Quantifier Elimination for Database Driven Verification

Diego Calvanese[1] , Silvio Ghilardi[2], Alessandro Gianola[1] ,
Marco Montali[1], Andrey Rivkin[1]

[1]Faculty of Computer Science, Free University of Bozen-Bolzano (Italy)
[2]Dipartimento di Matematica, Università degli Studi di Milano (Italy)

Abstract. Running verification tasks in database driven systems requires solving quantifier elimination problems (not including arithmetic) of a new kind. In this paper, we supply quantifier elimination algorithms based on Knuth-Bendix completions and begin studying the complexity of these problems, arguing that they are much better behaved than their arithmetic counterparts. This observation is confirmed by analyzing the preliminary results obtained using the MCMT tool on the verification of data-aware process benchmarks. These benchmarks can be found in the last version of the tool distribution. *The content of this manuscript is very preliminary, its role being simply that of expanding the documentation available from* MCMT *v. 2.8 distribution.*

1 Introduction

During the last two decades, the (fundamental) problem studying integrated management of business processes and master data received great attention in academia and the industry [24,16,23]. In its core, the problem requires a change of an entrenched control-flow perspective adopted within the business process community to a more holistic approach that moves towards considering how data are manipulated and evolved by the process, and how the flow of activities is affected by the presence of data as well as the evaluation of data-driven decisions.

In the light of this recent development, two main lines of research emerged: one on the development of integrated models for processes and data [22], and the other on their static analysis and verification [8]. Many various concrete languages (as well as software platforms for their modeling and enactment) for data-aware processes spawned from the first line of research. The main unifying theme for such approaches is a shift from standard activity-centric models to data-centric ones, where the focus is put on key business entities of the organization, integrating their structural and behavioral (lifecycle) aspects. This resulted in the creation of various languages and frameworks [20,19] for modeling and execution such as IBM's declarative rule-based Guard-Stage-Milestone (GSM) notation [13], OMG's modeling standard CMMN (Case Management Model and Notation)[1] and object-aware PHILharmonic Flows [20].

[1] `http://www.omg.org/spec/CMMN/`

In turn, the second line of research resulted in a series various results on the boundaries of decidability and complexity for the static analysis of data-aware processes [26,8]. It is worth noting that formal models adopted along this line of research can be divided into two main classes. The first one considers very general data-aware processes that evolve a (full-fledged) relational database with integrity constrains by means of atomic create-read-update-delete operations that may introduce new values[2] [6,5,1,9]. Here, verification tasks take an initial database instance as input and proof desired properties by constructing an infinite-state transition system (whose states are labeled wit database instances) considering all possible process evolutions. Conversely, the second class adopts artifact-centric processes [14,12] with the underlying formal model based on: *(i)* a read-only relational database that stores fixed, background information, *(ii)* a working memory that stores the evolving state of artifacts, and *(iii)* actions that update the working memory. Different variants of this model have been considered towards decidability of verification, by carefully tuning the relative expressive power of these three components. The most interesting settings consider pure relational structures with a single-tuple working memory [7], and artifact systems operating over a read-only database equipped with constraints and tracking the co-evolution of multiple, unboundedly many artifacts [15]. Even though in these works the working memory can be updated only using values from the read-only database (i.e., no fresh values can be injected), verification is extremely challenging as it is studied parametrically to the read-only database itself, thus requiring to check infinitely many finite transition systems. This is done to assess whether the system behaves well irrespectively of the read-only data it operates on.

In [10], we propose a generalized model for artifact-centric systems and focus on the *(parameterized) safety problem*, which amounts to determining whether there exists an instance of the read-only database that allows the system to evolve from its initial configuration to an *undesired* configuration falsifying a given state property. We study this problem by establishing for the first time a bridge between verification of artifact-centric systems and model checking based on Satisfiability-Modulo-Theories (SMT). Specifically, our approach is grounded in *array-based systems* – a declarative formalism originally introduced in [17,18] to handle the verification of distributed systems (parameterized on the number of interacting processes), and afterwards successfully employed also to attack the static analysis of other types of systems [3,2]. The overall state of the system is typically described by means of arrays indexed by process identifiers, and used to store the content of process variables like locations and clocks. These arrays are genuine *second order* variables. In addition, *quantified formulae* are used to represent sets of system states. These formulae together with second order function variables form the core of the model checking methodologies developed in [17,18] and following papers. The declarative formalism of array-based systems is exploited as the model-theoretic framework of the tool MCMT. This tool

[2] The values are taken from an infinite data domain

manages the verification of infinite-state systems by implementing a symbolic version of the *backward reachability algorithm*.

In the work [10] we encode artifact systems into array-based systems by providing a "functional view" of relational theories endowed with primary and foreign key dependencies, where the read-only database and the artifact relations forming the working memory are represented with *sorted unary function symbols*. The resulting framework, however, requires novel and non-trivial extensions of the array-based technology to make it operational. In fact, quantifiers are handled in array-based systems both by their instantiation and elimination. While the first can be transposed to the new framework leveraging the Herbrand Theorem, the latter becomes problematic due to the following reason: quantified data variables do not range over simple data types (e.g., integers, reals or enumerated sets) as in standard array-based systems, but instead refer to the content of a full-fledged (read-only) relational database. To overcome this problem, we employ classic model-theoretic machinery, namely *model completions* [25], using which we prove that the runs of the systems we are interested in can be lifted w.l.o.g. to richer contexts – so-called *random-like structures* –, where quantifier elimination is indeed available, despite the fact that it was not available in the original setting. This allows us to recast the original safety problem into an equivalent safety problem in the richer setting where quantifier elimination is available. Specifically, the quantifier elimination permits to resort for symbolic representation of sets of reachable states without using quantifiers over data taken from the read-only database.

