Mathematical Structures in Computer Science (2020), **30**, pp. 271–313
doi:10.1017/S0960129520000067

CAMBRIDGE
UNIVERSITY PRESS

PAPER

SMT-based verification of data-aware processes: a model-theoretic approach

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

[1]Free University of Bozen-Bolzano and [2]Università degli Studi di Milano
[*]Corresponding author. Email: gianola@inf.unibz.it

(Received 12 December 2018; revised 22 February 2020; accepted 26 February 2020; first published online 3 April 2020)

Abstract

In recent times, satisfiability modulo theories (SMT) techniques gained increasing attention and obtained remarkable success in model-checking infinite-state systems. Still, we believe that whenever more expressivity is needed in order to specify the systems to be verified, more and more support is needed from mathematical logic and model theory. This is the case of the applications considered in this paper: we study verification over a general model of relational, data-aware processes, to assess (parameterized) safety properties irrespectively of the initial database (DB) instance. Toward this goal, we take inspiration from array-based systems and tackle safety algorithmically via backward reachability. To enable the adoption of this technique in our rich setting, we make use of the model-theoretic machinery of model completion, which surprisingly turns out to be an effective tool for verification of relational systems and represents the main original contribution of this paper. In this way, we pursue a twofold purpose. On the one hand, we isolate three notable classes for which backward reachability terminates, in turn witnessing decidability. Two of such classes relate our approach to conditions singled out in the literature, whereas the third one is genuinely novel. On the other hand, we are able to exploit SMT technology in implementations, building on the well-known MCMT (Model Checker Modulo Theories) model checker for array-based systems and extending it to make all our foundational results fully operational. All in all, the present contribution is deeply rooted in the long-standing tradition of the application of model theory in computer science. In particular, this paper applies these ideas in an original mathematical context and shows how these techniques can be used for the first time to empower algorithmic techniques for the verification of infinite-state systems based on arrays, so as to make such techniques applicable to the timely, challenging settings of data-aware processes.

Keywords: Verification of data-aware processes; satisfiability modulo theories; model completeness; well-quasi-orders; database theory

1. Introduction

The main contribution of this paper comes from a rather surprising confluence of two well-established research traditions: *model-theoretic algebra* from mathematical logic and *satisfiability modulo theories* (SMT), an emerging technologically oriented area in computational logic. We believe that such seemingly very different scientific paradigms can indeed cooperate in formal verification, and we shall supply an evidence for this claim by developing an innovative application to the hot topic of the management of dynamic *data-aware processes*. In this introduction, we shall briefly explain how the above-mentioned ingredients fit into the plan of our paper.

1.1 Model-theoretic algebra

Finding solutions to equations is a challenge at the heart of both mathematics and computer science. Model-theoretic algebra, originating with the ground-breaking work of Robinson (1951, 1963), cast the problem of solving equations in a logical form and used this setting to solve algebraic problems via model theory. The central notion is that of an *existentially closed model*, which we explain now. Call a quantifier-free formula with parameters in a model M *solvable* if there is an extension M' of M where the formula is satisfied. A model M is *existentially closed* if any solvable quantifier-free formula already has a solution in M itself. For example, the field of real numbers is not existentially closed, but the field of complex numbers is.

Although this definition is formally clear, it has a main drawback: it is not a first-order notion in general. However, in fortunate and important cases, the class of existentially closed models of T is exactly the models of another first-order theory T^*. In this case, the theory T^* can be characterized abstractly as the *model companion* of T. Model companions become *model completions* (cf. Definition 2.2) in the case of universal theories with the amalgamation property; in such model completions, quantifier elimination holds, unlike in the original theory T. The model companion/model completion of a theory identifies the class of those models where *all satisfiable existential statements can be satisfied*. For example, the theory of algebraically closed fields is the model companion of the theory of fields, and dense linear orders without endpoints give the model companion of linear orders.

In declarative approaches to model checking, the runs of a system are identified with certain definable paths in the models of a theory T: we shall show that, without loss of generality, one may *restrict to paths within existentially closed models*, thus taking profit from the properties enjoyed by the model completion T^* (quantifier elimination being the key property to be carefully exploited).

Model completeness has other well-known applications in computer science. It has been applied to discover interesting connections between temporal logic and monadic second-order logic (Ghilardi and van Gool 2016, 2017). In automated reasoning, it has been used to design complete algorithms for constraint satisfiability in combined theories over non-disjoint signatures (Baader et al. 2006; Ghilardi 2004; Ghilardi et al. 2008b; Nicolini et al. 2009a,b, 2010) and in theory extensions (Sofronie-Stokkermans 2016, 2018). Applications to combined interpolation (both for modal logics and for software verification theories) can be found in Ghilardi and Gianola (2017, 2018).

1.2 Satisfiability modulo theories

The SMT-LIB project http://smtlib.cs.uiowa.edu/ (started in 2003) aims at bringing together people interested in developing powerful tools combining sophisticated techniques in SAT-solving with dedicated decision procedures involving specific theories used in applications (especially in software verification). SMT-tools are at the heart of declarative approaches to model checking, both in the bounded and in the unbounded case: they are employed in many advanced techniques, for instance, in interpolation-based (McMillan 2006) and IC3-based (Hoder and Bjørner 2012) techniques.

Specifically, our approach is grounded in *array-based systems*. Array-based systems are a declarative formalism originally introduced in Ghilardi et al. (2008a), Ghilardi and Ranise (2010a) to handle the verification of distributed systems and afterwards successfully employed also to attack the static analysis of other types of systems (Alberti et al. 2014a, 2017). Distributed systems are parameterized in their essence: the number N of interacting processes within a distributed system is unbounded, and the challenge is that of supplying certifications that are valid for all possible values of the parameter N. 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, *first-order quantifiers* are used to represent sets of system states. Quantified formulae and second-order function variables are at

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

the heart of the model-checking methodologies developed in Ghilardi et al. (2008a), Ghilardi and Ranise (2010a) and following papers.

It is worth noting that the term "array-based systems" is an umbrella term generically referring to transition systems specified with logical formulae having second-order variables (i.e., arrays). The precise formal notion depends on the application and is defined on the spot (in this paper, we shall introduce a specific instance of the notion of an array-based system tailored to database-driven verification).

