$ of T and an assignment $\mathcal{I}$ to the free variables $\underline{x}$ such that $\mathcal{M}, \mathcal{I} \models \phi(\underline{x})$.

A theory T has *quantifier elimination* iff for every formula $\phi(\underline{x})$ in the signature of T there is a quantifier-free formula $\phi'(\underline{x})$ such that $T \models \phi(\underline{x}) \leftrightarrow \phi'(\underline{x})$. It is well-known (and easily seen) that quantifier elimination holds in case we can eliminate quantifiers from *primitive* formulae, i.e. from formulae of the kind $\exists \underline{y}\, \phi(\underline{x}, \underline{y})$, where ϕ is a constraint. Since we are interested in effective computability, we assume that when we talk about quantifier elimination, an effective procedure for eliminating quantifiers is given.

We recall also some basic definitions and notions from model theory. Let Σ be a first-order signature. The signature obtained from Σ by adding to it a set $\underline{a}$ of new constants (i.e., 0-ary function symbols) is denoted by $\Sigma^{\underline{a}}$. Analogously, given a Σ-structure $\mathcal{M}$, the signature Σ can be expanded to a new signature $\Sigma^{|\mathcal{M}|} := \Sigma \cup \{\bar{a} \mid a \in |\mathcal{M}|\}$ by adding a set of new constants $\bar{a}$ (the *name* for a), one for each element a in $\mathcal{M}$, with the convention that two distinct elements are

denoted by different "name" constants. $\mathcal{M}$ can be expanded to a $\Sigma^{|\mathcal{M}|}$-structure just interpreting the additional constants over the corresponding elements. From now on, we confuse $\mathcal{M}$ and this expanded structure and we do not distinguish from an element of $|\mathcal{M}|$ and its name. Thus we employ notations like $\mathcal{M} \models \phi(\underline{a})$ to mean that the sentence $\phi(\underline{a})$ (obtained by replacing the free variables $\underline{x}$ of $\phi(\underline{x})$ by the names of some tuple $\underline{a}$ from $|\mathcal{M}|$) is true in $\mathcal{M}$, once $\mathcal{M}$ is canonically expanded to a $\Sigma^{|\mathcal{M}|}$-structure as explained above. Notice that this is the same as saying that $\phi(\underline{x})$ is true in $\mathcal{M}$ under the assignment mapping the $\underline{x}$ to the $\underline{a}$.

A Σ-*embedding* [18] (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 that 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 $\mathcal{M}$. We recall that a substructure *preserves* and *reflects* validity of ground formulae, in the following sense: given a Σ-substructure $\mathcal{M}_1$ of a Σ-structure $\mathcal{M}_2$, a ground $\Sigma^{|\mathcal{M}_1|}$-sentence θ is true in $\mathcal{M}_1$ iff θ is true in $\mathcal{M}_2$.

Let $\mathcal{M}$ be a Σ-structure. The *diagram* of $\mathcal{M}$, 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* [18], 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: $\bar{a}$ is interpreted as $\mu(a)$. The typical use of the Robinson Diagram Lemma is the following: suppose we want to show that some structure $\mathcal{M}$ can be embedded into a structure $\mathcal{N}$ in such a way that some set of sentences Θ are true. Then, by the Lemma, this turns out to be equivalent to the fact that the set of sentences $\Delta(\mathcal{M}) \cup \Theta$ is consistent: thus, the Diagram Lemma can be used to transform an *embeddability* problem into a *consistency* problem (the latter is a problem of a logical nature, to be solved for instance by appealing to the compactness theorem for first-order logic).

Amalgamation is a classical algebraic concept. We give the formal definition:

Definition 2.1 (Amalgamation). *A theory T has the* amalgamation property *if for every couple of embeddings $\mu_1 : \mathcal{M}_0 \longrightarrow \mathcal{M}_1$, $\mu_2 : \mathcal{M}_0 \longrightarrow \mathcal{M}_2$ among models of T, there exists a model $\mathcal{M}$ of T endowed with embeddings $\nu_1 : \mathcal{M}_1 \longrightarrow \mathcal{M}$ and $\nu_2 : \mathcal{M}_2 \longrightarrow \mathcal{M}$ such that $\nu_1 \circ \mu_1 = \nu_2 \circ \mu_2$.* ◁

3 Covers, Uniform Interpolation and Model Completions

We report the notion of *cover* taken from [38]. Fix a theory T and an existential formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$; call a *residue* of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ any quantifier-free formula belonging to the set of quantifier-free formulae

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

A quantifier-free formula $\psi(\underline{y})$ is said to be a T-*cover* (or, simply, a *cover*) of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ iff $\psi(\underline{y}) \in Res(\exists \underline{e}\, \phi)$ and $\psi(\underline{y})$ implies (modulo T) all the other formulae in $Res(\exists \underline{e}\, \phi)$. Notice that the cover is unique, modulo T-equivalence. Alternatively, $\psi(\underline{y})$ is also said to be a T-*uniform (quantifier-free) interpolant* of $\phi(\underline{e}, \underline{y})$. The following Lemma (to be widely used throughout the paper) supplies a semantic counterpart to the notion of a cover:

Lemma 3.1 (Cover-by-Extensions). *A formula $\psi(\underline{y})$ is a T-cover of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ iff it satisfies the following two conditions: (i) $T \models \forall \underline{y}\, (\exists \underline{e}\, \phi(\underline{e}, \underline{y}) \rightarrow \psi(\underline{y}))$; (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 $\mathcal{N} \models \exists \underline{e}\, \phi(\underline{e}, \underline{a})$.* $\triangleleft$

Proof. Suppose that $\psi(\underline{y})$ satisfies conditions (i) and (ii) above. Condition (i) says that $\psi(\underline{y}) \in Res(\exists \underline{e}\, \phi)$, so ψ is a residue. In order to show that ψ is also a cover, we have to prove that $T \models \forall \underline{y}, \underline{z} (\psi(\underline{y}) \rightarrow \theta(\underline{y}, \underline{z}))$, for every $\theta(\underline{y}, \underline{z})$ that is a residue for $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$. Given a model $\mathcal{M}$ of T, take a pair of tuples $\underline{a}, \underline{b}$ of elements from $|\mathcal{M}|$ and suppose that $\mathcal{M} \models \psi(\underline{a})$. By condition (ii), there is a model $\mathcal{N}$ of T such that $\mathcal{M}$ embeds into $\mathcal{N}$ and $\mathcal{N} \models \exists \underline{e}\, \phi(\underline{e}, \underline{a})$. Using the definition of $Res(\exists \underline{e}\, \phi)$, we have $\mathcal{N} \models \theta(\underline{a}, \underline{b})$, since $\theta(\underline{y}, \underline{z}) \in Res(\exists \underline{x}\, \phi)$. Since $\mathcal{M}$ is a substructure of $\mathcal{N}$ and θ is quantifier-free, $\mathcal{M} \models \theta(\underline{a}, \underline{b})$ as well, as required.

Suppose that $\psi(\underline{y})$ is a cover. The definition of residue implies condition (i). To show condition (ii) we have to prove that, given a model $\mathcal{M}$ of T, for every tuple $\underline{a}$ of elements from $|\mathcal{M}|$, if $\mathcal{M} \models \psi(\underline{a})$, then there exists a model $\mathcal{N}$ of T such that $\mathcal{M}$ embeds into $\mathcal{N}$ and $\mathcal{N} \models \exists \underline{x}\, \phi(\underline{x}, \underline{a})$. Using Robinson Diagram Lemma, we can reformulate the latter embeddability statement into a consistency statement: so what we need to prove is that $\Delta(\mathcal{M}) \cup \{\exists \underline{x}\, \phi(\underline{x}, \underline{a})\}$ is a T-consistent $\Sigma^{|\mathcal{M}|}$-set of sentences (Σ is the signature of T). By reduction to absurdity, suppose that this is not the case: by compactness, there is a finite number of literals $\ell_1(\underline{a}, \underline{b}), ..., \ell_m(\underline{a}, \underline{b})$ (for some tuple $\underline{b}$ of elements from $|\mathcal{M}|$) such that $\mathcal{M} \models \ell_i(\underline{a}, \underline{b})$ (for all $i = 1, \ldots, m$) and

$$(*) \quad T \models \exists \underline{e}\, \phi(\underline{e}, \underline{a}) \rightarrow \neg(\ell_1(\underline{a}, \underline{b}) \wedge \cdots \wedge \ell_m(\underline{a}, \underline{b})) \quad .$$

Now, the constants $\underline{a}, \underline{b}$ do not occur in the axioms of T and do not belong to Σ, hence we can replace them by variables $\underline{y}, \underline{z}$ in the T-proof witnessing $(*)$: indeed, since they do not occur in the axioms of T, they are generic from the point of view of T. As a consequence, we then get

$$T \models \exists \underline{e}\, \phi(\underline{e}, \underline{y}) \rightarrow (\neg \ell_1(\underline{y}, \underline{z}) \vee \cdots \vee \neg \ell_m(\underline{y}, \underline{z})).$$

By definition of residue, clearly $(\neg \ell_1(\underline{y}, \underline{z}) \vee \cdots \vee \neg \ell_m(\underline{y}, \underline{z})) \in Res(\exists \underline{x}\, \phi)$; then, since $\psi(\underline{y})$ is a cover, $T \models \psi(\underline{y}) \rightarrow (\neg \ell_1(\underline{y}, \underline{z}) \vee \cdots \vee \neg \ell_m(\underline{y}, \underline{z}))$. Replacing back the variables $\underline{y}, \underline{z}$ by the constants $\underline{a}, \underline{b}$ and recalling that $\mathcal{M} \models \psi(\underline{a})$, this implies that $\mathcal{M} \models \neg \ell_j(\underline{a}, \underline{b})$ for some $j = 1, \ldots, m$, which is a contradiction. Thus, $\psi(\underline{y})$ satisfies conditions (ii) too. $\dashv$

We say that a theory T has *uniform quantifier-free interpolation* iff every existential formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ (equivalently, every primitive formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$) has a T-cover. It is clear that if T has uniform quantifier-free interpolation, then it has ordinary quantifier-free interpolation [9], in the sense that if we have $T \models \phi(\underline{e}, \underline{y}) \rightarrow \phi'(\underline{y}, \underline{z})$ (for quantifier-free formulae ϕ, ϕ'), then there is a quantifier-free formula $\theta(\underline{y})$ such that $T \models \phi(\underline{e}, \underline{y}) \rightarrow \theta(\underline{y})$ and $T \models \theta(\underline{y}) \rightarrow \phi'(\underline{y}, \underline{z})$. In fact, if T has uniform quantifier-free interpolation, then the interpolant θ is independent on ϕ': indeed, the same $\theta(\underline{y})$ can be used as interpolant for all entailments $T \models \phi(\underline{e}, \underline{y}) \rightarrow \phi'(\underline{y}, \underline{z})$, varying ϕ'.

We say that a *universal* theory T has a *model completion* iff there is a stronger theory $T^* \supseteq T$ (still within the same signature Σ of T) such that (i) every Σ-constraint that is satisfiable in a model of T is satisfiable in a model of T^*; (ii) T^* eliminates quantifiers. Other equivalent definitions are possible [18]: for instance,

(i) is equivalent to the fact that T and T^* prove the same quantifier-free formulae or again to the fact that every model of T can be embedded into a model of T^*. We recall that the model completion, if it exists, is unique and that its existence implies the amalgamation property for T [18]. The relationship between uniform interpolation in a propositional logic and the model completion of the equational theory of the variety algebraizing it was extensively studied in [33]. In the context of first order theories, we prove an even more direct connection:

Theorem 3.2. *Suppose that T is a universal theory. Then T has a model completion T^* iff T has uniform quantifier-free interpolation. If this happens, T^* is axiomatized by the infinitely many sentences*

$$\forall \underline{y} \, (\psi(\underline{y}) \rightarrow \exists \underline{e} \, \phi(\underline{e}, \underline{y})) \tag{1}$$

where $\exists \underline{e} \, \phi(\underline{e}, \underline{y})$ is a primitive formula and ψ is a cover of it. ◁

Proof. Suppose first that there is a model completion T^* of T and let $\exists \underline{e} \, \phi(\underline{e}, \underline{y})$ be a primitive formula. Since T^* eliminates quantifiers, we have $T^* \models \exists \underline{e} \, \phi(\underline{e}, \underline{y}) \leftrightarrow \psi(\underline{y})$ for some quantifier-free formula $\psi(\underline{y})$. Since T and T^* prove the same quantifier-free formulae, from the left-to-right side $T^* \models \phi(\underline{e}, \underline{y}) \rightarrow \psi(\underline{y})$ we have that $\psi(\underline{y}) \in Res(\exists \underline{e} \, \phi)$. If $\theta(\underline{y}, \underline{z}) \in Res(\exists \underline{e} \, \phi)$, then we have $T \models \phi(\underline{e}, \underline{y}) \rightarrow \theta(\underline{y}, \underline{z})$; the same entailment holds in T^* too, where we have $T^* \models \psi(\underline{y}) \rightarrow \theta(\underline{y}, \underline{z})$. Since $\psi(\underline{y}) \rightarrow \theta(\underline{y}, \underline{z})$ is quantifier-free, we have also $T \models \psi(\underline{y}) \rightarrow \theta(\underline{y}, \underline{z})$, showing that ψ is a cover of $\exists \underline{e} \, \phi(\underline{e}, \underline{y})$. Thus T has uniform interpolation, because we found a cover for every primitive formula.