The described quantifier elimination is the central topic of this paper. Note that, in order to be able to eliminate quantifiers from the data variables, it is important to study algorithms that could correctly perform this task. Specifically, we aim at developing formal procedures that eliminate quantifiers in the model completions of the theories of sorted unary functions mentioned before. In order to realize these procedures, we employ techniques based on Knuth-Bendix completions that not only show the correctness of the proposed approach, but also guarantee its computational efficiency. These procedures have been already partially implemented in MCMT version 2.8.

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 (the adaptation to multi-sorted languages is straightforward). 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 $\underline{x}$.

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.

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

We give now the definitions of constraint satisfiability problem and quantifier elimination for a theory T.

A Σ-formula ϕ is a *Σ-constraint* (or just a constraint) iff it is a conjunction of literals. The constraint satisfiability problem for T is the following: we are given an existential formula[3] $\exists \underline{y}\, \phi(\underline{x}, \underline{y})$ and we are asking whether there exist a model $\mathcal{M}$ of T and an assignment α to the free variables $\underline{x}$ such that $\mathcal{M}, \alpha \models \exists \underline{y}\, \phi(\underline{x}, \underline{y})$.

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 conjunction of literals (i.e. of atomic formulae and their negations). 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 logic and model theory. We focus on the definitions of diagram, embedding, substructure and amalgamation.

2.1 Substructures and embeddings

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 $\mathcal{A}$, the signature Σ can be expanded to a new signature $\Sigma^{|\mathcal{A}|} := \Sigma \cup \{\bar{a} \mid a \in |\mathcal{A}|\}$ by adding a set of new constants $\bar{a}$ (the *name* for a), one for each element a in $\mathcal{A}$, with the convention that two distinct elements are denoted by different "name" constants. $\mathcal{A}$ can be expanded to a $\Sigma^{|\mathcal{A}|}$-structure $\mathcal{A}' := (\mathcal{A}, a)_{a \in |\mathcal{A}|}$ just interpreting the additional costants over the corresponding elements. From now on, when the meaning is clear from the context, we will freely use the notation $\mathcal{A}$ and $\mathcal{A}'$ interchangeably: in particular, given a Σ-structure $\mathcal{A}$ and a Σ-formula $\phi(\underline{x})$ with free variables that are all in $\underline{x}$, we will write, by abuse of notation, $\mathcal{A} \models \phi(\underline{a})$ instead of $\mathcal{A}' \models \phi(\underline{\bar{a}})$.

A *Σ-homomorphism* (or, simply, a homomorphism) between two Σ-structures $\mathcal{M}$ and $\mathcal{N}$ is any mapping $\mu : |\mathcal{M}| \longrightarrow |\mathcal{N}|$ among the support sets $|\mathcal{M}|$ of

[3] For the purposes of this definition, we may equivalently take ϕ to be quantifier-free.

$\mathcal{M}$ and $|\mathcal{N}|$ of $\mathcal{N}$ satisfying the condition

$$\mathcal{M} \models \varphi \quad \Rightarrow \quad \mathcal{N} \models \varphi \tag{1}$$

for all $\Sigma^{|\mathcal{M}|}$-atoms φ (here $\mathcal{M}$ 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 $\mu(a)$). In case condition (1) holds for all $\Sigma^{|\mathcal{M}|}$-literals, the homomorphism μ is said to be an *embedding* and if it holds for all first order formulae, the embedding μ is said to be *elementary*. Notice the following facts:

(a) since we have equality in the signature, an embedding is an injective function;
(b) an embedding $\mu : \mathcal{M} \longrightarrow \mathcal{N}$ must be an algebraic homomorphism, that is for every n-ary function symbol f and for every $m_1, ..., m_n$ in $|\mathcal{M}|$, we must have $f^{\mathcal{N}}(\mu(m_1), ..., \mu(m_n)) = \mu(f^{\mathcal{M}}(m_1, ..., m_n))$;
(c) for an n-ary predicate symbol P we must have $(m_1, ..., m_n) \in P^{\mathcal{M}}$ iff $(\mu(m_1), ..., \mu(m_n)) \in P^{\mathcal{N}}$.

It is easily seen that an embedding $\mu : \mathcal{M} \longrightarrow \mathcal{N}$ can be equivalently defined as a map $\mu : |\mathcal{M}| \longrightarrow |\mathcal{N}|$ satisfying the conditions (a)-(b)-(c) above. If $\mu : \mathcal{M} \longrightarrow \mathcal{N}$ is an embedding which is just the identity inclusion $|\mathcal{M}| \subseteq |\mathcal{N}|$, we say that $\mathcal{M}$ is a *substructure* of $\mathcal{N}$ or that $\mathcal{N}$ is an *extension* of $\mathcal{M}$. A Σ-structure $\mathcal{M}$ is said to be *generated by* a set X included in its support $|\mathcal{M}|$ iff there are no proper substructures of $\mathcal{M}$ including X.

The notion of substructure can be equivalently defined as follows: given a Σ-structure $\mathcal{N}$ and a Σ-structure $\mathcal{M}$ such that $|\mathcal{M}| \subseteq |\mathcal{N}|$, we say that $\mathcal{M}$ is a *Σ-substructure* of $\mathcal{N}$ if:

- for every function symbol f inf Σ, the interpretation of f in $\mathcal{M}$ (denoted using $f^{\mathcal{M}}$) is the restriction of the interpretation of f in $\mathcal{N}$ to $|\mathcal{M}|$ (i.e. $f^{\mathcal{M}}(m) = f^{\mathcal{N}}(m)$ for every m in $|\mathcal{M}|$); this fact implies that a substructure $\mathcal{M}$ must be a subset of $\mathcal{N}$ which is closed under the application of $f^{\mathcal{N}}$.
- for every relation symbol P in Σ and every tuple $(m_1, ..., m_n) \in |\mathcal{M}|^n$, $(m_1, ..., m_n) \in P^{\mathcal{M}}$ iff $(m_1, ..., m_n) \in P^{\mathcal{N}}$, which means that the relation $P^{\mathcal{M}}$ is the restriction of $P^{\mathcal{N}}$ to the support of $\mathcal{M}$.