Model checkers for array-based systems handle safety problems by *backward reachability*: they iteratively *regress* bad states by computing their predecessors, the predecessors of the predecessors, etc., until a fixpoint is reached or the initial state(s) is intersected. This is done symbolically by manipulating logical formulae that describe sets of states. Depending on the specific features of the array-based system, to guarantee the regressability of such formulae, the procedure may require to eliminate existentially quantified variables present in the formula.

There is an historical reason for choosing backward reachability: it was known since the seminal paper (Abdulla et al. 1996) that backward search decides safety problems for a large class of systems, called *well-structured transition systems*. What backward search in array-based systems is meant to achieve is precisely to reproduce the results of Abdulla et al. (1996) in a declarative setting: in such a declarative setting, the abstract wqo underlying well-structured transitions systems is replaced by the standard model-theoretic notion of an *embedding* between finitely generated models (in many practical cases, in fact, such embeddability relation can be proved to be a wqo, using a suitable version of Dickson or of Higman lemma).

Backward search, once done in a declarative symbolic setting, requires discharging proof obligations that can be reduced to satisfiability tests for quantified formulae, albeit of a restricted syntactic shape. This raises the question of how to handle such (first-order) quantifiers. In the original papers (Ghilardi et al. 2008a; Ghilardi and Ranise 2010a), first-order quantifiers were handled in satisfiability tests by *instantiation*, whereas in successive applications (Alberti et al. 2014a; Carioni et al. 2010), *quantifier elimination* was also used to handle quantifiers ranging over data structures (typically, real-valued clocks). There, quantifier elimination was made possible by the fact that theories axiomatizing such data structures were limited to light versions of arithmetics (mostly even strictly included in what is called "difference logic" in SMT terminology), where quantifier elimination is indeed available and at least in the examples arising from benchmarks seems not to be as harmful as in the general arithmetic case. Suitable combinations of quantifier instantiations and quantifier eliminations are needed at the foundational level to design complete algorithms for the satisfiability tests that a model checker for array-based systems has to discharge during search: a specific form of such combination will be developed in this paper too. By means of such combinations, *satisfiability tests involving quantified formulae of special shape are reduced to satisfiability tests at quantifier-free level*, to be very efficiently discharged by existing SMT-solvers (as confirmed by the extensive experiments, see Section 6 for a brief historical account).

1.3 Data-aware processes

To capture data-aware processes, we follow the traditional line of research focused on the formal representation of *artifact systems* (Calvanese et al. 2013; Damaggio et al. 2012; Deutsch et al. 2018, 2009; Hull 2008; Vianu 2009). Since their initial versions (e.g., in Deutsch et al. 2009), such systems are traditionally formalized using three components: *(i) a read-only database (DB)*, storing background information that does *not* change during the system evolution; *(ii) an artifact working memory*, storing data and lifecycle information about artifact(s) that *does* change during the system evolution; *(iii) actions* (also called *services*) that access the read-only DB and the working memory and determine how the working memory itself has to be updated.

Different variants of this framework have been considered toward decidability of verification, by carefully tuning the expressive power of the three components. For instance, for the working memory, radically different models are obtained depending on whether only a single artifact

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

instance is evolved, or whether instead the co-evolution of multiple instances of possibly different artifacts is supported. In particular, early formal models for artifact systems merely considered a fixed set of so-called *artifact variables*, altogether instantiated into a single tuple of data. This, in turn, allows one to capture the evolution of a single artifact instance (Deutsch et al. 2009). We call artifact systems of this form *Simple Artifact System (SAS)*. Instead, more sophisticated types of artifact systems have been studied recently in Deutsch et al. (2016), Li et al. (2017), Deutsch et al. (2019). Here, the working memory is not only equipped with artifact variables as in SAS , but also with so-called *artifact relations*, which supports storing arbitrarily many tuples, each accounting for a different artifact instance that can be separately evolved on its own. We call artifact systems of this form *Relational Artifact System (RAS)*.

Actions are usually specified using logical formulae relating the content of the read-only DB as well as the current configuration of the working memory to (possibly different) next configurations. An applicable action may be executed, nondeterministically transforming the current configuration of working memory in one of such next configurations.

1.4 Bringing all the ingredients together

RASs naturally fit the paradigm of array-based systems: the read-only DB is axiomatized by a suitable universal first-order theory T, and the artifact variables and relations are modeled by second-order variables. The identifiers of the tuples (i.e., the "entries") of the artifact relations play the role of the identifiers of the processes in distributed systems: formally, in both cases, they are just sorted first-order variables whose sort is the domain sort of a second-order function variable.

The resulting framework, however, requires novel and nontrivial extensions of the array-based technology to make it operational. In fact, as we saw, quantifiers are handled in array-based systems both by quantifier instantiation and by quantifier elimination. Quantifier instantiation (ultimately referring to variants of the Herbrand Theorem) can be transposed to the new framework, whereas quantifier elimination becomes problematic. In fact, quantifier elimination should be applied to data variables, which do not simply range over data types (like integers, reals, or enumerated sets) as in standard array-based systems, but instead point to the content of a whole, full-fledged (read-only) relational database, and there is no reason for the theory T axiomatizing it to enjoy quantifier elimination. Here, model-theoretic algebra comes into the picture: we show that, without loss of generality, we can assume that system runs take place in existentially closed structures, so that we can exploit quantifier elimination, *provided T has a model completion*.

The question on whether T admits or not a model completion is related to the way we represent the read-only DB. Model completions exist in case the read-only DB is represented in the most simple way, as consisting on free n-ary relations, not subject to any kind of constraint. However, applications require the introduction of some minimal integrity constraints, like *primary* and *foreign* keys. A naif declarative modeling of such requirements (for instance, via relations which are partially functional) would destroy amalgamation property, thus compromising the existence of model completions. Instead, we propose "functional view" of relations, where the read-only DB and the artifact relations forming the working memory are represented with *sorted unary function symbols*. We introduce formally the above framework in Section 3 (Definitions 3.1 and 3.2): there we supply a detailed example (Example 3.1 and Figure 1) and we also show how to recover the traditional relational model (Subsection 3.1).[1]

1.5 Main contributions

By exploiting the above explained machinery and its model-theoretic properties, we then provide a fourfold contribution.