Suppose vice versa that T has uniform interpolation. Let T^* be the theory axiomatized by all the formulae (1) above. From (i) of Lemma 3.1 and (1) above, we clearly get that $T^\star$ admits quantifier elimination: in fact, in order to prove that a theory enjoys quantifier elimination, it is sufficient to eliminate quantifiers from *primitive* formulae (then the quantifier elimination for all formulae can be easily shown by an induction over their complexity). This is exactly what is guaranteed by (i) of Lemma 3.1 and (1).

Let $\mathcal{M}$ be a model of T. By using a chain argument [17] (see [18], Lemma 3.5.7 for an almost identical construction), we show that there exists a model $\mathcal{M}'$ of $T^\star$ such that $\mathcal{M}$ embeds into $\mathcal{M}'$. Consider the set of all pairs $(\underline{a}, \exists \underline{e} \, \phi(\underline{e}, \underline{a}))$ where $\underline{a}$ is a tuple from $|\mathcal{M}|$, $\exists \underline{e} \, \phi(\underline{e}, \underline{y})$ is a primitive formula and $\mathcal{M} \models \psi(\underline{a})$ (here ψ is a cover of ϕ). By Zermelo's Theorem, the set of such pairs $(\underline{a}, \exists \underline{e} \, \phi(\underline{e}, \underline{a}))$ can be well-ordered: let $\{(\underline{a}_i, \exists \underline{e}_i \, \phi_i(\underline{e}_i, \underline{a}_i))\}_{i \in I}$ be such a well-ordered set of pairs, where I is some ordinal.[1] By transfinite induction on this well-order, we define $\mathcal{M}_0 := \mathcal{M}$ and, for each $i \in I$, $\mathcal{M}_i$ as an extension of $\bigcup_{j<i} \mathcal{M}_j$ such that $\mathcal{M}_i \models \exists \underline{e}_i \, \phi_i(\underline{e}_i, \underline{a}_i)$, which exists for (ii) of Lemma 3.1 since $\bigcup_{j<i} \mathcal{M}_j \models \psi_i(\underline{a}_i)$ (remember that validity of ground formulae is preserved passing through substructures and superstructures, and $\mathcal{M}_0 \models \psi_i(\underline{a}_i)$).

Now we take the chain union $\mathcal{M}^1 := \bigcup_{i \in I} \mathcal{M}_i$: since T is universal, $\mathcal{M}^1$ is again a model of T. Thanks to this construction, we added, for every pair $(\underline{a}_i, \exists \underline{e}_i \, \phi_i(\underline{e}_i, \underline{a}_i))$ (with $\underline{a}_i \in \mathcal{M}$ and $\mathcal{M} \models \psi_i(\underline{a}_i)$), a corresponding tuple $\underline{b}_i$ such that $\mathcal{M}^1 \models \phi_i(\underline{b}_i, \underline{a}_i)$; however, this only guarantees that such a tuple $\underline{b}_i$ exists for every pair $(\underline{a}_i, \exists \underline{e}_i \, \phi_i(\underline{e}_i, \underline{a}_i))$ such that the tuple $\underline{a}_i$ is from $|\mathcal{M}|$, whereas nothing is said for the pairs where the tuple $\underline{a}$ is in $|\mathcal{M}^1| \setminus |\mathcal{M}|$. Then, we iteratively repeat the chain construction above for these new $\underline{a}$. Indeed, it is possible to construct,

[1] I is possibly different from ω (there can be uncountably many tuples $\underline{a}_i$).

by an analogous chain argument, a model $\mathcal{M}^2$ as done above, starting from $\mathcal{M}^1$ instead of $\mathcal{M}$. Clearly, we get $\mathcal{M}_0 := \mathcal{M} \subseteq \mathcal{M}^1 \subseteq \mathcal{M}^2$ by construction.

At this point, we iterate the same argument countably many times, so as to define a new chain of models of T:

$$\mathcal{M}_0 := \mathcal{M} \subseteq \mathcal{M}^1 \subseteq ... \subseteq \mathcal{M}^n \subseteq ...$$

Defining $\mathcal{M}' := \bigcup_n \mathcal{M}^n$, we trivially get that $\mathcal{M}'$ is a model of T such that $\mathcal{M} \subseteq \mathcal{M}'$ and satisfies all the sentences of type (1): the last fact is immediate, recalling that truth of ground formulae (in expanded languages with names from support sets) is preserved by substructures and extensions. After ω steps we are done, because every tuple $\underline{a} \in |\mathcal{M}'|$ occurs after finitely many steps, and its corresponding $\underline{b}$ in the construction are added at the immediately subsequent step.$\dashv$

To sum up, Theorem 3.2 states that, thanks to Formulae (1), the T-uniform interpolant (or cover) ψ of the formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ is exactly the T^*-equivalent quantifier-free formula that *eliminates* the quantified variables $\underline{e}$ from $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$: this means that *computing covers in T* is equivalent to *eliminating quantifiers in its model completion T^**.

4 Model-Checking Applications

In this section we supply old and new motivations for investigating covers and model completions in view of model-checking applications. We first report the considerations from [38,11,13,17] on symbolic model-checking via model completions (or, equivalently, via covers) in the basic case where system variables are represented as individual variables; for more advanced applications where system variables are both individual and higher order variables, see [11,13,17]. Similar ideas (i.e., 'to use quantifier elimination in the model completion even if T does not allow quantifier elimination') were used in [60] for interpolation and symbol elimination.

Definition 4.1. *A (quantifier-free)* transition system *is a tuple*

$$\mathcal{S} \;=\; \langle \Sigma, T, \underline{x}, \iota(\underline{x}), \tau(\underline{x}, \underline{x}') \rangle$$

where: (i) Σ is a signature and T is a Σ-theory; (ii) $\underline{x} = x_1, \ldots, x_n$ are individual variables; (iii) $\iota(\underline{x})$ is a quantifier-free formula; (iv) $\tau(\underline{x}, \underline{x}')$ is a quantifier-free formula (here the $\underline{x}'$ are renamed copies of the $\underline{x}$). $\triangleleft$

A *safety* formula for a transition system $\mathcal{S}$ is a further quantifier-free formula $\upsilon(\underline{x})$ describing undesired states of $\mathcal{S}$. We say that $\mathcal{S}$ is *safe with respect to* υ if the system has no finite run leading from ι to υ, i.e. (formally) if there is no model $\mathcal{M}$ of T and no $k \geq 0$ such that the formula

$$\iota(\underline{x}^0) \wedge \tau(\underline{x}^0, \underline{x}^1) \wedge \cdots \wedge \tau(\underline{x}^{k-1}, \underline{x}^k) \wedge \upsilon(\underline{x}^k) \tag{2}$$

is satisfiable in $\mathcal{M}$ (here $\underline{x}^i$'s are renamed copies of $\underline{x}$). The *safety problem* for $\mathcal{S}$ is the following: *given υ, decide whether $\mathcal{S}$ is safe with respect to υ.*

Suppose now that the theory T mentioned in Definition 4.1 (i) is universal, has decidable constraint satisfability problem and admits a model completion T^*. Algorithm 1 describes the *backward reachability algorithm* for handling the safety problem for $\mathcal{S}$ (the dual algorithm working via forward search is described in equivalent terms in [38]). An integral part of the algorithm is to compute preimages. For that purpose, for any $\varphi_1(\underline{x}, \underline{x}')$ and $\varphi_2(\underline{x})$ (where $\underline{x}'$ are renamed copies of $\underline{x}$), we define $Pre(\varphi_1, \varphi_2)$ to be the formula $\exists \underline{x}'(\varphi_1(\underline{x}, \underline{x}') \wedge \varphi_2(\underline{x}'))$.

The *preimage* of the set of states described by a state formula $\phi(\underline{x})$ is the set of states described by $Pre(\tau, \phi)$. The subprocedure $\mathsf{QE}(T^*, \phi)$ in Line 6 applies the quantifier elimination algorithm of T^* to the existential formula ϕ. Without the application of this subprocedure, the existential prefix generated by the computation of preimages would grow in an unlimited way and some decidability results (see, e.g., the locally finite case mentioned below) would be compromized. Algorithm 1 computes iterated preimages of υ (storing their disjunction into the variable B) and applies to them quantifier elimination, until a fixpoint is reached or until a set intersecting the initial states (i.e., satisfying ι) is found. *Inclusion* (Line 2) and *disjointness* (Line 3) tests produce proof obligations that can be discharged because T has decidable constraint satisfiability problem.

Algorithm 1: Backward reachability algorithm

```
   Function BReach(υ)
1  |  φ ⟵ υ; B ⟵ ⊥;
2  |  while φ ∧ ¬B is T-satisfiable
   |   do
3  |  |  if ι ∧ φ is T-satisfiable.
   |  |   then
   |  |  |  return unsafe
4  |  |  B ⟵ φ ∨ B;
5  |  |  φ ⟵ Pre(τ, φ);
6  |  |  φ ⟵ QE(T*, φ);
   |  |  return (safe, B);
```

The proof of Proposition 4.2 (which is a slight variant of a similar result for Simple Artifact Systems (SASs) in [17]) consists just in the observation that the formulae (2) are quantifier-free and that a quantifier-free formula is satisfiable in a model of T iff so is it in a model of T^*: thus, if an unsafe trace exists at all, it arises in a model of T^*, so that the subprocedure $\mathsf{QE}(T^*, \phi)$ in Line 6 does not introduce overapproximations and consequently no spurious trace can be produced during the search performed by our algorithm.

Proposition 4.2. *Suppose that the universal Σ-theory T has decidable constraint satisfiability problem and admits a model completion T^*. For every transition system $\mathcal{S} = \langle \Sigma, T, \underline{x}, \iota, \tau \rangle$, the backward search algorithm is effective and partially correct for solving safety problems for $\mathcal{S}$.*[2] ◁

Despite its simplicity, Proposition 4.2 is a crucial fact. Notice that it implies decidability of the safety problems in some interesting cases: this happens, for instance, when in T there are only finitely many quantifier-free formulae in which $\underline{x}$ occur, as in case T has a purely relational signature or, more generally, T is *locally finite*[3]. Since a theory is universal iff it is closed under substructures [18] and since a universal locally finite theory has a model completion iff it has the amalgamation property [64,46], it follows that Proposition 4.2 can be used to cover the decidability result stated in Theorem 5 of [8] (once restricted to transition systems over a first-order definable class of Σ-structures).

4.1 Database Schemas

In this subsection, we provide a new application for the above explained model-checking techniques [13,17]. The application relates to the verification of integrated models of business processes and data [10], referred to as artifact systems [62], where the behavior of the process is influenced by data stored in a relational

[2] *Partial correctness* means that, when the algorithm terminates, it gives a correct answer. *Effectiveness* means that all subprocedures in the algorithm can be effectively executed.

[3] For our purposes, it is convenient to define a theory T to be locally finite iff for every finite tuple of variables $\underline{x}$ there are only finitely many T-equivalence classes of atoms $A(\underline{x})$ involving only the variables $\underline{x}$.

database (DB) with constraints. The data contained therein are read-only: they can be queried by the process and stored in a working memory, which in the context of this paper is constituted by a set of system variables. In this context, safety amounts to checking whether the system never reaches an undesired property, irrespectively of what is contained in the read-only DB.

We define next the two key notions of (read-only) DB schema and instance, by relying on an algebraic, functional characterization.

Definition 4.3. *A* DB schema *is a pair $\langle \Sigma, T \rangle$, where: (i) Σ is a* DB signature*, that is, a finite multi-sorted signature whose function symbols are all unary; (ii) T is a* DB theory*, that is, a set of universal Σ-sentences.* ◁

Given a DB signature Σ, we denote by Σ_{srt} the set of sorts, by Σ_{fun} the set of functions in Σ and by Σ_{rel} the set of relations in Σ. We associate to a DB signature Σ a characteristic (directed) graph $G(\Sigma)$ capturing the dependencies induced by functions over sorts. Specifically, $G(\Sigma)$ is an edge-labeled graph whose set of nodes is Σ_{srt}, and with a labeled edge $S \xrightarrow{f} S'$ for each $f : S \longrightarrow S'$ in Σ_{fun}. We say that Σ is *acyclic* if $G(\Sigma)$ is so. The *leaves* of Σ are the nodes of $G(\Sigma)$ without outgoing edges. These terminal sorts are divided in two subsets, respectively representing *unary relations* and *value sorts*. Non-value sorts (i.e., unary relations and non-leaf sorts) are called *id sorts*, and are conceptually used to represent (identifiers of) different kinds of objects. Value sorts, instead, represent datatypes such as strings, numbers, clock values, etc. We denote the set of id sorts in Σ by Σ_{ids}, and that of value sorts by Σ_{val}, hence $\Sigma_{srt} = \Sigma_{ids} \uplus \Sigma_{val}$.