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

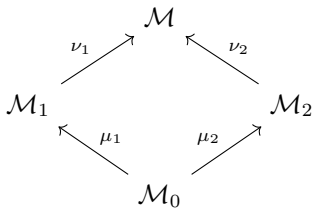
2.2 Robinson Diagrams and Amalgamation

Let $\mathcal{A}$ be a Σ-structure. The *diagram* of $\mathcal{A}$, denoted by $\Delta_\Sigma(\mathcal{A})$, is defined as the set of ground $\Sigma^{|\mathcal{A}|}$-literals (i.e. atomic formulae and negations of atomic formulae) that are true in $\mathcal{A}$. For the sake of simplicity, once again by abuse of notation, we will freely say that $\Delta_\Sigma(\mathcal{A})$ is the set of $\Sigma^{|\mathcal{A}|}$-literals which are true in $\mathcal{A}$.

An easy but nevertheless important basic result, called *Robinson Diagram Lemma* [11], says that, given any Σ-structure $\mathcal{B}$, the embeddings $\mu : \mathcal{A} \longrightarrow \mathcal{B}$ are in bijective correspondence with expansions of $\mathcal{B}$ to $\Sigma^{|\mathcal{A}|}$-structures which are models of $\Delta_\Sigma(\mathcal{A})$. The expansions and the embeddings are related in the obvious way: $\bar{a}$ is interpreted as $\mu(a)$.

Amalgamation is a classical algebraic concept. We give the formal definition of this notion.

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 $\nu_1 \circ \mu_1 = \nu_2 \circ \mu_2$*



The triple $(\mathcal{M}, \mu_1, \mu_2)$ (or, by abuse, $\mathcal{M}$ itself) is said to be a T-amalgama of $\mathcal{M}_1, \mathcal{M}_2$ over $\mathcal{M}_0$

3 Read-only Database Schemas

In this section, we provide a formal definition of (read-only) DB-schemas by relying on an algebraic, functional characterization.

Definition 3.1. *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 only symbols are equality, unary functions, and constants; (ii) T is a* DB theory*, that is, a set of universal Σ-sentences.*

Given a DB signature Σ, we respectively denote by Σ_{srt} and Σ_{fun} the set of sorts and functions in Σ. In the following, we sometimes omit the explicit definition of DB schema, and refer directly to a (DB) theory T with a (DB) signature Σ.

We assume that in a DB signature Σ all function and constant symbols are typed. Thus, every function symbol has a *source* and a *target*: given a function symbol f in Σ_{fun}, we write $f : S \longrightarrow S'$ to say that S is the source of S and S' is the target of f, where S and S' are sorts from Σ. Constant symbols are also sorted. 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. Consequently, sorts are implicitly determined by the context: if we write $g(f(\mathsf{c}))$, we implicitly get that the sort of constant c is the source of f, and that the target sort of f is the source sort of g. Since only unary function symbols and equality are allowed in Σ, all atomic Σ-formulae are of

the form $t_1(v_1) = t_2(v_2)$, where t_1, t_2 are possibly complex terms, and v_1, v_2 are either variables or constants.

We associate to a DB signature Σ a characteristic graph $G(\Sigma)$ capturing the dependencies that are induced by functions over sorts. Specifically, $G(\Sigma)$ is an edge-labeled graph whose nodes are the sorts in Σ_{srt}, and such that $G(\Sigma)$ contains a labeled edge $S \xrightarrow{f} S'$ if and only if Σ_{fun} contains function symbol $f : S \longrightarrow S'$. We say that Σ is *acyclic* if $G(\Sigma)$ is so. The *leaves* of Σ are the nodes of $G(\Sigma)$ without outgoing edges. From a pragmatic point of view, 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. Whenever needed, we identify the set of id sorts in Σ by Σ_{ids}, and that of value sorts by Σ_{val} (recall that $\Sigma_{srt} = \Sigma_{ids} \uplus \Sigma_{val}$).

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

Definition 3.2. *A* DB instance *of DB schema $\langle \Sigma, T \rangle$ is a Σ-structure $\mathcal{M}$ such that: (i) $\mathcal{M}$ is a model of T, and (ii) every id sort of Σ is interpreted by $\mathcal{M}$ on a* finite *set.*

As usual, a DB instance has to be distinguished from an arbitrary *model* of T, where no finiteness assumption is posed on the interpretation of id sorts. What may appear as not customary in Definition 3.2 is the fact that value sorts can be interpreted on infinite sets. This allows us, at once, to reconstruct the classical notion of DB instance as a finite model (since only finitely many values can be pointed from id sorts using functions), at the same time supplying a potentially infinite set of fresh values to be dynamically introduced in the working memory during the evolution of the artifact system. 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), 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 $U^{\mathcal{M}}$.

We close our discussion on the formalization of DB schemas by discussing DB theories. The role of a DB theory is to encode background axioms to express constraints on the different elements of the corresponding signature. 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 AAS whose working memory is initially undefined, in the style of [15,21]. To accommodate this, we add to every sort S of Σ a constant $\texttt{undef}_S$ (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 add the following axioms to the DB theory:

$$\forall x \ (x = \texttt{undef} \leftrightarrow f(x) = \texttt{undef}) \tag{2}$$

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.

Remark 3.3. In the remainder of the paper, we always implicitly assume that the DB theory consists of Axiom 2, but our technical results are not bound to this specific choice. The specific conditions we require on the DB Theory towards our results will be explicitly stated later.

As shown in [10], the algebraic, functional characterization of DB schema and instance can be actually reinterpreted in the classical, relational model. Definition 3.1 naturally corresponds to the definition of relational database schema equipped with single-attribute *primary keys* and *foreign keys* (plus a reformulation of constraint (2)). In order to do so, 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 id sort $S \in \Sigma_{ids}$ 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 $f \in \Sigma_{fun}$ 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, we obtain the following constraints for R_S:

- For each non-id attribute a_f of R_S, we get a functional dependency from id_S to a_f. Altogether, such dependencies in turn witness that id_S is the *(primary) key* of R_S.
- 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 $R_{S'}$.