Our first contribution is to define a general framework for SASs and RASs, in which artifacts are formalized in the spirit of *array-based systems*, one of the most sophisticated setting within the SMT tradition. In this setting, SASs are a particular class of RASs, where only artifact variables

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

are allowed. RASs employ arrays to capture a very rich working memory that simultaneously accounts for artifact variables storing single data elements, and full-fledged artifact relations storing unboundedly many tuples. Each artifact relation is captured using a collection of arrays, so that a tuple in the relation can be retrieved by inspecting the content of the arrays with a given index. The elements stored therein may be fresh values injected into the RAS , or data elements extracted from the read-only DB, whose relations are subject to key and foreign key constraints.

To attack this modeling complexity within array-based systems, RASs encode the read-only DB using a functional, algebraic view, where relations and constraints are captured using multiple sorts and unary functions. To the best of our knowledge, this encoding has never been explored in the past but is essential in our context. In fact, more direct attempts to encode the read-only DB into standard array-based systems would fail; for example, not using unary functions for relations with key dependencies would destroy amalgamability, which is a model-theoretic notion that is crucial toward decidability of verification (Bojańczyk et al. 2013).

Our resulting RAS model captures the essential features of Li et al. (2017), which in turn is tightly related (though incomparable) to one of the most sophisticated formal models for artifact-centric systems of Deutsch et al. (2016, 2019).

Our second contribution is the development of a new version of the backward reachability algorithm employed in traditional array-based systems (Ghilardi et al. 2008a; Ghilardi and Ranise 2010a), making it able to assess safety of RASs (and consequently SASs) in a sound and complete way. The main technical difficulty, which makes the original algorithm not applicable anymore, is that transition formulae in RASs contain special existentially quantified *"data" variables* pointing to the read-only DB, which contains data elements possibly constrained by primary keys and foreign keys. Such data variables are central in our approach as they are needed:

— from the modeling point of view, to equip array-based systems with the ability of querying the read-only DB;
— again for modeling reasons, to express nondeterministic inputs from the external environment, such as users (a feature that is customary in business processes);
— to encode typical forms of updates employed in the artifact-centric literature (Deutsch et al. 2016; Li et al. 2017).

The presence of these quantified data variables constitutes a big leap from traditional array-based systems. In previous works on array-based systems, existentially quantified variables to be eliminated were just arithmetic variables, and the corresponding quantifier elimination procedures were consequently the standard ones studied in the context of arithmetics (such as Fourier–Motzkin and Presburger). Due to the peculiar nature of data variables pointing to the read-only DB, and in particular of the constraints that they must satisfy, such standard techniques do not carry over. Hence, genuinely novel research is needed in order to eliminate new existentially quantified data variables that are introduced during the computation of predecessors in the backward reachability procedure.

From a theoretical point of view, we solve this problem by introducing a dedicated machinery based on model completions. While in the case of arithmetic variables the corresponding theories admit themselves quantifier elimination, this is not the case anymore for our data variables. However, we show that quantifier elimination for data variables that is available in the model completion of their theory can actually be safely employed in the context of backward reachability, retaining soundness and completeness when checking safety of RASs. This requires to *significantly modify* the original procedure and the original proofs.

In the general case, backward reachability is not guaranteed to terminate when checking safety of SASs and RASs. As a third contribution, we consequently isolate three notable classes of RASs for which backward reachability is guaranteed to terminate, in turn witnessing decidability of safety. The first class restricts the working memory to variables only, that is, focuses on SAS . The second class focuses on RAS operating under the restrictions imposed in Li et al. (2017): it requires

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

acyclicity of foreign keys and requires a sort of locality principle in the action guards, ensuring that different artifact tuples are not compared. Consequently, it reconstructs in our setting the essence of the decidability result exploited in Li et al. (2017) if one restricts the verification logic used there to safety properties only. In addition, our second class supports full-fledged bulk updates, which greatly increase the expressive power of dynamic systems (Schmitz and Schnoebelen 2013) and, in our setting, witness the incomparability of our results and those in Li et al. (2017). The third class is genuinely novel, and while it further restricts foreign keys to form a tree-shaped structure, it does not impose any restriction on the shape of updates, consequently supporting not only bulk updates, but also comparisons between artifact tuples. To prove termination of backward reachability for this class, we resort to techniques based on well-quasi-orders (in particular, a nontrivial application of Kruskal's Tree Theorem Kruskal 1960).

Our fourth and last contribution is to implement the new version of backward reachability required to handle the verification of RASs. We do so by extending the well-known MCMT (Model Checker Modulo Theories) model checker for array-based systems (Ghilardi and Ranise 2010b). The resulting version of MCMT (version 2.8) provides a fully operational counterpart to all the foundational results presented in the paper. Even though implementation and experimental evaluation are not the central goal of this paper, we also note that our model checker correctly handles the examples produced to test VERIFAS (Li et al. 2017), as well as additional examples that go beyond the verification capabilities of VERIFAS, and report some interesting case here. The performance of MCMT to conduct verification of these examples is very encouraging and indeed provides the first stepping stone toward effective, SMT-based verification techniques for artifact-centric systems.

All in all, the present contribution is deeply rooted in the long-standing tradition of the application of model theory in computer science, as witnessed by notable approaches like the one in Ghilardi (2004), Baader et al. (2006), Ghilardi et al. (2008b), Ghilardi and van Gool (2017), Nicolini et al. (2009a,b, 2010), Sofronie-Stokkermans (2008, 2016, 2018), Ghilardi and Gianola (2017, 2018). In particular, this paper applies these ideas in a genuinely novel mathematical context and shows how these techniques can be used for the first time to empower algorithmic techniques for the verification of infinite-state systems based on arrays in the style of Ghilardi et al. (2008a), Ghilardi and Ranise (2010a,b), Alberti et al. (2014a,b, 2017), Conchon et al. (2012, 2015, 2018a,b), Delzanno (2018), Cimatti et al. (2018), so as to make such techniques applicable to the timely, challenging settings of data-aware processes (Calvanese et al. 2019d). For an explicit linking between the use of model completeness in computer science and our application to verification, see in particular the survey (Calvanese et al. 2019b).