Before giving the formal definition of DB instance, we show an interesting example of DB signature inspired by concrete business processes.

Example 4.1 ([17]). The human resource (HR) branch of a company stores the following information inside a relational database: *(i)* users registered to the company website, who are potentially interested in job positions offered by the company; *(ii)* the different, available job categories; *(iii)* employees belonging to HR, together with the job categories they are competent in (in turn indicating which job applicants they could interview). To formalize these different aspects, we make use of a DB signature Σ_{hr} consisting of: *(i)* four id sorts, used to respectively identify users, employees, job categories, and the competence relationship connecting employees to job categories; *(ii)* one value sort containing strings used to name users and employees, and describe job categories. In addition, Σ_{hr} contains five function symbols mapping: *(i)* user identifiers to their corresponding names; *(ii)* employee identifiers to their corresponding names; *(iii)* job category identifiers to their corresponding descriptions; *(iv)* competence identifiers to their corresponding employees and job categories. The characteristic graph of Σ_{hr} is shown in Figure 1 (left part). ◁

We now focus on extensional data conforming to a given DB schema.

Definition 4.4. *A* DB instance *of DB schema $\langle \Sigma, T \rangle$ is a Σ-structure $\mathcal{M}$ such that $\mathcal{M}$ is a model of T.*[4] ◁

We respectively denote by $S^{\mathcal{M}}$, $f^{\mathcal{M}}$, and $c^{\mathcal{M}}$ the interpretation in $\mathcal{M}$ of the sort S (this is a set), of the function symbol f (this is a set-theoretic function),

[4] One may restrict to models interpreting sorts as *finite* sets, as customary in database theory. Since the theories we are dealing with usually have finite model property for constraint satisfiability, assuming such restriction turns out to be irrelevant, as far as safety problems are concerned (see [11,13] for an accurate discussion).

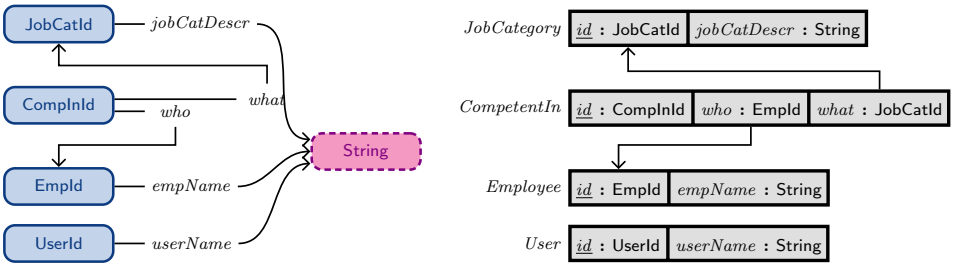

Fig. 1 On the left: characteristic graph of the human resources DB signature from Example 4.1. On the right: relational view of the DB signature; each cell denotes an attribute with its type, underlined attributes denote primary keys, and directed edges capture foreign keys.

and of the constant c (this is an element of the interpretation of the corresponding sort). Obviously, $f^{\mathcal{M}}$ and $c^{\mathcal{M}}$ must match the sorts declared in Σ. For instance, if the source and the target of f are, respectively, S and U, then the function $f^{\mathcal{M}}$ has domain $S^{\mathcal{M}}$ and range $U^{\mathcal{M}}$.

One might be surprised by the fact that signatures in our DB schemas contain unary function symbols, beside relational symbols. As shown in [11,13,17], the algebraic, functional characterization of DB schema and instance can be actually reinterpreted in the classical, relational model so as to reconstruct the requirements posed in [45]. In this last work, the schema of the *read-only* database must satisfy the following conditions: *(i)* each relation schema has a single-attribute primary key; *(ii)* attributes are typed; *(iii)* attributes may be foreign keys referencing other relation schemas; *(iv)* the primary keys of different relation schemas are pairwise disjoint. We now discuss why these requirements are matched by DB schemas.

Definition 4.3 naturally corresponds to the definition of relational database schema with single-attribute *primary and foreign keys*. To see this, we adopt the *named perspective*, where each relation schema is defined by a signature containing a *relation name* and a set of *typed attribute names*. Let $\langle \Sigma, T \rangle$ be a DB schema. Each sort S from Σ corresponds to a dedicated relation R_S with the following attributes: *(i)* one identifier attribute id_S with type S; *(ii)* one dedicated attribute a_f with type S' for every function symbol f from Σ of the form $f : S \longrightarrow S'$.

The fact that R_S is constructed starting from functions in Σ naturally induces corresponding functional dependencies within R_S, and inclusion dependencies from R_S to other relation schemas. In particular, for each non-id attribute a_f of R_S, we get a functional dependency from id_S to a_f. Altogether, such dependencies witness that id_S is the *primary key* of R_S. In addition, for each non-id attribute a_f of R_S whose corresponding function symbol f has id sort S' as image, we get an inclusion dependency from a_f to the id attribute $id_{S'}$ of $R_{S'}$. This captures that a_f is a *foreign key* referencing $R_{S'}$. This view is shown in the following example.

Example 4.2. The diagram on the right in Figure 1 graphically depicts the relational view corresponding to the DB signature of Example 4.1. ◁

Given a DB instance $\mathcal{M}$ of $\langle \Sigma, T \rangle$, its corresponding relational instance $\mathcal{R}[\mathcal{M}]$ is the minimal set satisfying the following property: for every id sort S from Σ, let $f_1, \ldots, f_n$ be all functions in Σ with domain $S^{\mathcal{M}}$; then, for every identifier $\mathsf{o} \in S^{\mathcal{M}}$, $\mathcal{R}[\mathcal{M}]$ contains a *labeled fact* of the form $R_S(id_S : \mathsf{o}^{\mathcal{M}}, a_{f_1} : f_1(\mathsf{o})^{\mathcal{M}}, \ldots, a_{f_n} : f_n(\mathsf{o})^{\mathcal{M}})$, where $attr : \mathsf{c}^{\mathcal{M}}$ means that the element $\mathsf{c}^{\mathcal{M}}$ corresponds to the attribute $attr$ of

the relation R_S. In addition, $\mathcal{R}[\mathcal{M}]$ contains the tuples from $r^{\mathcal{M}}$, for every relational symbol r from Σ (these relational symbols represent plain relations, i.e. those not possessing a key).

We close our discussion by focusing on DB theories. Notice that $\mathcal{EUF}$ suffices to handle the sophisticated setting of artifact systems from [13,17] (e.g., key dependencies). The role of a non-empty DB theory is to encode background axioms to express additional constraints. We illustrate a typical background axiom, required to handle the possible presence of *undefined identifiers/values* in the different sorts. This, in turn, is essential to capture artifact systems whose working memory is initially undefined, in the style of [21,45]. To accommodate this, we add to every sort S of Σ a constant $\mathtt{undef}_S$ (written by abuse of notation just `undef` from now on), used to specify an undefined value. Then, for each function symbol f of Σ, we can impose additional constraints involving `undef`, for example by adding the following axioms to the DB theory:

$$\forall x \ (x = \mathtt{undef} \leftrightarrow f(x) = \mathtt{undef}) \tag{3}$$

This axiom states that the application of f to the undefined value produces an undefined value, and it is the only situation for which f is undefined. A slightly different approach may handle *many* undefined values for each sort; the reader is referred to [11,13,17] for examples of concrete database instances formalized in our framework. We just point out that in most cases the kind of axioms that we need for our DB theories T are just *one-variable universal axioms* (like Axioms 3), so that they fit the hypotheses of Proposition 4.5 below.

We are interested in applying the algorithm of Proposition 4.2 to a (nondeterministic) version of *Simple Artifact Systems (SASs)* [17], i.e. transition systems $\mathcal{S} = \langle \Sigma, T, \underline{x}, \iota(\underline{x}), \tau(\underline{x}, \underline{x}') \rangle$, where $\langle \Sigma, T \rangle$ is a DB schema in the sense of Definition 4.3. To this aim, it is sufficient to identify a suitable class of DB theories having a model completion and whose constraint satisfiability problem is decidable. A first result in this sense is given below. Given the characteristic graph $G(\Sigma)$ of a DB signature Σ, we recall that Σ is said to be *acyclic* if $G(\Sigma)$ is so.

Proposition 4.5. *[17] A DB theory T has decidable constraint satisfiability problem and admits a model completion in case it is axiomatized by finitely many universal one-variable formulae and Σ is acyclic.* ◁

We omit the proof of the above proposition, because the proposition does not play a role in the following: the proof can be easily obtained by well-known facts from the literature and is nevertheless reported in full detail in [17]. We only report here the algorithm for quantifier elimination in T^* suggested by that proof: given a primitive formula $\exists \underline{e} \, \phi(\underline{e}, \underline{y})$, the output $\psi(\underline{y})$ of the algorithm is simply the conjunction of the set of all quantifier-free $\chi(\underline{y})$-formulae such that $\phi(\underline{e}, \underline{y}) \rightarrow \chi(\underline{y})$ is a logical consequences of T (they are finitely many - up to T-equivalence - because Σ is acyclic). We also notice that, since acyclicity of Σ yields local finiteness, we immediately get as a Corollary the decidability of safety problems for transitions systems based on DB schemas satisfying the hypotheses of the above theorem.

5 Covers via Constrained Superposition

Of course, a model completion may not exist at all. Proposition 4.5 shows that it exists in case T is a DB theory axiomatized by universal one-variable formulae

and Σ is acyclic. The second hypothesis is unnecessarily restrictive and the algorithm for quantifier elimination suggested by the proof of Proposition 4.5 is highly impractical: for this reason we are trying a different approach. In this section, we drop the acyclicity hypothesis and examine the case where the theory T is empty and the signature Σ may contain function symbols of any arity. Covers in this context were shown to exist already in [38], using an algorithm that, very roughly speaking, determines all the conditional equations that can be derived concerning the nodes of the congruence closure graph. An algorithm for the generation of interpolants, still relying on congruence closure [41], is sketched in [42].

We follow a different plan and we want to produce covers (and show that they exist) using *saturation-based theorem proving*. The natural idea to proceed in this sense is to take the matrix $\phi(\underline{e}, \underline{y})$ of the primitive formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ we want to compute the cover of: this is a conjunction of literals, so we consider each variable as a free constant, we saturate the corresponding set of ground literals and finally we output the literals involving only the $\underline{y}$. For saturation, one can use any version of the superposition calculus [55]. However, this procedure for our problem is not sufficient. As a trivial counterexample consider the primitive formula $\exists e\, (R(e, y_1) \wedge \neg R(e, y_2))$: the set of literals $\{R(e, y_1), \neg R(e, y_2)\}$ is saturated (recall that we view e, y_1, y_2 as constants), however the formula has a non-trivial cover $y_1 \neq y_2$ which is *not* produced by saturation. If we move to signatures with function symbols, the situation is even worse: the set of literals $\{f(e, y_1) = y_1', f(e, y_2) = y_2'\}$ is saturated but the formula $\exists e\, (f(e, y_1) = y_1' \wedge f(e, y_2) = y_2')$ has the *conditional equality* $y_1 = y_2 \rightarrow y_1' = y_2'$ as cover. *Disjunctions of disequations* might also arise: the cover of $\exists e\, h(e, y_1, y_2) \neq h(e, y_1', y_2')$ (as well as the cover of $\exists e\, f(f(e, y_1), y_2) \neq f(f(e, y_1'), y_2')$, see Example 5.5 below) is $y_1 \neq y_1' \vee y_2 \neq y_2'$. [5]

Notice that our problem is different from the problem of producing ordinary quantifier-free interpolants via saturation based theorem proving [43]: for ordinary Craig interpolants, we have as input *two* quantifier-free formulae $\phi(\underline{e}, \underline{y}), \phi'(\underline{y}, \underline{z})$ such that $\phi(\underline{e}, \underline{y}) \rightarrow \phi'(\underline{y}, \underline{z})$ is valid; here we have a *single* formula $\phi(\underline{e}, \underline{y})$ as input and we are asked to find an interpolant which is good *for all possible* $\phi'(\underline{y}, \underline{z})$ such that $\phi(\underline{e}, \underline{y}) \rightarrow \phi'(\underline{y}, \underline{z})$ is valid. Ordinary interpolants can be extracted from a refutation of $\phi(\underline{e}, \underline{y}) \wedge \neg\phi'(\underline{y}, \underline{z})$, whereas here we are not given any refutation at all (and we are not even supposed to find one).