Given a DB instance $\mathcal{M}$ of $\langle \Sigma, T \rangle$, its corresponding relational instance $\mathcal{I}$ is the minimal set satisfying the following property: for every id sort $S \in \Sigma_{ids}$, let $f_1, \ldots, f_n$ be all functions in Σ with domain S; then, for every identifier $\mathtt{o} \in S^{\mathcal{M}}$, $\mathcal{I}$ contains a *labeled fact* of the form $R_S(id_S : \mathtt{o}^{\mathcal{M}}, a_{f_1} : f_1^{\mathcal{M}}(\mathtt{o}^{\mathcal{M}}), \ldots, a_{f_n} : f_n^{\mathcal{M}}(\mathtt{o}^{\mathcal{M}}))$. With this interpretation, the active domain of $\mathcal{I}$ is the finite set

$$\bigcup_{S \in \Sigma_{ids}} (S^{\mathcal{M}} \setminus \{\mathtt{undef}^{\mathcal{M}}\}) \cup \left\{ \mathtt{v} \in \bigcup_{V \in \Sigma_{val}} V^{\mathcal{M}} \;\middle|\; \begin{array}{l} \text{there exist } f \in \Sigma_{fun} \\ \text{and } \mathtt{o} \in dom(f^{\mathcal{M}}) \text{ s.t. } f^{\mathcal{M}}(\mathtt{o}) = \mathtt{v} \end{array} \right\}$$

consisting of all (proper) identifiers assigned by $\mathcal{M}$ to id sorts, as well as values obtained in $\mathcal{M}$ via the application of some function. Since such values are necessarily finitely many, one may wonder why in Definition 3.2 we allow for interpreting value sorts over infinite sets. The reason is that, in our framework, an evolving artifact system may use such infinite provision to inject and manipulate new values into the working memory.

4 Quantifier Elimination and Model Completion for DB schemata

We fix a DB signature Σ and a DB theory T as in Definition 3.1.

A DB theory T (in the sense of Definition 3.1) need not eliminate quantifiers; it is however often possible to strengthen T in a conservative way (with respect to constraint satisfiability) and get quantifier elimination. We say that 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 which is satisfiable in a model of T is satisfiable in a model of T^*; (ii) T^* eliminates quantifiers.

The following Lemma gives a useful folklore technique for finding model completions:

Lemma 4.1. *Suppose that for every primitive Σ-formula $\exists x\, \phi(x, \underline{y})$ it is possible to find a quantifier-free formula $\psi(\underline{y})$ such that*

(i) $T \models \forall x\, \forall \underline{y}\, (\phi(x, \underline{y}) \rightarrow \psi(\underline{y}))$;
(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 $\mathcal{N} \models \exists x \phi(x, \underline{a})$.*

Then T has a model completion T^* axiomatized by the infinitely many sentences [4]

$$\forall \underline{y}\, (\psi(\underline{y}) \rightarrow \exists x\, \phi(x, \underline{y})) \ . \tag{3}$$

Proof. From (i) and (3) 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 teliminate 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) and (3).

Let $\mathcal{M}$ be a model of T. We show (by using a chain argument) that there exists a model $\mathcal{M}'$ of $T^\star$ such that $\mathcal{M}$ embeds into $\mathcal{M}'$. For every primitive formula $\exists x \phi(x, \underline{y})$, consider the set $\{(\underline{a}, \exists x \phi(x, \underline{a}))\}$ such that $\mathcal{M} \models \psi(\underline{a})$ (where ψ is related to ϕ as in (i)). By Zermelo's Theorem, the set $\{(\underline{a}, \exists x \phi(x, \underline{a}))\}$ can be well-ordered: let $\{(\underline{a}_i, \exists x \phi_i(x, \underline{a}_i))\}_{i \in I}$ be such a well-ordered set (where I is an ordinal). By transfinite induction on this well-order, we define $\mathcal{M}_0 := \mathcal{M}$ and, for each $i \in I$, $\mathcal{M}_{i+1}$ as the extension of $\mathcal{M}_i$ such that $\mathcal{M}_{i+1} \models \exists x \phi(x, \underline{y})$, which exists for (ii) since $\mathcal{M}_i \models \psi(a)$ (remember that validity of ground formulae is preserved passing through substructures and superstructures, and $\mathcal{M}_0 \models \psi(a)$).

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, and it is possible to construct an analogous chain $\mathcal{M}^2$ as done above, starting from $\mathcal{M}^1$ instead of $\mathcal{M}$. Clearly, we get $\mathcal{M}_0 := \mathcal{M} \subseteq \mathcal{M}^1 \subseteq \mathcal{M}^2$

[4] Notice that our T is assumed to be universal according to Definition 3.1, whereas T^* turns out to be universal-existential.

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 (3). The last fact can be shown using the following finiteness argument.

Fix ϕ, ψ as in (3). For every tuple $\underline{a}' \in \mathcal{M}'$ such that $\mathcal{M}' \models \psi(\underline{a}')$, by definition of $\mathcal{M}'$ there exists a natural number k such that $\underline{a}' \in \mathcal{M}^k$: since $\psi(a')$ is a ground formula, we get that also $\mathcal{M}^k \models \psi(\underline{a}')$. Therefore, we consider the step k of the countable chain: there, we have that the pair $(\underline{a}', \psi(\underline{a}'))$ appears in the enumeration given by the well-ordered set $\{(\underline{a}_i, \exists x \phi_i(x, \underline{a}_i))\}_{i \in I}$ (for such ordinal I) such that $\mathcal{M}^k \models \psi_i(\underline{a})$. Hence, by construction and since $\psi(\underline{a}')$ is a ground formula, we have that there exists a $j \in I$ such that $\mathcal{M}^k_j \models \psi(\underline{a}')$ and $\mathcal{M}^k_{j+1} \models \exists x \phi(x, \underline{a}')$. In conclusion, since the existential formulae are preserved passing to extensions, we obtain $\mathcal{M}' \models \exists x \phi(x, \underline{a}')$, as wanted. ⊣

Observe that if Σ is acyclic, there are only finitely many terms involving a single variable x: in fact, there are as many terms as paths in $G(\Sigma)$ starting from the sort of x. If k_Σ is the maximum number of terms involving a single variable, then (since all function symbols are unary) there are at most k^n_Σ terms involving n variables.