1.6 Plan of the paper and prerequisites

There are no specific prerequisites in model checking or in database theory to read this paper; we only assume some familiarity with some basic model theory (like that supplied in a one-semester course); some elementary notions will be revised mainly to fix notations. All definitions we introduce concerning read-only databases and relational artifact systems are explained with the help of a running example; however, we did not give the straightforward (and rather boring) details showing how basic operations on artifact relations (like insertion/deletions, resetting, etc.) can be modeled in our systems: there is a preliminary but more elementary exposition of the content of this paper available from the network (Calvanese et al. 2018b) which covers these low-level details and some more model-theoretic prerequisites.

The paper is organized as follows. After fixing preliminary notions in Section 2, in Section 3, we introduce our functional view of databases and discuss the related model-theoretic properties. In Section 4, we introduce a comprehensive framework for artifact-centric systems: SASs are studied in Subsection 4.1, whereas the most general framework of RASs is investigated in Subsection 4.2. Termination and decidability results are supplied in Section 5. Finally, first experiments are discussed in Section 6, and Section 7 concludes.

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

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* (respectively, *existential*) if it has the form $\forall \underline{x}(\phi(\underline{x}))$ (respectively, $\exists \underline{x}(\phi(\underline{x}))$), 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 $T \models \phi$ ("φ *is a logical consequence of* T") 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 $\mathcal{M}$. Thus, according to this definition, ϕ is T-satisfiable iff its *existential closure* is true in a model of T (notice that this convention might not be uniform in the literature). A Σ-theory T is *complete* iff for every Σ-sentence φ, either φ or $\neg\varphi$ is a logical consequence of 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 $\exists \underline{y}\, \phi(\underline{x}, \underline{y})$ (with ϕ a constraint, but, for the purposes of this definition, we may equivalently take ϕ to be quantifier-free), 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, that is, 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 *whenever* we talk about quantifier elimination, an *effective procedure* for eliminating quantifiers is given.

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 (we use $|\mathcal{A}|$ to denote the support of the structure $\mathcal{A}$). $\mathcal{A}$ can be expanded to a $\Sigma^{|\mathcal{A}|}$-structure $\mathcal{A}' := (\mathcal{A}, a)_{a\in|\mathcal{A}|}$ just interpreting the additional constants 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 $\mathcal{A}$, a Σ-formula $\phi(\underline{x})$, and elements $\underline{a}$ from $|\mathcal{A}|$, 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 $\mathcal{A}$ and $\mathcal{B}$ is any mapping $\mu : |\mathcal{A}| \longrightarrow |\mathcal{B}|$ among the support sets $|\mathcal{A}|$ of $\mathcal{A}$ and $|\mathcal{B}|$ of $\mathcal{B}$ satisfying the condition

$$\mathcal{A} \models \varphi \quad \Rightarrow \quad \mathcal{B} \models \varphi \tag{1}$$

for all $\Sigma^{|\mathcal{A}|}$-atoms φ (here – as above – $\mathcal{A}$ is regarded as a $\Sigma^{|\mathcal{A}|}$-structure, by interpreting each additional constant $a \in |\mathcal{A}|$ into itself and $\mathcal{B}$ is regarded as a $\Sigma^{|\mathcal{A}|}$-structure by interpreting each additional constant $a \in |\mathcal{A}|$ into $\mu(a)$). In case condition (1) holds for all $\Sigma^{|\mathcal{A}|}$-literals, the homomorphism μ is said to be an *embedding* (denoted by $\mu : \mathcal{A} \hookrightarrow \mathcal{B}$), and if it holds for all first-order

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

formulae, the embedding μ is said to be *elementary*. If $\mu : \mathcal{A} \longrightarrow \mathcal{B}$ is an embedding which is just the identity inclusion $|\mathcal{A}| \subseteq |\mathcal{B}|$, we say that $\mathcal{A}$ is a *substructure* of $\mathcal{B}$ or that $\mathcal{B}$ is an *extension* of $\mathcal{A}$. A Σ-structure $\mathcal{A}$ is said to be *generated by* a set X included in its support $|\mathcal{A}|$ iff there are no proper substructures of $\mathcal{A}$ including X.

2.1 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* (Chang and Keisler 1990), 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 $\nu_1 \circ \mu_1 = \nu_2 \circ \mu_2$*

$$\begin{tikzcd}
 & \mathcal{M} & \\
\mathcal{M}_1 \arrow[ur] & & \mathcal{M}_2 \arrow[ul] \\
 & \mathcal{M}_0 \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

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

2.2 Model completions

We recall a standard notion in model theory, namely, the notion of a *model completion* of a first-order theory (Chang and Keisler 1990) (we limit the definition to universal theories, because we shall use only this case):

Definition 2.2. *Let T be a universal Σ-theory and let $T^\star \supseteq T$ be a further Σ-theory; we say that $T^\star$ is a model completion of T iff:* (i) *every model of T can be embedded into a model of $T^\star$;* (ii) *for every model $\mathcal{M}$ of T, we have that $T^\star \cup \Delta_\Sigma(\mathcal{M})$ is a complete theory in the signature $\Sigma^{|\mathcal{M}|}$.*

Since T is universal, condition (ii) is equivalent to the fact that $T^\star$ *has quantifier elimination*; on the other hand, a standard argument (based on diagrams and compactness) shows that condition (i) is the same as asking that T and $T^\star$ have the same universal consequences. Thus, we have an equivalent definition (Ghilardi 2004) (to be used in the following):

Proposition 2.1. *Let T be a universal Σ-theory and let $T^\star \supseteq T$ be a further Σ-theory; $T^\star$ is a model completion of T iff: (i) every Σ-constraint satisfiable in a model of T is also satisfiable in a model of T^*; (ii) T^* has quantifier elimination.*

As stated before, we assume that a model completion has an *effective procedure* for eliminating quantifiers. We also recall that the model completion $T^\star$ of a theory T is unique, if it exists (see Chang and Keisler 1990 for these results and for examples). It is well known that a universal theory T which admits a model completion is also amalgamable (Chang and Keisler 1990).

Example 2.1. The theory of undirected graphs admits a model completion (and, hence, is amalgamable); this is the theory T whose signature Σ contains only a binary relational symbol R, and whose axioms are specified as follows:

$$T := \{\forall x \neg R(x,x), \forall x \forall y\, (R(x,y) \to R(y,x))\}.$$

Indeed, it is folklore that the model completion of T is the theory of the *Rado graph* (Rado 1964): a Rado (also called *random*) graph is a countably infinite graph in which, given any non-empty sets $X = \{x_0, ..., x_m\}$ and $Y = \{y_0, ..., y_n\}$ of nodes with $X \cap Y = \emptyset$, there is a node z (with $z \notin X \cup Y$) such that there is an edge between z and all elements of X and there is no edge between z and any element of Y. This theory is first-order definable (Fagin 1976).