What we are going to show is that, nevertheless, saturation via superposition can be used to produce covers, if suitably adjusted. In this section we consider signatures with n-ary function symbols (for all $n \geq 1$). For simplicity, we omit n-ary relation symbols (they can be easily handled by rewriting $R(t_1, \ldots, t_n)$ as $R(t_1, \ldots, t_n) = true$, as customary in the paramodulation literature [55]).

We are going to compute the cover of a primitive formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ to be fixed for the remainder of this section. We call variables $\underline{e}$ *existential* and variables $\underline{y}$ *parameters*. By applying abstraction steps, we can assume that ϕ is *primitive flat*, i.e. that it is a conjunction of $\underline{e}$*-flat literals*, defined below. [By an abstraction step we mean replacing $\exists \underline{e}\, \phi$ with $\exists \underline{e}\, \exists e'\, (e' = u \wedge \phi')$, where e' is a fresh variable and ϕ' is obtained from ϕ by replacing some occurrences of a term $u(\underline{e}, \underline{y})$ by e'].

[5] This example points out a problem that needs to be fixed in the algorithm presented in [38]: that algorithm in fact outputs only equalities, conditional equalities and single disequalities, so it cannot correctly handle this example.

A term or a formula are said to be $\underline{e}$-free iff the existential variables do not occur in it. An $\underline{e}$-flat term is an $\underline{e}$-free term $t(\underline{y})$ or a variable from $\underline{e}$ or again it is of the kind $f(u_1, \ldots, u_n)$, where f is a function symbol and $u_1, \ldots, u_n$ are $\underline{e}$-free terms or variables from $\underline{e}$. An $\underline{e}$-flat literal is a literal of the form

$$t = a, \quad a \neq b$$

where t is an $\underline{e}$-flat term and a, b are either $\underline{e}$-free terms or variables from $\underline{e}$.

We assume the reader is familiar with standard conventions used in rewriting and paramodulation literature: in particular $s_{|p}$ denotes the subterm of s in position p and $s[u]_p$ denotes the term obtained from s by replacing $s_{|p}$ with u. We use $\equiv$ to indicate coincidence of syntactic expressions (as strings) to avoid confusion with equality symbol; when we write equalities like $s = t$ below, we may mean both $s = t$ or $t = s$ (an equality is seen as a multiset of two terms). For information on reduction orderings, see for instance [2].

We first replace variables $\underline{e} = e_1, \ldots, e_n$ and $\underline{y} = y_1, \ldots, y_m$ by free constants - we keep the names $e_1, \ldots, e_n, y_1, \ldots, y_m$ for these constants. Let $>$ be a reduction ordering that is total for ground terms such that $\underline{e}$-flat literals $t = a$ are always oriented from left to right in the following two cases: (i) t is not $\underline{e}$-free and a is $\underline{e}$-free; (ii) t is not $\underline{e}$-free, it is not equal to any of the $\underline{e}$ and a is a variable from $\underline{e}$. To obtain such properties, one may for instance choose a suitable Knuth-Bendix ordering taking weights in some transfinite ordinal (see, e.g., [47]).

Given two $\underline{e}$-flat terms t, u, we indicate with $E(t, u)$ the following procedure, which intuitively is a unification algorithm for the terms t and u where the $\underline{e}$ variables are treated as constants; as shown by Lemma 5.1 below, $E(t, u)$ collects 'the equalities that are needed in order to force $t = u$', whenever the $\underline{e}$ are assumed to be free (i.e. not to satisfy any specific equational constraint):

- $E(t, u)$ fails if t is $\underline{e}$-free and u is not $\underline{e}$-free (or vice versa);
- $E(t, u)$ fails if $t \equiv e_i$ and (either $u \equiv f(t_1, \ldots, t_k)$ or $u \equiv e_j$ for $i \neq j$);
- $E(t, u) = \emptyset$ if $t \equiv u$;
- $E(t, u) = \{t = u\}$ if t and u are different but both $\underline{e}$-free;
- $E(t, u)$ fails if none of t, u is $\underline{e}$-free, $t \equiv f(t_1, \ldots, t_k)$ and $u \equiv g(u_1, \ldots, u_l)$ for $f \not\equiv g$;
- $E(t, u) = E(t_1, u_1) \cup \cdots \cup E(t_k, u_k)$ if none of t, u is $\underline{e}$-free, $t \equiv f(t_1, \ldots, t_k)$, $u \equiv f(u_1, \ldots, u_k)$ and none of the $E(t_i, u_i)$ fails.

Notice that, whenever $E(t, u)$ succeeds, the formula $\bigwedge E(t, u) \rightarrow t = u$ is universally valid. The definition of $E(t, u)$ is motivated by the next lemma.

Lemma 5.1. *Let R be a convergent (i.e. terminating and confluent) ground rewriting system, whose rules consist of $\underline{e}$-free terms. Suppose that t and u are $\underline{e}$-flat terms with the same R-normal form. Then $E(t, u)$ does not fail and all pairs from $E(t, u)$ have the same R-normal form as well.* ◁

Proof. This is due to the fact that if t is not $\underline{e}$-free, no R-rewriting is possible at root position because rules from R are $\underline{e}$-free. ⊣

In the following, we handle *constrained* ground flat literals of the form $L \,\|\, C$ where L is a ground flat literal and C is a conjunction of ground equalities among $\underline{e}$-free terms. The logical meaning of $L \,\|\, C$ is the Horn clause $\bigwedge C \rightarrow L$.

In the literature, various calculi with constrained clauses were considered, starting, e.g., from the non-ground constrained versions of the Superposition Calculus of [4, 54]. The calculus we propose here is inspired by such versions and it has close similarities with a subcase of hierarchic superposition calculus [5], or rather to its

"weak abstraction" variant from [6] (we thank an anonymous referee of our CADE 2019 submission for pointing out this connection).

The rules of our *Constrained Superposition Calculus* follow; each rule applies provided the E subprocedure called by it does not fail. The symbol $\bot$ indicates the empty clause. Further explanations and restrictions to the calculus are given in the Remarks below.

Superposition Right (Constrained)	$\dfrac{l = r \parallel C \qquad s = t \parallel D}{s[r]_p = t \parallel C \cup D \cup E(s_{\mid p}, l)}$	if $l > r$ and $s > t$
Superposition Left (Constrained)	$\dfrac{l = r \parallel C \qquad s \neq t \parallel D}{s[r]_p \neq t \parallel C \cup D \cup E(s_{\mid p}, l)}$	if $l > r$ and $s > t$
Reflection (Constrained)	$\dfrac{t \neq u \parallel C}{\bot \parallel C \cup E(t, u)}$	
Demodulation (Constrained)	$\dfrac{L \parallel C, \qquad l = r \parallel D}{L[r]_p \parallel C}$	if $l > r$, $L_{\mid p} \equiv l$ and $C \supseteq D$

Remark 5.1. The first three rules are inference rules: they are non-deterministically selected for application, until no rule applies anymore. The selection strategy for the rule to be applied is not relevant for the correctness and completeness of the algorithm (some variant of a 'given clause algorithm' can be applied). An inference rule *is not applied in case one premise is $\underline{e}$-free* (we have no reason to apply inferences to $\underline{e}$-free premises, since we are not looking for a refutation). ◁

Remark 5.2. The Demodulation rule is a simplification rule: its application not only adds the conclusion to the current set of constrained literals, but it also removes the first premise. It is easy to see (e.g., representing literals as multisets of terms and extending the total reduction ordering to multisets), that one cannot have an infinite sequence of consecutive applications of Demodulation rules. ◁

Remark 5.3. The calculus takes $\{L \| \emptyset \mid L \text{ is a flat literal from the matrix of } \phi\}$ as the initial set of constrained literals. It terminates when a *saturated* set of constrained literals is reached. We say that S is saturated iff every constrained literal that can be produced by an inference rule, after being exhaustively simplified via Demodulation, is already in S (there are more sophisticated notions of 'saturation up to redundancy' in the literature, but we do not need them). When it reaches a saturated set S, the algorithm outputs the conjunction of the clauses $\bigwedge C \to L$, varying $L \parallel C$ among the $\underline{e}$-free constrained literals from S. ◁

We need some rule application policy to ensure termination: without any such policy, a set like

$$\{e = y \parallel \emptyset, f(e) = e \parallel \emptyset\} \tag{4}$$

may produce by Right Superposition the infinitely many literals (all oriented from right to left) $f(y) = e \parallel \emptyset$, $f(f(y)) = e \parallel \emptyset$, $f(f(f(y))) = e \parallel \emptyset$, etc. The next remark explains the policy we follow.

Remark 5.4. [Policy Remark] We apply Demodulation *only in case the second premise is of the kind $e_j = t(\underline{y}) \parallel D$, where t is $\underline{e}$-free.* Demodulation rule is applied with *higher priority* with respect to the inference rules.[6] Inside all possible applications of Demodulation rule, we give priority to the applications where *both*

[6] Thus we cannot apply Superposition to $\{e = y \parallel \emptyset, f(e) = e \parallel \emptyset\}$ until Demodulation is exhaustively applied (the latter causes the deletion of $f(e) = e \parallel \emptyset$ and its replacement with $f(y) = y \parallel \emptyset$, thus blocking the above generation of infinitely many clauses).

premises have the form $e_j = t(\underline{y}) \,\|D$ (for the same e_j but with possibly different D's - the D from the second premise being included in the D of the first). In case we have two constrained literals of the kind $e_j = t_1(\underline{y}) \,\|D$, $e_j = t_2(\underline{y}) \,\|D$ inside our current set of constrained literals (notice that the e_j's and the D's here are the same), among the two possible applications of the Demodulation rule, we apply the rule that keeps the smallest t_i. Notice that in this way two different constrained literals cannot simplify each other. ◁

We say that a constrained literal $L \,\|C$ belonging to a set of constrained literals S is *simplifiable in* S iff it is possible to apply (according to the above policy) a Demodulation rule removing it. A first effect of our policy is:

Lemma 5.2. *If a constrained literal* $L \,\|\, C$ *is simplifiable in* S*, then after applying to* S *any sequence of rules, it remains simplifiable until it gets removed. After being removed, if it is regenerated, it is still simplifiable and so it is eventually removed again.* ◁

Proof. Suppose that $L \,\|\, C$ can be simplified by $e = t \,\|\, D$ and suppose that a rule is applied to the current set of constrained literals. Since there are simplifiable constrained literals, that rule cannot be an inference rule by the priority stated in Remark 5.4. For simplification rules, keep in mind again Remark 5.4. If $L \,\|\, C$ is simplified, it is removed; if none of $L \,\|\, C$ and $e = t \,\|\, D$ get simplified, the situation does not change; if $e = t \,\|\, D$ gets simplified, this can be done by some $e = t' \|\, D'$, but then $L \,\|\, C$ is still simplifiable - although in a different way - using $e = t' \|\, D'$ (we have that D' is included in D, which is in turn included in C). Similar observations apply if $L \,\|\, C$ is removed and re-generated. ⊣

Due to Lemma 5.2, if we show that a derivation (i.e., a sequence of applications of rules) can produce terms only from a finite set, it is clear that when no new constrained literal is produced, saturation is reached. First notice that:

Lemma 5.3. *Every constrained literal* $L \,\|C$ *produced during the run of the algorithm is* $\underline{e}$*-flat.* ◁

Proof. The constrained literals from initialization are $\underline{e}$-flat. The Demodulation rule, applied according to Remark 5.4, produces an $\underline{e}$-flat literal out of an $\underline{e}$-flat literal. The same happens for the Superposition rules: in fact, since both the terms s and l from these rules are $\underline{e}$-flat, a Superposition may take place at root position or may rewrite some $l \equiv e_j$ with $r \equiv e_i$ or with $r \equiv t(\underline{y})$.[7] ⊣

There are in principle infinitely many $\underline{e}$-flat terms that can be generated out of the $\underline{e}$-flat terms occurring in ϕ (see the above counterexample (4)). We show however that only finitely many $\underline{e}$-flat terms can in fact occur during saturation and that one can determine in advance the finite set they are taken from.