The following proposition shows an interesting family of theories T that admit model completion, and gives an explicit algorithm for quantifier elimination in their model completions T^*.

Theorem 4.2. *T has a model completion in case it is axiomatized by universal one-variable formulae and Σ is acyclic.*

Proof. We freely take inspiration from an analogous result in [27]. We preliminarly show that T is amalgamable. Then, for a suitable choice of ψ suggested by the acyclicity assumption, the amalgamation property will be used to prove the validy of the condition (ii) of Lemma 4.1: this fact (together with condition (i)) yields that T has a model completion which is axiomatized by the infinitely many sentences (3).

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ two models of T with a submodel $\mathcal{M}_0$ of T in common (we suppose for simplicity that $|\mathcal{M}_1| \cap |\mathcal{M}_2| = |\mathcal{M}_0|$). We define a T-amalgam $\mathcal{M}$ of $\mathcal{M}_1, \mathcal{M}_2$ over $\mathcal{M}_0$ as follows (we use in an essential way the fact that Σ contains only *unary* function symbols). Let the support of $\mathcal{M}$ be the set-theoretic union of the supports of $\mathcal{M}_1$ and $\mathcal{M}_2$, i.e. $|\mathcal{M}| := |\mathcal{M}_1| \cup |\mathcal{M}_2|$. $\mathcal{M}$ has a natural Σ-structure inherited by the Σ-structures $\mathcal{M}_1$ and $\mathcal{M}_2$: for every function symbol f in Σ, we define, for each $m_i \in |\mathcal{M}_i| (i = 1, 2)$, $f^{\mathcal{M}}(m_i) := f^{\mathcal{M}_1}(m_i)$, i.e. the interpretation of f in $\mathcal{M}$ is the restriction of the interpretation of f in $\mathcal{M}_i$ for every element $m_i \in |\mathcal{M}_i|$. This is well-defined since, for every $a \in |\mathcal{M}_1| \cap |\mathcal{M}_2| = |\mathcal{M}_0|$, we have that $f^{\mathcal{M}}(a) := f^{\mathcal{M}_1}(a) = f^{\mathcal{M}_0}(a) = f^{\mathcal{M}_2}(a)$. It is clear that $\mathcal{M}_1$ and $\mathcal{M}_2$ are substructures of $\mathcal{M}$, and their inclusions agree on $\mathcal{M}_0$.

We show that the Σ-structure $\mathcal{M}$, as defined above, is a model of T. By hypothesis, T is axiomatized by universal one-variable formulae: so, we can consider T as a theory formed by axioms ϕ which are universal closures of clauses with just one variable, i.e. $\phi := \forall x(A_1(x) \wedge ... \wedge A_n(x) \rightarrow B_1(x) \vee ... \vee B_m(x))$, where A_j and B_k ($j = 1, ..., n$ and $k = 1, ..., m$) are atoms.

We show that $\mathcal{M}$ satisfies all such formulae ϕ. In order to do that, suppose that, for every $a \in |\mathcal{M}|$, $\mathcal{M} \models A_j(a)$ for all $j = 1, ..., n$. If $a \in |\mathcal{M}_i|$, then $\mathcal{M} \models A_j(a)$ implies $\mathcal{M}_i \models A_j(a)$, since $A_j(a)$ is a ground formula. Since $\mathcal{M}_i$ is model of T and so $\mathcal{M}_i \models \phi$, we get that $\mathcal{M}_i \models B_k(a)$ for some $k = 1, ..., m$, which means that $\mathcal{M} \models B_k(a)$, since $B_k(a)$ is a ground formula. Thus, $\mathcal{M} \models \phi$ for every axiom ϕ of T, i.e. $\mathcal{M} \models T$ and, hence, $\mathcal{M}$ is a T-amalgam of $\mathcal{M}_1, \mathcal{M}_2$ over $\mathcal{M}_0$, as wanted

Now, given a primitive formula $\exists x\phi(x, \underline{y})$, we find a suitable ψ such that the hypothesis of Lemma 4.1 holds. We define $\psi(\underline{y})$ as the conjunction of the set of all quantifier-free $\chi(\underline{y})$-formulae such that $\phi(x, \underline{y}) \rightarrow \chi(\underline{y})$ is a logical consequences of T (they are finitely many - up to T-equivalence - because Σ is acyclic). By definition, clearly we have that (i) of Lemma 4.1 holds.