2.3 Definable extensions and λ-notations

In the following, we specify transitions of artifact-centric systems using first-order formulae. To obtain a more compact representation, we make use of definable extensions as a means to introduce case-defined functions, abbreviating more complicated (still first-order) expressions. Let us fix a signature Σ and a Σ-theory T; a T-*partition* is a finite set $\kappa_1(\underline{x}), \ldots, \kappa_n(\underline{x})$ of quantifier-free formulae such that $T \models \forall \underline{x} \bigvee_{i=1}^{n} \kappa_i(\underline{x})$ and $T \models \bigwedge_{i \neq j} \forall \underline{x} \neg(\kappa_i(\underline{x}) \wedge \kappa_j(\underline{x}))$. Given such a T-partition $\kappa_1(\underline{x}), \ldots, \kappa_n(\underline{x})$ together with Σ-terms $t_1(\underline{x}), \ldots, t_n(\underline{x})$ (all of the same target sort), a *case-definable extension* is the Σ'-theory T', where $\Sigma' = \Sigma \cup \{F\}$, with F a "fresh" function symbol (i.e., $F \notin \Sigma$) , and $T' = T \cup \bigcup_{i=1}^{n} \{\forall \underline{x}\, (\kappa_i(\underline{x}) \to F(\underline{x}) = t_i(\underline{x}))\}$. Arity, source sorts, and target sort for F can be deduced from the context (considering that everything is well typed).

Intuitively, F represents a case-defined function, which can be reformulated using nested if-then-else expressions and can be written as

$$F(\underline{x}) \;:=\; \texttt{case of}\ \{\kappa_1(\underline{x}) : t_1; \cdots ; \kappa_n(\underline{x}) : t_n\}.$$

By abuse of notation, we shall identify T with any of its case-definable extensions T'. In fact, it is easy to produce from a Σ'-formula ϕ' a Σ-formula ϕ that is equivalent to ϕ' in all models of T': just remove (in the appropriate order) every occurrence $F(\underline{v})$ of the new symbol F in an atomic formula A, by replacing A with $\bigvee_{i=1}^{n} (\kappa_i(\underline{v}) \wedge A(t_i(\underline{v})))$.

We also exploit λ-abstractions (see, e.g., formula (13) below) for more "compact" representation of some complex expressions and always use them in atoms like $b = \lambda y.F(y, \underline{z})$ as abbreviations of $\forall y.\ b(y) = F(y, \underline{z})$ (where, typically, F is a symbol introduced in a case-defined extension as above). Thus, also our formulae containing lambda abstractions can be converted into plain first-order formulae.

3. Read-Only DB Schemas

We now provide a formal definition of (read-only) DB schemas by relying on an algebraic, functional characterization and derive some key model-theoretic properties instrumental to the technical treatment.

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.*

Next, we refer to a DB schema simply through its (DB) signature Σ and (DB) theory T. Given a DB signature Σ, we denote by Σ_{srt} the set of sorts and by Σ_{fun} the set of functions in Σ. Since Σ contains only unary function symbols and equality, all atomic Σ-formulae are of the form $t_1(v_1) = t_2(v_2)$, where t_1 and t_2 are possibly complex terms and v_1 and v_2 are either variables or constants.

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

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

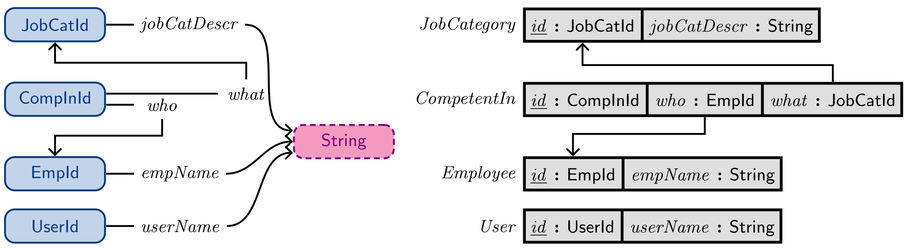

Figure 1. On the left: characteristic graph of the human resources DB signature from Example 3.1. On the right: relational view of the DB signature; each cell denotes an attribute with its type, underlined attributes denote primary keys, and directed edges capture foreign keys.

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 into 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, and clock values. We denote the set of id sorts in Σ by Σ_{ids}, and that of value sorts by Σ_{val}, hence $\Sigma_{srt} = \Sigma_{ids} \uplus \Sigma_{val}$.

We now consider extensional data.

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

Contrast this to arbitrary *models* of T, where no finiteness assumption is made. 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. More details on this will be given in Section 3.1.

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 in Σ. For example, if f has source S and target U, then $f^{\mathcal{M}}$ has domain $S^{\mathcal{M}}$ and range $U^{\mathcal{M}}$.

Example 3.1. The human resource (HR) branch of a company stores the following information inside a relational DB: *(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 close 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

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

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 artifact systems whose working memory is initially undefined, in the style of Deutsch et al. (2016), Li et al. (2017). To accommodate this, to specify an undefined value, we add to every sort S of Σ a constant $\texttt{undef}_S$ (written from now on, by abuse of notation, just as `undef`, used also to indicate a tuple). Then, for each function symbol f of Σ, we add the following axiom 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 3.1. In the artifact-centric model in the style of Deutsch et al. (2016), Li et al. (2017) that we intend to capture, the DB theory consists of Axioms (2) only. However, our technical results do not require this specific choice, and more general sufficient conditions will be discussed in Section 3.2.

Remark 3.2. If desired, we can freely extend DB schemas by adding arbitrary n-ary relation symbols to the signature Σ. For this purpose, we give the following definition.