To formalize this idea, let us introduce a hierarchy of $\underline{e}$-flat terms (this hierarchy concerns terms, not clauses or constraints - although it will be used to delimit the kind of clauses or constraints that might occur in a saturation process). Let D_0 be the $\underline{e}$-flat terms occurring in ϕ and let D_{k+1} be the set of $\underline{e}$-flat terms obtained

[7] Notice that Superposition Left is considerably restricted in our calculus: recall in fact that $\underline{e}$-flat *negative* literals must be of the kind $s \neq t$ where s, t are either variables from $\underline{e}$ or $\underline{e}$-free terms. Since rules do not apply to $\underline{e}$-free literals, the only possibility is that the term s from the literal $s \neq t$ of the right premise of Superposition Left is a variable from $\underline{e}$ and that the term l from the left premise concides with it. Thus Superposition Left looks like a Demodulation, however it is *not* a Demodulation because the constraint of its left premise may not be included into the constraint of its right premise. It would be harmless to allow a more liberal version of Superposition Left, but we do not need it.

by simultaneous rewriting of an $\underline{e}$-flat term from $\bigcup_{i \leq k} D_i$ via rewriting rules of the kind $e_j \to t_j(\underline{y})$ where the t_j are $\underline{e}$-free terms from $\bigcup_{i \leq k} D_i$. The *degree* of an $\underline{e}$-flat term is the minimum k such that it belongs to set D_k (it is necessary to take the minimum because the same term can be obtained at different stages and via different rewritings).

Lemma 5.4. *Let the $\underline{e}$-flat term t' be obtained by a rewriting $e_j \to u(\underline{y})$ from the $\underline{e}$-flat term t; then, if t has degree $k > 1$ and u has degree at most $k-1$, we have that t' has degree at most k.* ◁

Proof. This is clear, because at the k-stage one can directly produce t' instead of just t: in fact, all rewriting producing directly t' replace an occurrence of some e_i by an $\underline{e}$-free term, so they are all done in parallel positions. [We illustrate the phenomenon via an example: suppose that t is $f(e_1, g(g(c)))$ and that t' is obtained from t by rewriting e_1 to $g(c)$. Now it might well be that t has degree 2, being obtained from $f(e_1, e_2)$ via $e_2 \mapsto g(g(c)))$ (the latter having been previously obtained from $g(e_3)$ via $e_3 \mapsto g(c)$). Now t' still has degree 2 because it can be directly obtained from $f(e_1, e_2)$ via the parallel rewritings $e_1 \mapsto g(c)$, $e_2 \mapsto g(g(c)))$.] ⊣

Proposition 5.5. *The saturation of the initial set of $\underline{e}$-flat constrained literals always terminates after finitely many steps.* ◁

Proof. We show that all $\underline{e}$-flat terms that may occur during saturation have at most degree n (where n is the cardinality of $\underline{e}$). This shows that the saturation must terminate, because only finitely many terms may occur in a derivation (see the above observations). Let the algorithm during saturation reach the state S; we say that *a constraint C allows the explicit definition of e_j in S* iff S contains a constrained literal of the kind $e_j = t(\underline{y}) \,\|D$ with $D \subseteq C$. Now we show by mutual induction two facts concerning a constrained literal $L \,\|\, C \in S$:

(1) if an $\underline{e}$-flat term u of degree k occurs in L, then C allows the explicit definition of k different e_j in S;
(2) if L is of the kind $e_i = t(\underline{y})$, for an $\underline{e}$-free term t of degree k, then either $e_i = t \,\|\, C$ can be simplified in S or C allows the explicit definition of $k+1$ different e_j in S (e_i itself is of course included among these e_j).

Notice that (1) is sufficient to exclude that any $\underline{e}$-flat term of degree bigger than n can occur in a constrained literal arising during the saturation process.

We prove (1) and (2) by induction on the length of the derivation leading to $L \,\|\, C \in S$. Notice that it is sufficient to check that (1) and (2) hold for the first time where $L \,\|\, C \in S$ because if C allows the explicit definition of a certain variable in S, it will continue to do so in any S' obtained from S by continuing the derivation (the definition may be changed by the Demodulation rule, but the fact that e_i is explicitly defined is forever). Also, by Lemma 5.2, a literal cannot become non simplifiable if it is simplifiable.

(1) and (2) are evident if S is the initial status. To show (1), suppose that u occurs for the first time in $L \,\|\, C$ as the effect of the application of a certain rule: we can freely assume that u does not occur in the literals from the premisses of the rule (otherwise induction trivially applies) and that u of degree k is obtained by rewriting *in a non-root position* some u' occurring in a constrained literal $L' \,\|\, D'$ via some $e_j \to t \,\|\, D$. This might be the effect of a Demodulation or Superposition in a non-root position (Superpositions in root position do not produce new terms). If u' has degree k, then by induction D' contains the required k explicit definitions,

and we are done because D' is included in C. If u' has lower degree, then t must have degree at least $k-1$ (otherwise u does not reach degree k by Lemma 5.4). Then by induction on (2), the constraint D (also included in C) has $(k-1)+1=k$ explicit definitions (when a constraint $e_j \to t \,\|D$ is selected for Superposition or for making Demodulations in a non-root position, it is itself not simplifiable according to the procedure explained in Remark 5.4).

To show (2), we analyze the reasons why the non simplifiable constrained literal $e_i = t(\underline{y}) \,\|\, C$ is produced (let k be the degree of t). Suppose it is produced from $e_i = u'(\underline{y}) \,\|\, C$ via Demodulation with $e_j = u(\underline{y}) \,\|\, D$ (with $D \subseteq C$) in a non-root position; if u' has degree at least k, we apply induction for (1) to $e_i = u' \,\|\, C$: by such induction hypotheses, we get k explicit definitions in C and we can add to them the further explicit definition $e_i = t(\underline{y})$ (the explicit definitions from C cannot concern e_i because $e_i = t(\underline{y}) \,\|\, C$ is not simplifiable). Otherwise, u' has degree less than k and u has degree at least $k-1$ by Lemma 5.4 (recall that t has degree k): by induction, $e_j = u \,\|\, D$ is not simplifiable (it is used as the active part of a Demodulation in a non-root position, see Remark 5.4) and supplies k explicit definitions, inherited by $C \supseteq D$. Note that e_i cannot have a definition in D, otherwise $e_i = t(\underline{y}) \,\|\, C$ would be simplifiable, so with $e_i = t(\underline{y}) \,\|\, C$ we get the required $k+1$ definitions.

The remaining case is when $e_i = t(\underline{y}) \,\|\, C$ is produced via Superposition Right. Such a Superposition might be at root or at a non-root position. We first analyse the case of a root position. This might be via $e_j = e_i \,\|\, C_1$ and $e_j = t(\underline{y}) \,\|\, C_2$ (with $e_j > e_i$ and $C = C_1 \cup C_2$ because $E(e_j, e_j) = \emptyset$), but in such a case one can easily apply induction. Otherwise, we have a different kind of Superposition at root position: $e_i = t(\underline{y}) \,\|\, C$ is obtained from $s = e_i \,\|\, C_1$ and $s' = t(\underline{y}) \,\|\, C_2$, with $C = C_1 \cup C_2 \cup E(s, s')$. In this case, by induction for (1), C_2 supplies k explicit definitions, to be inherited by C. Among such definitions, there cannot be an explicit definition of e_i otherwise $e_i = t(\underline{y}) \,\|\, C$ would be simplifiable, so again we get the required $k+1$ definitions.

In case of a Superposition at a non root-position, we have that $e_i = t(\underline{y}) \,\|\, C$ is obtained from $u' = e_i \,\|\, C_1$ and $e_j = u(\underline{y}) \,\|\, C_2$, with $C = C_1 \cup C_2$; here t is obtained from u' by rewriting e_j to u. This case is handled similarly to the case where $e_i = t(\underline{y}) \,\|\, C$ is obtained via Demodulation rule. ⊣

Having established termination, we now prove that our calculus computes covers. To this aim, we rely on refutational completeness of unconstrained Superposition Calculus: thus, our technique resembles the technique used [5,6] in order to prove refutational completeness of hierarchic superposition, although it is not clear whether Theorem 5.6 below can be derived from the results concerning hierarchic superposition[8]. We state the following theorem:

Theorem 5.6. *Let T be the theory $\mathcal{EUF}$. Suppose that the above algorithm, taking as input the primitive $\underline{e}$-flat formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$, gives as output the quantifier-free formula $\psi(\underline{y})$. Then the latter is a T-cover of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$.* ◁

Proof. Let S be the saturated set of constrained literals produced upon termination of the algorithm; let $S = S_1 \cup S_2$, where S_1 contains the constrained literals

[8] An important difference between our proof and the proof of completeness for hierarchic superposition is that we must build an expansion *of a superstructure* of the model $\mathcal{M}$ below (expanding $\mathcal{M}$ to a larger signature without enlarging its domain might not be possible in principle).

in which the $\underline{e}$ do not occur and S_2 is its complement. Clearly $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ turns out to be logically equivalent to

$$\bigwedge_{L \,\|\, C \in S_1} (\bigwedge C \to L) \wedge \exists \underline{e} \bigwedge_{L \,\|\, C \in S_2} (\bigwedge C \to L)$$

so, as a consequence, in view of Lemma 3.1 it is sufficient to show that every model $\mathcal{M}$ satisfying $\bigwedge_{L \,\|\, C \in S_1} (\bigwedge C \to L)$ via an assignment $\mathcal{I}$ to the variables $\underline{y}$ can be embedded into a model $\mathcal{M}'$ such that for a suitable extension $\mathcal{I}'$ of $\mathcal{I}$ to the variables $\underline{e}$ we have that $(\mathcal{M}', \mathcal{I}')$ satisfies also $\bigwedge_{L \,\|\, C \in S_2} (\bigwedge C \to L)$.

Fix $\mathcal{M}, \mathcal{I}$ as above. The diagram $\Delta(\mathcal{M})$ of $\mathcal{M}$ is obtained as follows. We take one free constant for each element of the support of $\mathcal{M}$ (by Löwenheim-Skolem theorem one can keep $\mathcal{M}$ at most countable, if you like) and we put in $\Delta(\mathcal{M})$ all the literals of the kind $f(c_1, \ldots, c_k) = c_{k+1}$ and $c_1 \neq c_2$ which are true in $\mathcal{M}$ (here the c_i are names for the elements of the support of $\mathcal{M}$). Let R be the set of ground equalities of the form $y_i = c_i$, where c_i is the name of $\mathcal{I}(y_i)$. Extend our reduction ordering in the natural way (so that $y_i = c_i$ and $f(c_1, \ldots, c_k) = c_{k+1}$ are oriented from left to right). Consider now the set of clauses

$$\Delta(\mathcal{M}) \;\cup\; R \;\cup\; \{\bigwedge C \to L \mid (L \,\|\, C) \in S\} \tag{5}$$

(below, we distinguish the positive and the negative literals of $\Delta(\mathcal{M})$ so that $\Delta(\mathcal{M}) = \Delta^+(\mathcal{M}) \cup \Delta^-(\mathcal{M})$). We want to saturate the above set in the standard Superposition Calculus. Clearly the rewriting rules in R, used as reduction rules, replace everywhere y_i by c_i inside the clauses of the kind $\bigwedge C \to L$. At this point, the negative literals from the equality constraints all disappear: if they are true in $\mathcal{M}$, they $\Delta^+(\mathcal{M})$-normalize to trivial equalities $c_i = c_i$ (to be eliminated by standard reduction rules) and if they are false in $\mathcal{M}$ they become part of clauses subsumed by true inequalities from $\Delta^-(\mathcal{M})$. Similarly all the $\underline{e}$-free literals not coming from $\Delta(\mathcal{M}) \cup R$ get removed. Let $\tilde{S}$ be the set of survived literals involving the $\underline{e}$ (they are not constrained anymore and they are $\Delta^+(\mathcal{M}) \cup R$-normalized): we show that they cannot produce new clauses. Let in fact (π) be an inference from the Superposition Calculus [55] applying to them. Since no superposition with $\Delta(\mathcal{M}) \cup R$ is possible, this inference must involve only literals from $\tilde{S}$; suppose it produces a literal $\tilde{L}$ from the literals $\tilde{L}_1, \tilde{L}_2$ (coming via $\Delta^+(\mathcal{M}) \cup R$-normalization from $L_1 \,\|\, C_1 \in S$ and $L_2 \,\|\, C_2 \in S$) as parent clauses. Then, by Lemma 5.1, our constrained inferences produce a constrained literal $L \,\|\, C$ such that the clause $\bigwedge C \to L$ normalizes to $\tilde{L}$ via $\Delta^+(\mathcal{M}) \cup R$. Since S is saturated, the constrained literal $L \,\|\, C$, after simplification, belongs to S. Now simplifications via our Constrained Demodulation and $\Delta(\mathcal{M})^+ \cup R$-normalization commute (they work at parallel positions, see Remark 5.4), so the inference (π) is redundant because $\tilde{L}$ simplifies to a literal already in $\tilde{S} \cup \Delta(\mathcal{M})$.