We show that also condition (ii) is satisfied. Let $\mathcal{M}$ be a model of T such that $\mathcal{M} \models \psi(\underline{a})$ for some tuple of elements $\underline{a}$ from the support of $\mathcal{M}$. Then, consider the Σ-substructure $\mathcal{M}[\underline{a}]$ of $\mathcal{M}$ generated by the elements $\underline{a}$: this substructure is finite (since Σ is acyclic), it is a model of T and we trivially have that $\mathcal{M}[\underline{a}] \models \psi(\underline{a})$, since $\psi(\underline{a})$ is a ground formula. In order to prove that there exists an extension $\mathcal{N}'$ of $\mathcal{M}[a]$ such that $\mathcal{N} \models \exists x\phi(x, \underline{a})$, it is sufficient to prove (by the Robinson Diagram Lemma) that the $\Sigma^{|\mathcal{M}[a]|\cup\{e\}}$-theory $\Delta(\mathcal{M}[a]) \cup \{\phi(e, \underline{a})\}$ is T-consistent. For reduction to absurdity, suppose that the last theory is T-inconsistent. Then, there are finitely many literals $l_1(\underline{a}), ..., l_m(\underline{a})$ from $\Delta(\mathcal{M}[a])$ (remember that $\Delta(\mathcal{M}[a])$ is a finite set of literals since $\mathcal{M}[a]$ is a finite structure) such that $\phi(e, \underline{a}) \models_T \neg(l_1(\underline{a}) \wedge ... \wedge l_m(\underline{a}))$. Therefore, defining $A(\underline{a}) := l_1(\underline{a}) \wedge ... \wedge l_m(\underline{a})$, we get that $\phi(e, \underline{a}) \models_T \neg A(\underline{a})$, which implies that $\neg A(\underline{a})$ is one of the $\chi(\underline{y})$-formulae appearing in $\psi(\underline{a})$. Since $\mathcal{M}[\underline{a}] \models \psi(\underline{a})$, we also have that $\mathcal{M}[\underline{a}] \models \neg A(\underline{a})$, which is a contraddiction: in fact, by definition of diagram, $\mathcal{M}[\underline{a}] \models A(\underline{a})$ must hold. Hence, there exists an extension $\mathcal{N}'$ of $\mathcal{M}[\underline{a}]$ such that $\mathcal{N}' \models \exists x\phi(x, \underline{a})$. Now, by amalgamation property, there exists a T-amalgam $\mathcal{N}$ of $\mathcal{M}$ and $\mathcal{N}'$ over $\mathcal{M}[\underline{a}]$: clearly, $\mathcal{N}$ is an extension of $\mathcal{M}$ and, since $\mathcal{N}' \hookrightarrow \mathcal{N}$ and $\mathcal{N}' \models \exists x\phi(x, \underline{a})$, also $\mathcal{N} \models \exists x\phi(x, \underline{a})$ holds, as required.

$\dashv$

The proof of Theorem 4.2 gives an algorithm for quantifier elimination in the model completion. The algorithm works as follows (see the formula (3)): to eliminate the quantifier x from $\exists x\, \phi(x, \underline{y})$ take the conjunction of the clauses $\chi(\underline{y})$ implied by $\phi(x, \underline{y})$. Note that this algorithm is not practically efficient. In fact, better algorithms can be obtained by using Knuth-Bendix procedure, which we are going to study in detail in the following section.

5 Algorithms for quantifier elimination

The algorithm for quantifier elimination suggested by the proof of Theorem 4.2 is highly impractical: it relies on the formula (3), where ψ is in fact obtained by conjoining the clauses $\chi(\underline{y})$ implied by $\phi(x, \underline{y})$.

In this section, we introduce better algorithms for the special theories we are interested in and discuss their complexities. The content of this section gives some details about our implementation in MCMT.

We take as complexity of a quantifier-elimination procedure the time/space cost of applying it to a primitive formula: this reflects the needs of our applications and separates the cost of the procedure itself from other costs related to disjunctive normal form conversions. Notice that array-based model checkers, in order to represent sets of states - in particular, sets of states which are backward reachable - use lists of primitive formulae [5] and it is precisely to these formulae that quantifier elimination in T^* is applied in our tool MCMT.

One of the reasons for the high complexity of quantifier elimination in linear arithmetics is that eliminating quantifiers from a primitive formula does not yield in general a primitive formula: we shall see that in our contexts the situation is different. Another problem in quantifier elimination for linear arithmetic (even in real linear arithmetic, which is handled e.g. by Fourier-Motzkin algorithm) is that the size of terms might grow after eliminating quantified variables - in fact terms are here arbitrary linear polynomials. Again, this is not the case for us: if we show that eliminating quantifiers from a primitive formula $\exists \underline{y}\, \phi(\underline{x}, \underline{y})$ yields a conjunction of literals (and not a conjunction of clauses), then it is clear that the space of the output is polynomially bounded in the length of the tuple $\underline{x}$ (keeping k_Σ as a constant). This may suggest that also the time for the computation might be polynomial in relevant cases. In other words, quantifier elimination in our context is computationally much better behaved than in the arithmetic case, so that more sophisticated machinery (predicate abstraction, interpolants, etc.) used in infinite state model checking to circumvent quantifier elimination might not be needed here.

In all the algorithms below, we make reference to the Knuth-Bendix completion procedure, applied to a set of ground literals. Such procedure always terminates in the ground case, we refer the reader to [4] for the necessary background.

5.1 The Basic Algorithm

We first give an algorithm for the case in which T *is empty* (notice that the algorithm applies also to signatures Σ which may not be acyclic). The steps of the algorithm are the following:

[5] Conjunctions of literals (i.e. matrices of primitive formulae) are often called 'cubes', whence the name 'Cubicle' for the tool developed at LRI-Intel for bakward reachability in array-based systems.

Input: $C := \exists e\ \phi(e, y_1, ..., y_n)$, with $\phi(e, y_1, ..., y_n)$ a conjunction of literals (we write $\underline{y}$ for the tuple $y_1, ..., y_n$).

1. Replace variables $e, y_1, \ldots, y_n$ by free constants - we keep the names $e, y_1, \ldots, y_n$ for these constants.
2. Choose a reduction ordering total for ground terms giving higher precedence to e with respect to all the other symbols (thus equations $t(e) = u(\underline{y})$ are always oriented as $t(e) \to u(\underline{y})$).
3. Run the Knuth-Bendix completion procedure (with simplification) to the literals in ϕ considered as ground literals; let ϕ_c be the conjunction of the literals resulting from the completion.
4. Delete from ϕ_c the literals in which e occurs and terminate.

Output: Let C' be the output.

We assume that in case a literal like $t \neq t$ is produced (while normalizing a negative literal in Step 3 above), then the procedure stops with output $\bot$.

We want to prove that the algorithm is correct in the sense that

Proposition 5.1. *Let T be empty; then the set of axioms $C' \to \exists e\, C$ (varying C among the conjunctions of finite sets of literals) axiomatize the model completion T^* of T.*

Proof. In order to reach our goal, we apply Lemma 4.1; condition (i) of the Lemma follows from the fact that Knuth-Bendix completion manipulates a set of literals only up to logical equivalence. As a consequence, it is sufficient to show the validity of the following Claim, corresponding to condition (ii) of the Lemma.