Definition 3.3. *A* DB extended-schema *is a pair* $\langle \Sigma, T\rangle$*, where (i)* Σ *is a* DB extended-signature, *that is, a finite multi-sorted signature whose only symbols are equality, n-ary relations, unary functions, and constants and (ii) T is a* DB extended-theory*, that is, a set of universal* Σ*-sentences.*

For simplicity, even if our implementation takes into account also the case of “free” relations, that is, without key dependencies, we restrict our focus on DB schemas, which are sufficient to capture those constraints (as explained in the following subsection). The extension is straightforward and left to the reader. In fact, we can give in analogous the definitions of the characteristic graph $G(\Sigma)$ and of acyclicity for extended DB schemas. We notice that, in case Assumption 3.4 discussed below holds for DB extended-theories, all the results presented in Section 4 still hold even considering DB extended-schemas instead of DB schemas.

3.1 Relational view of DB schemas

We now clarify how 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 DB schemas equipped with single-attribute *primary keys* and *foreign keys* (plus a reformulation of constraint (2)). To technically explain the correspondence, 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 built starting from functions in Σ naturally induces different database dependencies in R_S. 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 in turn 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 $R_{S'}$.

Example 3.2. The diagram on the right in Figure 1 graphically depicts the relational view corresponding to the DB signature of Example 3.1. ◁

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

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 set

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

consisting of all (proper) identifiers assigned by $\mathcal{M}$ to id sorts, as well as all 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. From the definition of active domain above, exploiting Axioms (2), we get that the membership of a tuple $(x_0, \ldots, x_n)$ to a generic $n+1$-ary relation R_S with key dependencies (corresponding to an id sort S) can be expressed in our setting by using just unary function symbols and equality:

$$R_S(x_0, \ldots, x_n) \text{ iff } x_0 \neq \mathtt{undef} \wedge x_1 = f_1(x_0) \wedge \cdots \wedge x_n = f_n(x_0) \tag{3}$$

Hence, the representation of negated atoms is the one that directly follows from negating (3):

$$\neg R_S(x_0, \ldots, x_n) \text{ iff } x_0 = \mathtt{undef} \vee x_1 \neq f_1(x_0) \vee \cdots \vee x_n \neq f_n(x_0) \tag{4}$$

This relational interpretation of DB schemas exactly reconstructs the requirements posed by Deutsch et al. (2016), Li et al. (2017) on the schema of the *read-only* DB: *(i)* each relation schema has a single-attribute primary key; *(ii)* attributes are typed; *(iii)* attributes may be foreign keys referencing other relation schemas; and *(iv)* the primary keys of different relation schemas are pairwise disjoint.

We stress that all such requirements are natively captured in our functional definition of a DB signature and do not need to be formulated as axioms in the DB theory. The DB theory is used to express additional constraints, like that in Axiom (2). In the following subsection, we thoroughly discuss which properties must be respected by signatures and theories to guarantee that our verification machinery is well behaved.

One may wonder why we have not directly adopted a relational view for DB schemas. This will become clear during the technical development. We anticipate the main, intuitive reasons. First, our functional view allows us to guarantee that our framework remains well behaved even in the presence of key dependencies, since our DB theories *do* enjoy the crucial condition of Assumption 3.4 introduced below (i.e., that the DB theories admit a model completion), whereas relational structures with key constraints *do not*. Second, our functional view makes the dependencies among different types explicit. In fact, our notion of characteristic graph, which is readily computed from a DB signature, exactly reconstructs the central notion of foreign key graph used in Deutsch et al. (2016) toward the main decidability results.

3.2 Formal properties of DB schemas

The theory T from Definition 3.1 must satisfy few crucial requirements for our approach to work. In this section, we define such requirements and show that they are matched, for example, when we are concerned with an acyclic signature Σ and with key dependencies (i.e., the setting presented in Li et al. 2017). Actually, acyclicity is a stronger requirement than needed, which, however, simplifies our exposition.

3.2.1 Finite model property

We say that T has the *finite model property* (for constraint satisfiability) iff every constraint ϕ that is satisfiable in a model of T is satisfiable in a DB instance of T. It can be easily seen that this implies that ϕ is satisfiable also in a DB instance interpreting also value sorts into finite sets.

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 $k_\Sigma \cdot n$ terms involving n variables.

Proposition 3.1. *Let (Σ, T) be a DB schema (cf. Definition 3.1); T has the finite model property in case Σ is acyclic.*

Proof. If $T := \emptyset$, then congruence closure ensures that the finite model property holds and decides constraint satisfiability in polynomial time (Bradley and Manna 2007).

Otherwise, we reduce the argument to the Herbrand Theorem (recall that T is universal according to Definition 3.1). Indeed, suppose to have a finite set Φ of universal formulae and let $\phi(\underline{x})$ be the constraint we want to test for satisfiability. Replace the variables $\underline{x}$ with free constants $\underline{a}$. Herbrand Theorem states that $\Phi \cup \{\phi(\underline{a})\}$ has a model iff the set of ground $\Sigma^{\underline{a}}$-instances of $\Phi \cup \{\phi(\underline{a})\}$ has a model. These ground instances are finitely many by acyclicity, so we can apply congruence closure (as done in the case of the empty theory) to these ground instances. □

Remark 3.3. If T is finite, Proposition 3.1 ensures decidability of constraint satisfiability. In order to obtain a decision procedure, it is sufficient to instantiate the axioms of T and the axioms of equality (reflexivity, transitivity, symmetry, congruence) and to use a SAT-solver to decide constraint satisfiability. Alternatively, one can decide constraint satisfiability via congruence closure (Bradley and Manna 2007) and avoid instantiating the equality axioms.

Remark 3.4. Acyclity is a strong condition, often too strong. However, some condition must be imposed (otherwise, we have undecidability and then failure of finite model property, by reduction to word problem for finite presentations of monoids). In fact, the empty theory and the theory axiomatized by Axioms (2) both have the finite model property even without acyciclity assumptions.

The finite model property implies the decidability of the constraint satisfiability problem in case T is recursively axiomatized. Indeed, in this case, it is possible to enumerate unsatisfiable constraints via a logical calculus, and this enumeration can be interleaved with the enumeration of finite models, thus supplying a full decision procedure.

3.2.2 Model completion of DB theories

A DB theory T does not necessarily have quantifier elimination; it is, however, often possible to strengthen T in a conservative way (with respect to constraint satisfiability) and get quantifier elimination. In order to do that, we study the model completion of T, when it exists, and we will show that model completion turns out to be quite effective to attack the verification of dynamic systems operating over relational databases.