Thus the set of clauses (5) saturates without producing the empty clause. By the completeness theorem of the Superposition Calculus [40,3,55] it has a model $\mathcal{M}'$. This $\mathcal{M}'$ by construction fits our requests by Robinson Diagram Lemma. $\dashv$

Theorem 5.6, thanks to the relationship between model completions and covers stated in Theorem 3.2, proves also the existence of the model completion of $\mathcal{EUF}$.

Example 5.5. We compute the cover of the primitive formula $\exists e\, f(f(e, y_1), y_2) \neq f(f(e, y_1'), y_2')$. Flattening gives the set of literals

$$\{\; f(e, y_1) = e_1,\; f(e_1, y_2) = e_1',\; f(e, y_1') = e_2,\; f(e_2, y_2') = e_2',\; e_1' \neq e_2' \;\} \quad .$$

Superposition Right produces the constrained literal $e_1 = e_2 \parallel \{y_1 = y_1'\}$; supposing that we have $e_1 > e_2$, Superposition Right gives first $f(e_2, y_2) = e_1' \parallel \{y_1 = y_1'\}$ and then also $e_1' = e_2' \parallel \{y_1 = y_1', y_2 = y_2'\}$. Superposition Left and Reflection now produce $\bot \parallel \{y_1 = y_1', y_2 = y_2'\}$. Thus the clause $y_1 = y_1' \wedge y_2 = y_2' \rightarrow \bot$ will be part of the output (actually, this will be the only clause in the output). $\triangleleft$

We apply our algorithm to an additional example, taken from [38].

Example 5.6. We compute the cover of the primitive formula $\exists e\,(s_1 = f(y_3, e) \wedge s_2 = f(y_4, e) \wedge t = f(f(y_1, e), f(y_2, e)))$, where s_1, s_2, t are terms in $\underline{y}$. Flattening gives the set of literals

$$\{\ f(y_3, e) = s_1,\ f(y_4, e) = s_2,\ f(y_1, e) = e_1,\ f(y_2, e) = e_2,\ f(e_1, e_2) = t\ \}\ .$$

Suppose that we have $e > e_1 > e_2 > t > s_1 > s_2 > y_1 > y_2 > y_3 > y_4$. Superposition Right between the 3rd and the 4th clauses produces the constrained 6th clause $e_1 = e_2 \parallel \{y_1 = y_2\}$. From now on, we denote the application of a Superposition Right to the ith and jth clauses with $R(i, j)$. We list a derivation performed by our calculus:

$$\begin{aligned}
R(3,4) &\implies e_1 = e_2 \parallel \{y_1 = y_2\} && \text{(6th clause)}\\
R(1,2) &\implies s_1 = s_2 \parallel \{y_3 = y_4\} && \text{(7th clause)}\\
R(5,6) &\implies f(e_2, e_2) = t \parallel \{y_1 = y_2\} && \text{(8th clause)}\\
R(1,3) &\implies e_1 = s_1 \parallel \{y_1 = y_3\} && \text{(9th clause)}\\
R(1,4) &\implies e_2 = s_1 \parallel \{y_2 = y_3\} && \text{(10th clause)}\\
R(2,3) &\implies e_1 = s_2 \parallel \{y_1 = y_4\} && \text{(11th clause)}\\
R(2,4) &\implies e_2 = s_2 \parallel \{y_2 = y_4\} && \text{(12th clause)}\\
R(5,9) &\implies f(s_1, e_2) = t \parallel \{y_1 = y_3\} && \text{(13th clause)}\\
R(5,11) &\implies f(s_2, e_2) = t \parallel \{y_1 = y_4\} && \text{(14th clause)}\\
R(6,9) &\implies e_2 = s_1 \parallel \{y_1 = y_3, y_1 = y_2\} && \text{(15th clause)}\\
R(6,11) &\implies e_2 = s_2 \parallel \{y_1 = y_2, y_1 = y_4\} && \text{(16th clause)}\\
R(8,10) &\implies f(s_1, s_1) = t \parallel \{y_1 = y_3, y_2 = y_3\} && \text{(17th clause)}\\
R(8,12) &\implies f(s_2, s_2) = t \parallel \{y_1 = y_4, y_2 = y_4\} && \text{(18th clause)}\\
R(13,12) &\implies f(s_1, s_2) = t \parallel \{y_1 = y_3, y_2 = y_4\} && \text{(19th clause)}\\
R(14,10) &\implies f(s_2, s_1) = t \parallel \{y_1 = y_4, y_2 = y_3\} && \text{(20th clause)}\\
R(9,11) &\implies s_1 = s_2 \parallel \{y_1 = y_3, y_1 = y_4\} && \text{(21th clause)}
\end{aligned}$$

The set of clauses above is saturated. The 7th, 17th, 18th, 19th and 20th clauses are exactly the output clauses of [38]. The non-simplified clauses that do not appear as output in [38] are redundant and they could be simplified by introducing a Subsumption rule as an additional simplification rule of our calculus.

6 Complexity analysis of the fragment for database driven applications

The saturation procedure of Theorem 5.6 can in principle produce double exponentially many clauses, because there are exponentially many terms of degree n (if n is the cardinality of the variables to be eliminated); it is not clear whether we can improve this bound to a simple exponential one, by limiting the kind of terms that can be produced. An estimation of the complexity costs of computing uniform interpolants in $\mathcal{EUF}$ is better performed within approaches making use of compressed DAG-representations of terms [26]. In this paper, however, we are especially interested (for our applications to verification of data-aware processes) to the special case where the signature Σ contains only unary function symbols and relations of arbitrary arity (cf. Subsection 4.1). In this special case, important remarks apply. In fact, we shall see below that if the signature Σ contains only unary function symbols, only empty constraints can be generated; in case Σ contains also relation symbols of arity $n > 1$, the only constrained clauses that can be generated have the form $\bot \| \{t_1 = t'_1, \ldots, t_{n-1} = t'_{n-1}\}$. Also, it is not difficult to see that in a derivation at most one explicit definition $e_i = t(\underline{y}) || \emptyset$ can occur for every e_i: as soon as this definition is produced, all occurrences of e_i are rewritten to t. This implies that Constrained Superposition computes covers in polynomial time for the empty theory, whenever the signature Σ matches the restrictions of Definition 4.3 for DB schemas. We give here a finer complexity analysis, in order to obtain a quadratic bound.

In this section, *we assume that our signature Σ contains only unary function and m-ary relation symbols.* In order to attain the optimized quadratic complexity bound, we need to follow a *different strategy in applying the rules of our constrained superposition calculus* (this different strategy would not be correct for the general case). Thanks to this different strategy, we can make our procedure close to the algorithm of [38]: in fact, such algorithm is correct for the case of unary functions and requires only a minor adjustment for the case of unary functions and m-ary relations. Since relations play a special role in the present restricted context, we prefer to treat them as such, i.e. not to rewrite $R(t_1, \ldots, t_n)$ as $R(t_1, \ldots, t_n) = \mathit{true}$; the consequence is that we need an additional Constrained Resolution Rule[9]. We preliminarily notice that when function symbols are all unary, the constraints remain all *empty* during the run of the saturation procedure, except for the case of the newly introduced Resolution Rule below. This fact follows from the observation that given two terms u_1 and u_2, procedure $E(u_1, u_2)$ does not fail iff:
(1) either u_1 and u_2 are both terms containing only variables from $\underline{y}$, or
(2) u_1 and u_2 are terms that syntactically coincide.
In case (1), $E(u_1, u_2)$ is $\{u_1 = u_2\}$ and in case (2), $E(u_1, u_2)$ is $\emptyset$. In case (1), Superposition Rules are not applicable. To show this, suppose that $u_1 \equiv s_{|p}$ and $u_2 \equiv l$; then, terms l and r use only variables from $\underline{y}$, and consequently cannot be fed into Superposition Rules, since Superposition Rules are only applied when variables from $\underline{e}$ occur in both premises. Reflection Rule does not apply too in case (1), because this rule (like any other rule) cannot be applied to an $\underline{e}$-free literal.

Thus, in the particular case of m-ary relations and unary functions, the rules of the calculus are the following:

[9] We extend the definition of an $\underline{e}$-flat literal so as to include also the literals of the kind $R(t_1, .., t_n)$ and $\neg R(t_1, .., t_n)$ where the terms t_i are either $\underline{e}$-free terms or variables from $\underline{e}$.

Superposition	$\dfrac{l = r \qquad L}{L[r]_p}$	if	(i) $l > r$; (ii) if $L \equiv s = t$ or $L \equiv s \neq t$, then $s > t$ and $p \in Pos(s)$; (iii) $E(s_{\|p}, l)$ does not fail.
Resolution	$\dfrac{R(t_1, \ldots, t_n)\ \neg R(s_1, \ldots, s_n)}{\bot \parallel \bigcup_i E(s_i, t_i)}$	if	$E(s_i, t_i)$ does not fail for all $i = 1, \ldots, n$
Reflection	$\dfrac{t \neq u}{\bot}$	if	$E(t, u)$ does not fail
Demodulation	$\dfrac{L \qquad l = r}{L[r]_p}$	if	$l > r$ and $L_{\|p} \equiv l$

We still restrict the use of our rules to the case where all premises are not $\underline{e}$-free literals; again Demodulation is applied only in the case where $l = r$ is of the kind $e_i = t(\underline{y})$. For the order of applications of the Rules, Lemma 6.1 below show that we can apply (restricted) Superpositions, Demodulations, Reflections and Resolutions in this order and then stop.

An important preliminary observation to obtain such a result is that *we do not need to apply Superposition Rules whose left premise $l = r$ is of the kind $e_i = t(\underline{y})$*: this is because constraints are always empty (unless the constrained clause is the empty clause), so that a Superposition Rule with the left premise $e_i = t(\underline{y})$ can be replaced by a Demodulation Rule. [10] If the left premise of Superposition is not of the kind $e_i = t(\underline{y})$, then since our literals are $\underline{e}$-flat, it can be either of the kind $e_i = e_j$ (with $e_i > e_j$) or of the kind $f(e_i) = t$. In the latter case t is either $e_k \in \underline{e}$ or it is an $\underline{e}$-free term; for Superposition Left (i.e. for Superposition applied to a negative literal), the left premise can only be $e_i = e_j$, because our literals are $\underline{e}$-flat and so negative literals L cannot have a position p such that $L_{|p} \equiv f(e_i)$.

Let S be a set of $\underline{e}$-flat literals with empty constraints; we say that S is *RS-closed* iff it is closed under *Restricted Superposition Rules*, i.e under Superposition Rules whose left premise is not of the kind $e_i = t(\underline{y})$. In equivalent terms, as a consequence of the above discussion, S is RS-closed iff it satisfies the following two conditions:

- if $\{f(e_i) = t, f(e_i) = v\} \subseteq S$, then $t = v \in S$;
- if $\{e_i = e_j, L\} \subseteq S$ and $e_i > e_j$ and $L_{|p} \equiv e_i$, then $L[e_j]_p \in S$.

Since Restricted Superpositions do not introduce essentially new terms (newly introduced terms are just rewritings of variables with variables), it is clear that we can make a finite set S of $\underline{e}$-free literals RS-closed in finitely many steps. This can be naively done in time quadratic in the size of the formula. As an alternative, we can apply a *congruence closure algorithm to S* and produce a set of $\underline{e}$-free constraints S' which is RS-closed and logically equivalent to S: the latter can be done in $O(n \cdot log(n))$-time, as it is well-known from the literature [49, 53, 41].

Lemma 6.1. *Let S be a RS-closed set of empty-constrained $\underline{e}$-flat literals. Then, to saturate S it is sufficient to first exhaustively apply the Demodulation Rule, and then Reflection and Resolution Rules.* ◁

[10] This is not true in the general case where constraints are not empty, because the Demodulation Rule does not merge incomparable constraints.

Proof. Let $\tilde{S}$ be the set obtained from S after having exhaustively applied Demodulation. Notice that the final effect of the reiterated application of Demodulation can be synthetically described by saying that literals in S are rewritten by using some explicit definitions

$$e_{i_1} = t_1(\underline{y}), \ldots, e_{i_k} = t_k(\underline{y}) \quad . \tag{6}$$

These definitions are either in S, or are generated through the Demodulations themselves (we can freely assume that Demodulations are done in appropriate order: first all occurrences of e_{i_1} are rewritten to t_1, then all occurrences of e_{i_2} are rewritten to t_2, etc.).[11]

Suppose now that a pair $L, l = r \in \tilde{S}$ can generate a new literal $L[r]_p$ by Superposition. We know from above that we can limit ourselves to Restricted Superposition, so l is either of the form e_j or of the form $f(e_j)$, where moreover e_j is not among the set $\{e_{i_1}, \ldots, e_{i_k}\}$ from (6). The literals L and $l = r \in \tilde{S}$ happen to have been obtained from literals L' and $l = r'$ belonging to S by applying the rewriting rules (6) (notice that l cannot have been rewritten). Since such rewritings must have occurred in positions parallel to p and since S was closed under Restricted Superposition, we must have that S contained the literal $L'[r']_p$ that rewrites to $L[r]_p$ by the rewriting rules (6). This shows that $L[r]_p$ is already in $\tilde{S}$ (thus, in particular, Demodulation does not destroy RS-closedness) and proves the lemma, because Reflection and Resolution can only produce the empty clause and no rule applies to the empty clause. ⊣

Thus the strategy of applying (in this order)

Restricted Superposition+Demodulation+Reflection+Resolution

always saturates.