Claim: given a model $\mathcal{M}$ of T and elements $\underline{b} = b_1, \ldots, b_n$ from the support of $\mathcal{M}$ such that $\mathcal{M} \models C'(\underline{b})$ (where C' is the output formula), $\mathcal{M}$ can be embedded in a model $\mathcal{M}'$ of T such that $\mathcal{M}' \models C(\underline{b})$ (where C is the input formula).

To prove the Claim, we define a Σ-structure $\mathcal{M}'$ which extends $\mathcal{M}$ in the following way (we let $\mathcal{M} = (M, \mathcal{I})$, where $\mathcal{I}$ is the interpretation function, extended to an assignment mapping the $\underline{y}$ to the $\underline{b}$):

- Let N be the set of the normal forms of the terms of the kind $t(e)$ and let $N_0 \subseteq N$ be the set of such normal forms which contain at least an occurrence of e (notice that N_0 can be empty in case the completion procedure produces an equation like $e = t(\underline{y})$ - recall that such equation is oriented as $e \to t(\underline{y})$).
- We define $M' = M \cup N_0$; we extend $\mathcal{I}$ to $\mathcal{I}'$ as follows: (i) $\mathcal{I}'(e)$ is the normal form of e if it belongs to N_0, otherwise it is $t^{\mathcal{I}}$ where $t(\underline{y})$ is the normal form of e; (ii) $\mathcal{I}'(f)(u(e))$ is the normal form of $f(u(e))$ if it belongs to N_0, otherwise it is $t^{\mathcal{I}}$ where $t(\underline{y})$ is the normal form of $f(u(e))$.

An easy induction now shows that for every term $t(e)$ normalizing to some t_0, we have $t(e)^{\mathcal{I}'} = t_0^{\mathcal{I}'}$; moreover, if e occurs in t_0, then $t(e)^{\mathcal{I}'} = t_0$.

It remains to check that $\mathcal{M}' = (M', \mathcal{I}') \models \phi$; this is the same as saying that $\mathcal{M}' = (M', \mathcal{I}') \models \phi_c$, because Knuth Bendix completion operates up to logical equivalence.

Now, literals from ϕ_c not involving e are true in $\mathcal{M}$ and so also in $\mathcal{M}'$; we need to analize equalities and disequalities from ϕ_c where e occurs. These can be of four kinds:

(i) *equalities of the kind* $t(e) = u(e)$: since Knuth Bendix procedure removes trivial equalities and the order is total on ground terms, we must have e.g. $t(e) > u(e)$ and that $u(e)$ is the normal form of $t(e)$, so that the claim is obvious;
(ii) *inequalities of the kind* $t(e) \neq u(e)$: here $t(e)$ and $u(e)$ must both be in normal forms (and different, otherwise the procedure would have output $\perp$), so that once again the claim is immediate;
(iii) *equalities of the kind* $t(e) = u(\underline{y})$: here $t(e)$ normalizes to u, so that the claim holds;
(iv) *inequalities of the kind* $t(e) \neq u(y)$: here $t(e)$ and $u(y)$ are both in normal forms and as a consequence $t^{\mathcal{I}'} = t \neq u^{\mathcal{I}'} \in M$.

This concludes the proof of the above Claim. $\dashv$

Notice that the above algorithm maps a primitive formula to a conjunction of literals (not to a conjunction of clauses). In case of an acyclic signature Σ, it is easily seen to run in polynomial time: in fact, a step of Knuth-Bendix completion (with simplification, in the ground case), always *replaces* an equation by smaller ones and we already observed that, keeping k_Σ constant, there can be only polynomially many terms and equations in a given finite number of variables.

5.2 Extensions

We consider two extensions of the above basic algorithm, both have been implemented in our tool MCMT.

In the first extension, we consider the axiom

$$t(x) = \texttt{undef} \leftrightarrow x = \texttt{undef} \tag{4}$$

for every term t (here we assume to have many constants `undef`, one for every sort). One side of the above axiom is equivalent to the ground literal $t(\texttt{undef}) = \texttt{undef}$ and as such it does not interfer with the completion process and the quantifier elimination procedure (we just add it to our constraint C from the beginning).

To accommodate the other side, it is sufficient to do the following. We split the initial constraint into a disjunction $C_1 \vee C_2$, where C_1 contains the literal $e = \texttt{undef}$ and C_2 contains the literal $e \neq \texttt{undef}$. Then, C_1 is handled in the trivial way (replacing everywhere e with `undef`); as for C_2, we check whether, at the end of the completion, we have an equality like $t(e) = u(y_i)$ in the current

constraint: in that case, we add to the completion the literal $u(y_i) \neq \texttt{undef}$.[6] The above correctness proof can be adjusted as follows to cover this modification. If (by absurd) there is a term $t(e)$ (in which e occurs) such that $t(e)^{\mathcal{I}'} = \texttt{undef}^{\mathcal{I}'}$, then pick a minimal (wrt the ordering) such term t; since $t(e)^{\mathcal{I}'} = \texttt{undef}^{\mathcal{I}'}$, $t(e)$ cannot be in normal form by the definition of $\mathcal{I}'$. Since it is minimal, there is an equality $t(e) = u(y_i)$ in the completion that rewrites $t(e)$ itself (not a subterm!) to its normal form $u(y_i)$. Hence $t(e)^{\mathcal{I}'} = u(y_i)^{\mathcal{I}'}$ and as a consequence $\texttt{undef}^{\mathcal{I}'} = u(y_i)^{\mathcal{I}'}$, which is the same as $\texttt{undef}^{\mathcal{I}} = u(y_i)^{\mathcal{I}}$, but the latter is absurd because $\mathcal{M}$ was a model of $u(y_i) \neq \texttt{undef}$ (because such a literal is added to the completion).