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

Lemma 3.1. *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*

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

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

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

Then, T has a model completion T^* axiomatized by the infinitely many sentences

$$\forall \underline{y} \, (\psi(\underline{y}) \to \exists x \, \phi(x, \underline{y})) \, . \tag{5}$$

Proof. From (i) and (5), 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) and (5).

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 pair $(\underline{a}, \exists x \phi(x, \underline{a}))$ such that $\mathcal{M} \models \psi(\underline{a})$ (where ψ is related to ϕ as in (i)). By Zermelo's Theorem, the set of all pairs $\{(\underline{a}, \exists x \phi(x, \underline{a}))\}$ 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 $i \in I$, $\mathcal{M}_i$ as an extension of $\bigcup_{j<i} \mathcal{M}_j$ such that $\mathcal{M}_i \models \exists x \phi_i(x, \underline{a}_i)$, which exists for (ii) 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, 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 $\mathcal{M}_0 := \mathcal{M} \subseteq \mathcal{M}^1 \subseteq \mathcal{M}^2$ 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 (5). The last fact can be shown using the following finiteness argument.

Fix ϕ, ψ as in (5). 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 $\psi(\underline{a}')$ 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 $(\underline{a}', \exists x \phi(x, \underline{a}'))$ appears in the enumeration given by the well-ordered set of pairs $\{(\underline{a}_i, \exists x \phi_i(x, \underline{a}_i))\}_{i \in I}$ such that $\mathcal{M}^k \models \psi_i(\underline{a}_i)$. Hence, by construction, we have that $\mathcal{M}^k_i \models \exists x \phi(x, \underline{a}')$ for some i. In conclusion, since the existential formulae are preserved passing to extensions, we obtain $\mathcal{M}' \models \exists x \phi(x, \underline{a}')$, as wanted. □

Proposition 3.2. *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 Wheeler (1976). We preliminarily 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 validity of the condition (ii) of Lemma 3.1: this fact (together with condition (i)) yields that T has a model completion which is axiomatized by the infinitely many sentences (5).

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be 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 $\mathcal{M}_2$, that is, $|\mathcal{M}| := |\mathcal{M}_1| \cup |\mathcal{M}_2|$. $\mathcal{M}$ 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 $m_i \in |\mathcal{M}_i| (i = 1, 2)$, $f^{\mathcal{M}}(m_i) := f^{\mathcal{M}_i}(m_i)$, that is, the interpretation of f in

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

$\mathcal{M}$ is 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, that is, $\phi := \forall x(A_1(x) \wedge ... \wedge A_n(x) \rightarrow B_1(x) \vee ... \vee B_m(x))$, 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 a 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, that is, $\mathcal{M} \models T$ 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 3.1 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 3.1 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 $\mathcal{M}[\underline{a}]$ such that $\mathcal{N} \models \exists x \phi(x, \underline{a})$, it is sufficient to prove (by the Robinson Diagram Lemma) that the $\Sigma^{|\mathcal{M}[\underline{a}]| \cup \{e\}}$-theory $\Delta(\mathcal{M}[\underline{a}]) \cup \{\phi(e, \underline{a})\}$ 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 $\Delta(\mathcal{M}[\underline{a}])$ (remember that $\Delta(\mathcal{M}[\underline{a}])$ is a finite set of literals since $\mathcal{M}[\underline{a}]$ 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. □

Remark 3.5. The proof of Proposition 3.2 gives an algorithm for quantifier elimination in the model completion. The algorithm works as follows (see the formula (5)): to eliminate the quantifier $\exists x$ from $\exists x\, \phi(x, \underline{y})$ take the conjunction of the clauses $\chi(\underline{y})$ implied by $\phi(x, \underline{y})$. This algorithm is far from optimal from two points of view. First, contrary to what happens in linear arithmetics, the quantifier elimination needed to prove Proposition 3.2 has a much better behavior (from the complexity point of view) if obtained via a suitable version of the Knuth–Bendix procedure (Baader and Nipkow 1998) or of the Superposition Calculus (Nieuwenhuis and Rubio 2001). Since these aspects concerning quantifier elimination are rather delicate, we started studying them in a dedicated paper (Calvanese et al. 2019c) and in its extended version (Calvanese et al. 2018a) (our MCMT implementation, however, already partially takes into account such development), where, by using a constrained version of Superposition, we show that in the case of free unary functions and free relations the complexity has a quadratic bound even without assuming acyclicity.

Secondly, it is worth noting that the algorithm presented in Proposition 3.2 uses the acyclicity assumption, whereas such assumption, as just noticed, is in general not needed for Proposition 3.2 to hold: for instance, when $T := \emptyset$ or when T contains only Axiom (2), a model completion can be proved to exist, even if Σ is not acyclic, by using the constrained version of Superposition Calculus studied in Calvanese et al. (2018a, 2019c).

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

Remark 3.6. Proposition 3.2 holds also for DB extended-schemas, in case the universal one-variable formulae do not involve the relation symbols (so, the relations are "free"): as explained in Calvanese et al. (2018a), our implementation of the quantifier elimination algorithm takes into account also this case. More generally, the model completion exists whenever we consider an acyclic DB extended-schema with a DB extended-theory T that enjoys the amalgamation property.

Hereafter, we make the following assumption:

Assumption 3.4. *The DB theories we consider have decidable constraint satisfiability problem, finite model property, and admit a model completion.*

This assumption is matched, for instance, in the following three cases: *(i)* when T is empty; *(ii)* when T is axiomatized by Axioms (2); *(iii)* when Σ is acyclic and T is axiomatized by universal one-variable formulae (such as Axioms (2)).

Hence, the artifact-centric model in the style of Deutsch et al. (2016), Li et al. (2017) that we intend to capture *matches* Assumption 3.4.

Remark 3.7. Notice that the DB extended-schemas obtained by adding "free" relations to the DB schemas of *(i)*, *(ii)*, *(iii)* above match Assumption 3.4.

4. Artifact-Centric Systems

We are now in the position to define our formal models of SASs and RASs and to study parameterized safety problems over SASs and RASs. Since RASs are formalized in the spirit of array-based systems, we start by recalling the intuition behind them.