To produce an output in optimized format, it is convenient to get it in a DAG-like form. This can be simulated via explicit acyclic definitions as follows. When we write $Def(\underline{e}, \underline{y})$ (where $\underline{e}, \underline{y}$ are tuples of distinct variables), we mean any flat formula of the kind (let $\underline{e} := e_1 \ldots, e_n$) $\bigwedge_{i=1}^{n} e_i = t_i$, where in the term t_i only the variables $e_1, \ldots, e_{i-1}, \underline{y}$ can occur. We shall supply the output in the form

$$\exists \underline{e}' \, (Def(\underline{e}', \underline{y}) \wedge \psi(\underline{e}', \underline{y})) \tag{7}$$

where the $\underline{e}'$ is a subset of the $\underline{e}$ and ψ is quantifier-free. The *DAG-format* (7) is not quantifier-free but can be converted to a quantifier-free formula by unravelling the acyclic definitions of the $\underline{e}'$.

Thus our procedure for computing a cover in DAG-format of a primitive formula $\exists \underline{e} \, \phi(\underline{e}, \underline{y})$ (in case the function symbols of the signature Σ are all unary) runs by performing the following steps, one after the other. Let OUT be a quantifier-free formula (initially OUT is $\top$).

(1) We preprocess ϕ in order to produce a RS-closed set S of empty-constrained $\underline{e}$-flat literals.
(2) We *mark* the variables $\underline{e}$ in the following way (initially, all variables are unmarked): we scan S and, as soon as we find an equality of the kind $e_i = t$ where all variables from $\underline{e}$ occurring in t are marked, we mark e_i. This loop is repeated until no more variable gets marked.
(3) If Reflection is applicable, we output $\bot$ and exit.

[11] In addition, if we happen to have, say, two different explicit definitions of e_{i_1} as $e_{i_1} = t_1, e_{i_1} = t'_1$, we decide to use just one of them (and always the same one, until the other one is eventually removed by Demodulation).

(4) We conjoin OUT with all literals where, besides the $\underline{y}$, only marked variables occur.
(5) For every literal $R(t_1, \ldots, e, \ldots, t_m)$ that contains at least an unmarked e, we scan S until a literal of the type $\neg R(t_1, \ldots, e, \ldots, t_m)$ is found: then, we try to apply Resolution and if we succeed getting $\bot \parallel \{u_1 = u'_1, \ldots, u_m = u'_m\}$, we conjoin $\bigvee_j u_j \neq u'_j$ to OUT.
(6) We prefix to OUT a string of existential quantifiers binding all marked variables and output the result.

One remark is in order: when running the subprocedures $E(s_i, t_i)$ required by the Resolution Rule in (5) above, *all marked variables must be considered as part of the* $\underline{y}$ (thus, e.g. $R(e,t), \neg R(e,v)$ produces $\bot \parallel \{t = u\}$ if both t and u contain, besides the $\underline{y}$, only marked variables).

Proposition 6.2. *Let T be the theory EUF in a signature with unary functions and m-ary relation symbols. Consider a primitive formula $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$; then, the above algorithm returns a T-cover of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$ in DAG-format in time $O(n^2)$, where n is the size of $\exists \underline{e}\, \phi(\underline{e}, \underline{y})$.* ◁

Proof. The preprocessing step (1) requires an abstraction phase for producing $\underline{e}$-flat literals and a second phase in order to get a RS-closed set: the first phase requires linear time, whereas the second one requires $O(n \cdot log(n))$ time (via congruence closure). All the remaining steps require linear time, except steps (2) and (5) that requires quadratic time. This is the dominating cost, thus the entire procedure requires $O(n^2)$ time. ⊣

Although we do not deeply investigate the problem here, we conjecture that it might be possible to further lower down the above complexity to $O(n \cdot log(n))$.

7 An extension of the Constrained Superposition Calculus

We consider an extension of our Constrained Superposition Calculus which is useful for our applications to verification of data-aware processes. Let us assume that we have a theory whose axioms are (3), namely, for every function symbol f:

$$\forall x\ (x = \texttt{undef} \leftrightarrow f(x) = \texttt{undef}) \quad .$$

One direction of the above equivalence is equivalent to the ground literal $f(\texttt{undef}) = \texttt{undef}$ and as such it does not interfere with the completion process (we just add it to our constraints from the very beginning).

To handle the other direction, we need to modify our Calculus. First, we add to the Constrained Superposition Calculus of Section 5 the following extra Rule

Inference Rule $Ext(\texttt{undef})$ **(Constrained)**

$$\frac{f(e_j) = u(\underline{y}) \parallel D}{e_j = \texttt{undef} \parallel D \cup \{u(\underline{y}) = \texttt{undef}\}}$$

The Rule is sound because $u(\underline{y}) = \texttt{undef} \wedge f(e_j) = u(\underline{y}) \rightarrow e_j = \texttt{undef}$ follows from the axioms (3). For cover computation with our new axioms, we need a restricted version of Paramodulation Rule:

Paramodulation (Constrained)

$$\frac{e_j = r \parallel C \qquad L \parallel D}{L[r]_p \parallel C \cup D} \quad (\text{if } e_j > r \ \&\ L_{|p} \equiv e_j)$$

Notice that we can have $e_j > r$ only in case r is either some existential variable e_i or it is an $\underline{e}$-free term $u(\underline{y})$. Paramodulation Rule (if it is not a Superposition) can only apply to a right member of an equality and such a right member must be e_j itself (because our literals are flat). Thus the rule cannot introduce new terms and consequently it does not compromize the termination argument of Proposition 5.5.

Theorem 7.1. *Let T be the theory $\bigcup_{f\in\Sigma}\{\forall x\ (x = \mathtt{undef} \leftrightarrow f(x) = \mathtt{undef})\}$. Suppose that the algorithm from Section 5, taking as input the primitive $\underline{e}$-flat formula $\exists\underline{e}\,\phi(\underline{e},\underline{y})$, gives as output the quantifier-free formula $\psi(\underline{y})$. Then the latter is a T-cover of $\exists\underline{e}\,\phi(\underline{e},\underline{y})$.* ◁

Proof. The proof of Theorem 5.6 can be easily adjusted as follows. We proceed as in the proof of Theorem 5.6, so as to obtain the set $\Delta(\mathcal{M}) \cup R \cup \tilde{S}$ which is saturated in the standard (unconstrained) Superposition Calculus. Below, we refer to the general refutational completeness proof of the Superposition Calculus given in [55]. Since we only have unit literals here, in order to produce a model of $\Delta(\mathcal{M}) \cup R \cup \tilde{S}$, we can just consider the convergent ground rewriting system $\rightarrow$ consisting of the oriented equalities in $\Delta^+(\mathcal{M})\cup R\cup\tilde{S}$: the support of such model is formed by the $\rightarrow$-normal forms of our ground terms with the obvious interpretation for the function and constant symbols. For simplicity, we assume that `undef` is in normal form. [12] We need to check that whenever we have[13] $f(t) \rightarrow^* \mathtt{undef}$ then we have also $t \rightarrow^* \mathtt{undef}$: we prove this by induction on the reduction ordering for our ground terms. Let t be a term such that $f(t) \rightarrow^* \mathtt{undef}$: if t is $\underline{e}$-free then the claim is trivial (because the axioms (3) are supposed to hold in $\mathcal{M}$). Suppose also that induction hypothesis applies to all terms smaller than t. If t is not in normal form, then let $\tilde{t}$ be its normal form; then we have $f(t) \rightarrow^+ f(\tilde{t}) \rightarrow^* \mathtt{undef}$, by the fact that $\rightarrow$ is convergent. By induction hypothesis, $\tilde{t} \rightarrow \mathtt{undef}$, hence $t \rightarrow^+ \tilde{t} \rightarrow^* \mathtt{undef}$, as desired. Finally, let us consider the case in which t is in normal form; since $f(t)$ is reducible in root position by some rule $l \rightarrow r$, our rules $l \rightarrow r$ are $\underline{e}$-flat and t is not $\underline{e}$-free, we have that $t \equiv e_j$ for some existential variable e_j. Then, we must have that S contains an equality of the kind $f(e_j) = u(\underline{y}) \parallel D$ or of the kind $f(e_j) = e_i \parallel D$ (the constraint D being true in $\mathcal{M}$ under the given assignment to the $\underline{y}$). The latter case is reduced to the former, since $e_i \rightarrow^* \mathtt{undef}$ (by the convergence of $\rightarrow^*$) and since S is closed under Paramodulation. In the former case, by the rule $Ext(\mathtt{undef})$, we must have that S contains $e_j = \mathtt{undef} \parallel D\cup\{u(\underline{y}) = \mathtt{undef}\}$. Now, since $f(e_j) = u(\underline{y}) \parallel D$ belongs to S and D is true in $\mathcal{M}$, we have that the normal forms of $f(e_j)$ and of $u(\underline{y})$ are the same; since the normal form of $f(e_j)$ is `undef`, the normal form of $u(\underline{y})$ is `undef` too, which means that $u(\underline{y}) = \mathtt{undef}$ is true in $\mathcal{M}$. But $e_j = \mathtt{undef} \parallel D \cup \{u(\underline{y}) = \mathtt{undef}\}$ belongs to S, hence $e_j = \mathtt{undef}$ belongs to $\tilde{S}$, which implies $e_j \rightarrow^* \mathtt{undef}$, as desired. ⊣

8 Remarks on MCMT implementation

As evident from Subsection 4.1, our main motivation for investigating covers originated from the verification of data-aware processes. Such applications require database (DB) signatures to contain only unary function symbols (besides rela-

[12] To be pedantic, according to the definition of $\Delta^+(\mathcal{M})$, there should be an equality $\mathtt{undef} = c_0$ in $\Delta^+(\mathcal{M})$ so that c_0 is the normal form of `undef`.

[13] We use $\rightarrow^*$ for the reflexive-transitive closure of $\rightarrow$ and $\rightarrow^+$ for the transitive closure of $\rightarrow$.

tions of every arity). We observed that computing covers of primitive formulae in such signatures requires only polynomial time. In addition, if relation symbols are at most binary, *the cover of a primitive formula is a conjunction of literals* (this is due to the fact that the constrained literals produced during saturation either have empty constraints or are of the kind $\bot \,\|\, t_1 = t_2$): this is crucial in applications, because model checkers like MCMT [32] and CUBICLE [19] represent sets of reachable states as primitive formulae. This makes cover computations a quite attractive technique in verification of data-aware processes.

Our cover algorithm for DB signatures has been implemented in the model checker MCMT. The implementation is however still partial, nevertheless the tool is able to compute covers for the $\mathcal{EUF}$-fragment with unary function symbols, unary relations and binary relations. The optimized procedure of Section 6 has not yet been implemented, instead MCMT uses a customary Knuth-Bendix completion (in fact, for the above mentioned fragments the constraints are always trivial and our constrained Superposition Calculus essentially boils down to Knuth-Bendix completion for ground literals in $\mathcal{EUF}$).

Axioms (3) are also covered in the following way. We assume that constraints of which we want to compute the cover always contain either the literal $e_j = \texttt{undef}$ or the literal $e_j \neq \texttt{undef}$ for every existential variable e_j. Whenever a constraint contains the literal $e_j \neq \texttt{undef}$, the completion procedure adds the literal $u(y_i) \neq \texttt{undef}$ whenever it had produced a literal of the kind $f(e_j) = u(y_i)$.[14]

We wonder whether we are justified in assuming that all constraints of which we want to compute the cover always contain either the literal $e_j = \texttt{undef}$ or the literal $e_j \neq \texttt{undef}$ for every existential variable e_j. The answer is affirmative: according to the backward search algorithm implemented in array-based systems tools, the variable e_j to be eliminated always comes from the guard of a transition and we can assume that such a guard contains the literal $e_j \neq \texttt{undef}$ (if we need a transition with $e_j = \texttt{undef}$ - for an existentially quantified variable e_j - it is possible to write trivially this condition without using a quantified variable). The MCMT User Manual (available from the distribution) contains precise instructions on how to write specifications following the above prescriptions.