Thus, axioms (4) break our desired property that quantifier elimination applied to a primitive formula produces a conjunction of literals. However, in the implementation, it is possible to assume that $e \neq \texttt{undef}$ always occurs in the matrix of a primitive formula we want to eliminate e from. In fact, according to the backward search algorithm implemented in array-based systems tools, the variable e to be eliminated always comes from the guard of a transition and we can assume that such a guard contains the literal $e \neq \texttt{undef}$ (if we need a transition with $e = \texttt{undef}$ - for an existentially quantified variable e - it is possible to write trivially this condition without using a quantified variable).

In a second extension, we consider the possibility of enriching Σ with *unary and binary (sorted) relation symbols*. These symbols are not used in our formal framework (they would represent relations without a key), but the extension is easy, so we decided to cover it too in our implementation. The modification to the above quantifier elimination algorithm is straightforward. Of course, terms occurring in relational literals are also subject to normalization during the completion phase. For unary relations this observation is sufficient,[7] whereas for binary relations there is the need of the following further operation: if in Step 3, the constraint ϕ_c contains $R(t(e), u_1(y_i)) \wedge \neg R(t(e), u_2(y_j))$ (resp. $R(u_1(y_i), t(e)) \wedge \neg R(u_2(y_j), t(e))$), then $u_1(y_i) \neq u_2(y_j)$ must be added to ϕ_c.[8]

References

1. P. A. Abdulla, C. Aiswarya, M. F. M. M. Atig, and O. Rezine. Recency-bounded verification of dynamic database-driven systems. In *Proc. of PODS*. ACM Press, 2016.
2. F. Alberti, R. Bruttomesso, S. Ghilardi, S. Ranise, and N. Sharygina. An extension of lazy abstraction with interpolation for programs with arrays. *Formal Methods in System Design*, 45(1):63–109, 2014.
3. F. Alberti, S. Ghilardi, and N. Sharygina. A framework for the verification of parameterized infinite-state systems. *Fundam. Inform.*, 150(1):1–24, 2017.

[6] This is sound because $e \neq \texttt{undef}$ implies $t(e) \neq \texttt{undef}$, so $u(y_i) \neq \texttt{undef}$ follows.

[7] Remember that complementary literals nevertheless produce $\bot$ and that this applies to relational atoms too.

[8] Notice that ternary relations would generate disjunctions: $R(t(e), u_1(y_i), v_1(y_r)) \wedge \neg R(t(e), u_2(y_j), v_2(y_s))$ should produce the disjunction $u_1(y_i) \neq u_2(y_j) \vee v_1(y_r) \neq v_2(y_s)$.

4. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, United Kingdom, 1998.
5. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of relational data-centric dynamic systems with external services. In *Proc. of PODS*, 2013.
6. F. Belardinelli, A. Lomuscio, and F. Patrizi. An abstraction technique for the verification of artifact-centric systems. In *Proc. of KR*, 2012.
7. M. Bojańczyk, L. Segoufin, and S. Toruńczyk. Verification of database-driven systems via amalgamation. In *Proc. of PODS*, pages 63–74, 2013.
8. D. Calvanese, G. De Giacomo, and M. Montali. Foundations of data aware process analysis: A database theory perspective. In *Proc. of PODS*, 2013.
9. D. Calvanese, G. De Giacomo, M. Montali, and F. Patrizi. First-order mu-calculus over generic transition systems and applications to the situation calculus. *Inf. and Comp.*, 2017.
10. D. Calvanese, S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin. Verification of data-aware processes via array-based systems. pages 1–12. preprint submitted to PODS 2019.
11. C.-C. Chang and J. H. Keisler. *Model Theory*. North-Holland Publishing Co., Amsterdam-London, third edition, 1990.
12. E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. *ACM TODS*, 37(3), 2012.
13. E. Damaggio, R. Hull, and R. Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with Guard-Stage-Milestone lifecycles. In *Proc. of BPM*, 2011.
14. A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *Proc. of ICDT*, pages 252–267, 2009.
15. A. Deutsch, Y. Li, and V. Vianu. Verification of hierarchical artifact systems. In *Proc. of PODS*, pages 179–194. ACM Press, 2016.
16. M. Dumas. On the convergence of data and process engineering. In *Proc. of ADBIS*, volume 6909 of *LNCS*. Springer, 2011.
17. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Towards SMT model checking of array-based systems. In *Proc. of IJCAR*, pages 67–82, 2008.
18. S. Ghilardi and S. Ranise. Backward reachability of array-based systems by SMT solving: Termination and invariant synthesis. *Logical Methods in Computer Science*, 6(4), 2010.
19. R. Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *Proc. of OTM*, volume 5332 of *LNCS*. Springer, 2008.
20. V. Künzle, B. Weber, and M. Reichert. Object-aware business processes: Fundamental requirements and their support in existing approaches. *Int. J. of Information System Modeling and Design*, 2(2), 2011.
21. Y. Li, A. Deutsch, and V. Vianu. VERIFAS: A practical verifier for artifact systems. *PVLDB*, 11(3):283–296, 2017.
22. A. Meyer, S. Smirnov, and M. Weske. Data in business processes. Technical Report 50, Hasso-Plattner-Institut for IT Systems Engineering, Universität Potsdam, 2011.
23. M. Reichert. Process and data: Two sides of the same coin? In *Proc. of the On the Move Confederated Int. Conf. (OTM 2012)*, volume 7565 of *LNCS*. Springer, 2012.
24. C. Richardson. Warning: Don't assume your business processes use master data. In *Proc. of BPM*, volume 6336 of *LNCS*. Springer, 2010.

25. A. Robinson. *On the metamathematics of algebra*. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Co., Amsterdam, 1951.
26. V. Vianu. Automatic verification of database-driven systems: a new frontier. In *Proc. of ICDT*, pages 1–13, 2009.
27. W. H. Wheeler. Model-companions and definability in existentially complete structures. *Israel J. Math.*, 25(3-4):305–330, 1976.