A first experimental evaluation (based on the existing benchmark provided in [45], which samples 32 real-world BPMN workflows taken from the BPMN official website `http://www.bpmn.org/`) is described in [11,17]. The benchmark set is available as part of the last distribution 3.0 of MCMT `http://users.mat.unimi.it/users/ghilardi/mcmt/` (see the subdirectory `/examples/dbdriven` of the distribution). The User Manual, also included in the distribution, contains a dedicated section giving essential information on how to encode relational artifact systems (comprising *both first order and second order variables*) in MCMT specifications and how to produce user-defined examples in the database driven framework. The first experiments were very encouraging: the tool was able to solve in few seconds all the proposed benchmarks and the cover computations generated automatically during the model-checking search were discharged instantaneously: see [11,17] for more information about our experiments.

[14] This is sound because $e \neq \texttt{undef}$ implies $f(e) \neq \texttt{undef}$ according to (3), so $u(y_i) \neq \texttt{undef}$ follows from $f(e_j) = u(y_i)$ and $e \neq \texttt{undef}$.

9 Conclusions and Future Work

The above experimental setup motivates new research to extend Proposition 4.5 to further theories axiomatizing integrity constraints used in DB applications. Practical algorithms for the computation of covers in the theories falling under the hypotheses of Proposition 4.5 need to be designed: as a little first example, in Subsection 7 above we showed how to handle Axiom (3) by light modifications to our techniques. Symbol elimination of function and predicate variables should also be combined with cover computations. Combined cover algorithms (along the perspectives in [38]) could be crucial also in this setting: a first attempt to attack this problem, regarding the disjoint signatures combination, can be found in [16].

We consider the present work, together with [17, 13, 12, 28], as the starting point for a full line of research dedicated to SMT-based techniques for the effective verification of data-aware processes [15], addressing richer forms of verification beyond safety (such as liveness, fairness, or full LTL-FO) and richer classes of artifact systems, (e.g., with concrete data types and arithmetics), while identifying novel decidable classes (e.g., by restricting the structure of the DB and of transition and state formulae) beyond the ones presented in [17, 13]. Concerning implementation, we plan to further develop our tool to incorporate in it the plethora of optimizations and sophisticated search strategies available in infinite-state SMT-based model checking. Finally, in [12] we tackle more conventional process modeling notations, concerning data-aware extensions of the de-facto standard BPMN[15]: we plan to provide a full-automated translator from the data-aware BPMN model presented in [12] to the artifact systems setting of [13, 17].

References

1. Baader, F., Ghilardi, S., Tinelli, C.: A new combination procedure for the word problem that generalizes fusion decidability results in modal logics. Inform. and Comput. pp. 1413–1452 (2006)
2. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, United Kingdom (1998)
3. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. J. Log. Comput. **4**(3), 217–247 (1994)
4. Bachmair, L., Ganzinger, H., Lynch, C., Snyder, W.: Basic paramodulation. Inform. and Comput. **121**(2), 172–192 (1995)
5. Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational theorem proving for hierarchic first-order theories. Appl. Algebra Eng. Commun. Comput. **5**, 193–212 (1994)
6. Baumgartner, P., Waldmann, U.: Hierarchic superposition with weak abstraction. In: Proc. of CADE, *LNCS (LNAI)*, vol. 7898, pp. 39–57. Springer (2013)
7. Bílková, M.: Uniform interpolation and propositional quantifiers in modal logics. Studia Logica **85**(1), 1–31 (2007)
8. Bojańczyk, M., Segoufin, L., Toruńczyk, S.: Verification of database-driven systems via amalgamation. In: Proc. of PODS, pp. 63–74 (2013)
9. Bruttomesso, R., Ghilardi, S., Ranise, S.: Quantifier-free interpolation in combinations of equality interpolating theories. ACM Trans. Comput. Log. **15**(1), 5:1–5:34 (2014)
10. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data aware process analysis: A database theory perspective. In: Proc. of PODS (2013)
11. Calvanese, D., Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Verification of data-aware processes via array-based systems (extended version). Technical Report arXiv:1806.11459, arXiv.org (2018)

[15] `http://www.bpmn.org/`

12. Calvanese, D., Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Formal modeling and SMT-based parameterized verification of data-aware BPMN. In: Proc. of BPM, *LNCS*, vol. 11675, pp. 157–175. Springer (2019)
13. Calvanese, D., Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: From model completeness to verification of data aware processes. In: Description Logic, Theory Combination, and All That, *LNCS*, vol. 11560, pp. 212–239. Springer (2019)
14. Calvanese, D., Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Model completeness, covers and superposition. In: Proc. of CADE, *LNCS (LNAI)*, vol. 11716, pp. 142–160. Springer (2019)
15. Calvanese, D., Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Verification of data-aware processes: Challenges and opportunities for automated reasoning. In: Proceedings of the 2nd International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements (ARCADE), vol. 311. EPTCS (2019)
16. Calvanese, D., Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Combined Covers and Beth Definability. In: Proc. of IJCAR, *LNCS (LNAI)*, vol. 12166, pp. 181–200. Springer (2020)
17. Calvanese, D., Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: SMT-based verification of data-aware processes: a model-theoretic approach. Math. Struct. Comput. Sci. **30**(3), 271–313 (2020)
18. Chang, C.C., Keisler, J.H.: Model Theory, third edn. North-Holland Publishing Co., Amsterdam-London (1990)
19. Conchon, S., Goel, A., Krstic, S., Mebsout, A., Zaïdi, F.: Cubicle: A parallel SMT-based model checker for parameterized systems - tool paper. In: Proc. of CAV, pp. 718–724 (2012)
20. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: Proc. of ICDT, pp. 252–267 (2009)
21. Deutsch, A., Li, Y., Vianu, V.: Verification of hierarchical artifact systems. In: Proc. of PODS, pp. 179–194. ACM Press (2016)
22. Ghilardi, S.: An algebraic theory of normal forms. Ann. Pure Appl. Logic **71**(3), 189–245 (1995)
23. Ghilardi, S.: Model theoretic methods in combined constraint satisfiability. J. Autom. Reasoning **33**(3-4), 221–249 (2004)
24. Ghilardi, S., Gianola, A.: Interpolation, amalgamation and combination (the non-disjoint signatures case). In: Proc. of FroCoS, *LNCS (LNAI)*, vol. 10483, pp. 316–332. Springer (2017)
25. Ghilardi, S., Gianola, A.: Modularity results for interpolation, amalgamation and superamalgamation. Annals of Pure and Applied Logic **169**(8), 731–754 (2018)
26. Ghilardi, S., Gianola, A., Kapur, D.: Compactly representing uniform interpolants for EUF using (conditional) DAGS. Technical Report arXiv:2002.09784, arXiv.org (2020)
27. Ghilardi, S., Gianola, A., Kapur, D.: Computing uniform interpolants for EUF via (conditional) DAG-based compact representations. In: Proc. of CILC, *CEUR Workshop Proceedings*, vol. 2710, pp. 67–81. CEUR-WS.org (2020)
28. Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Petri nets with parameterised data - modelling and verification. In: Proc. of BPM, *LNCS*, vol. 12168, pp. 55–74. Springer (2020)
29. Ghilardi, S., van Gool, S.J.: Monadic second order logic as the model companion of temporal logic. In: Proc. of LICS, pp. 417–426 (2016)
30. Ghilardi, S., van Gool, S.J.: A model-theoretic characterization of monadic second order logic on infinite words. J. Symb. Log. **82**(1), 62–76 (2017)
31. Ghilardi, S., Nicolini, E., Zucchelli, D.: A comprehensive framework for combined decision procedures. ACM Trans. Comput. Log. pp. 1–54 (2008)
32. Ghilardi, S., Ranise, S.: MCMT: A model checker modulo theories. In: Proc. of IJCAR, *LNCS (LNAI)*, vol. 6173, pp. 22–29. Springer (2010)
33. Ghilardi, S., Zawadowski, M.: Sheaves, games, and model completions, *Trends in Logic—Studia Logica Library*, vol. 14. Kluwer Academic Publishers, Dordrecht (2002). A categorical approach to nonclassical propositional logics
34. Ghilardi, S., Zawadowski, M.W.: A sheaf representation and duality for finitely presenting heyting algebras. J. Symb. Log. **60**(3), 911–939 (1995)
35. Ghilardi, S., Zawadowski, M.W.: Undefinability of propositional quantifiers in the modal system S4. Studia Logica **55**(2), 259–271 (1995)

36. Ghilardi, S., Zawadowski, M.W.: Model completions, r-Heyting categories. Ann. Pure Appl. Logic **88**(1), 27–46 (1997)
37. van Gool, S.J., Metcalfe, G., Tsinakis, C.: Uniform interpolation and compact congruences. Ann. Pure Appl. Logic **168**(10), 1927–1948 (2017)
38. Gulwani, S., Musuvathi, M.: Cover algorithms and their combination. In: Proc. of ESOP, Held as Part of ETAPS, pp. 193–207 (2008)
39. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: Proc. of SAT, pp. 157–171 (2012)
40. Hsiang, J., Rusinowitch, M.: Proving refutational completeness of theorem-proving strategies: The transfinite semantic tree method. J. ACM **38**(3), 559–587 (1991)
41. Kapur, D.: Shostak's congruence closure as completion. In: Proc. of RTA, pp. 23–37 (1997)
42. Kapur, D.: 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)
43. Kovács, L., Voronkov, A.: Interpolation and symbol elimination. In: Proc. of CADE, *LNCS (LNAI)*, vol. 5663, pp. 199–213. Springer (2009)
44. Kowalski, T., Metcalfe, G.: Uniform interpolation and coherence. Ann. Pure Appl. Logic **170**(7), 825–841 (2019)
45. Li, Y., Deutsch, A., Vianu, V.: VERIFAS: A practical verifier for artifact systems. PVLDB **11**(3), 283–296 (2017)
46. Lipparini, P.: Locally finite theories with model companion. In: Atti della Accademia Nazionale dei Lincei. Classe di Scienze Fisiche, Matematiche e Naturali. Rendiconti, Serie 8, vol. 72. Accademia Nazionale dei Lincei (1982)
47. Ludwig, M., Waldmann, U.: An extension of the Knuth-Bendix ordering with lpo-like properties. In: Proc. of LPAR, pp. 348–362 (2007)
48. McMillan, K.L.: Lazy abstraction with interpolants. In: Proc. of CAV, pp. 123–136 (2006)
49. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM **27**(2), 356–364 (1980)
50. Nicolini, E., Ringeissen, C., Rusinowitch, M.: Data structures with arithmetic constraints: a non-disjoint combination. In: Proc. of FroCoS, *LNCS (LNAI)*, vol. 5749, pp. 319–334. Springer (2009)
51. Nicolini, E., Ringeissen, C., Rusinowitch, M.: Satisfiability procedures for combination of theories sharing integer offsets. In: Proc. of TACAS, *LNCS*, vol. 5505, pp. 428–442. Springer (2009)
52. Nicolini, E., Ringeissen, C., Rusinowitch, M.: Combining satisfiability procedures for unions of theories with a shared counting operator. Fund. Inform. pp. 163–187 (2010)
53. Nieuwenhuis, R., Oliveras, A.: Fast congruence closure and extensions. Inf. Comput. **205**(4), 557–580 (2007)
54. Nieuwenhuis, R., Rubio, A.: Theorem proving with ordering and equality constrained clauses. J. Symb. Comput. **19**(4), 321–351 (1995)
55. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Handbook of Automated Reasoning (in 2 volumes), pp. 371–443. MIT Press (2001)
56. Pitts, A.M.: On an interpretation of second order quantification in first order intuitionistic propositional logic. J. Symb. Log. **57**(1), 33–52 (1992)
57. Robinson, A.: On the metamathematics of algebra. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Co., Amsterdam (1951)
58. Rybina, T., Voronkov, A.: A logical reconstruction of reachability. In: Perspectives of Systems Informatics, 5th International Andrei Ershov Memorial Conference, PSI 2003, Revised Papers, pp. 222–237 (2003)
59. Shavrukov, V.: Subalgebras of diagonalizable algebras of theories containing arithmetic. Dissertationes Mathematicae **CCCXXIII** (1993)
60. Sofronie-Stokkermans, V.: On interpolation and symbol elimination in theory extensions. In: Proc. of IJCAR, *LNCS (LNAI)*, vol. 9706, pp. 273–289. Springer (2016)
61. Sofronie-Stokkermans, V.: On interpolation and symbol elimination in theory extensions. Log. Methods Comput. Sci. **14**(3) (2018)
62. Vianu, V.: Automatic verification of database-driven systems: a new frontier. In: Proc. of ICDT, pp. 1–13 (2009)
63. Visser, A.: Uniform interpolation and layered bisimulation. In: P. Hájek (ed.) Gödel 96: Logical foundations on mathematics, computer science and physics – Kurt Gödel's legacy. Springer Verlag (1996)
64. Wheeler, W.H.: Model-companions and definability in existentially complete structures. Israel J. Math. **25**(3-4), 305–330 (1976)