In general terms, an array-based system is described using a multi-sorted theory that contains two types of sorts, one accounting for the indexes of arrays and the other for the elements stored therein. The system variables changing over time are both individual first-order variables for data and second-order variables for arrays. The latter are referred to using second-order function variables, whose interpretation in a state is that of a total function mapping indexes to elements (so that applying the function to an index denotes the classical *read* operation for arrays). The definition of an array-based system with array and data variables a always requires: a formula $\iota(a)$ describing the *initial configuration* of the variables a and a formula $\tau(a, a')$ describing a *transition* that transforms the content of the variables from a to a'. In such a setting, verifying whether the system can reach unsafe configurations described by a formula $\upsilon(a)$ amounts to check whether the formula $\iota(a_0) \wedge \tau(a_0, a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge \upsilon(a_n)$ is satisfiable for some n.

Next, we make these ideas formally precise by grounding array-based systems in the artifact-centric setting. We start considering the case where we only have individual variables for data and then we pass to the complete framework where we also have second-order variables formalizing artifact relations (that is, relations which are mutable during system evolution).

4.1 Simple artifact systems

The SAS formal model. In this subsection, we consider systems manipulating only individual variables and reading data from a given DB instance. In order to introduce verification problems in a symbolic setting, one first has to specify which formulae are used to represent sets of states, the system initializations, and system evolution. Given a DB schema $\langle \Sigma, T\rangle$ and a tuple $\underline{x} = x_1, \ldots, x_n$ of variables, we introduce the following classes of Σ-formulae:

- a *state formula* is a quantifier-free Σ-formula $\phi(\underline{x})$;
- an *initial formula* is a conjunction of equalities of the form $\bigwedge_{i=1}^{n} x_i = c_i$, where each c_i is a constant (typically, c_i is an `undef` constant mentioned in Section 3);
- a *transition formula* $\hat{\tau}$ is an existential formula

$$\exists \underline{y} \left(G(\underline{x}, \underline{y}) \wedge \textstyle\bigwedge_{i=1}^{n} x'_i = F_i(\underline{x}, \underline{y}) \right) \tag{6}$$

where $\underline{x}'$ are renamed copies of $\underline{x}$, G is quantifier-free, and $F_1, \ldots, F_n$ are case-defined functions. We call G the *guard* and F_i the *updates* of Formula (6).

Definition 4.1. *A* Simple Artifact System *(SAS) has the form*

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

where (i) $\langle \Sigma, T \rangle$ is a DB schema, (ii) $\underline{x} = x_1, \ldots, x_n$ are variables (called artifact variables*), (iii) ι is an initial formula, and (iv) τ is a disjunction of transition formulae of the type* (6)*.*

We notice that a formula $\hat{\tau}$ of the kind (6) is a *single* transition formula, where τ from Definition 4.1 is a disjunction of formulae of the kind (6); hence, such τ symbolically represents the union of all the possible transitions of the system.

Example 4.1. We consider a SAS working over the DB schema of Example 3.1. It captures a global, single-instance artifact tracking the main, overall phases of a hiring process. The job hiring artifact employs a dedicated *pState* variable to store the current process state. Initially, hiring is disabled, which is captured by setting the *pState* variable to `undef`. A transition of the process from disabled to *enabled* may occur provided that the read-only HR DB contains at least one registered user (who, in turn, may decide to apply for a job). Technically, we introduce a dedicated artifact variable *uId* initialized to `undef` and used to load the identifier of such a registered user, if (s)he exists. The enabling action is then captured by the following transition formula:

$$\exists y : \texttt{UserId} \begin{pmatrix} \mathit{pState} = \texttt{undef} \wedge y \neq \texttt{undef} \\ \wedge\, \mathit{pState}' = \texttt{enabled} \wedge \mathit{uId}' = y \end{pmatrix}$$

The existential quantified variable $y : \texttt{UserId}$ is a data variable pointing to the read-only DB and is used to represent an external user input. Notice in particular how the existence of a user is checked using the typed variable y, checking that it is not `undef` and correspondingly assigning it to *uId*. ◁

Parameterized safety via backward reachability for SAS . A *safety* formula for a SAS $\mathcal{S}$ is a state formula $\upsilon(\underline{x})$ describing undesired states of $\mathcal{S}$. We say that $\mathcal{S}$ is *safe with respect to* υ if intuitively the system has no finite run leading from ι to υ. Formally, there is no DB instance $\mathcal{M}$ of $\langle \Sigma, T \rangle$, no $k \geq 0$, and no assignment in $\mathcal{M}$ to the variables $\underline{x}^0, \ldots, \underline{x}^k$ such that the formula

$$\iota(\underline{x}^0) \wedge \tau(\underline{x}^0, \underline{x}^1) \wedge \cdots \wedge \tau(\underline{x}^{k-1}, \underline{x}^k) \wedge \upsilon(\underline{x}^k) \tag{7}$$

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

Example 4.2. We provide an example of a safety formula for Example 4.1. Consider the *unsafe* configuration where the process is enabled but the identifier of the registered user loaded into *uId* is `undef`. Formally, this can be represented by the following state formula:

$$\mathit{pState} = \texttt{enabled} \wedge \mathit{uId} = \texttt{undef}$$

Notice that the following formula

$$\exists y \left(\mathit{pState} = \texttt{enabled} \wedge y \neq \texttt{undef} \wedge \mathit{uId} = y \right)$$

is *not* a safety formula, because of the existential quantified data variable y, but it is equivalent to

$$\mathit{pState} = \texttt{enabled} \wedge \mathit{uId} \neq \texttt{undef}$$

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

Algorithm 1: Backward reachability algorithm

```
Function BReach($\upsilon$)
1   $\phi \longleftarrow \upsilon$; $B \longleftarrow \bot$;
2   while $\phi \wedge \neg B$ is $T$-satisfiable do
3       if $\iota \wedge \phi$ is $T$-satisfiable then
            return unsafe
4       $B \longleftarrow \phi \vee B$;
5       $\phi \longleftarrow Pre(\tau, \phi)$;
6       $\phi \longleftarrow$ QE($T^*, \phi$);
    return safe;
```

which is a safety formula. We will see in Lemma 4.2 that this equivalence (in some sense) holds in the general case of RASs (which SASs are a specific case of).

Algorithm 1 describes the *modified version of the backward reachability algorithm* (simply called *backward reachability* or *backward search* in the following) for handling the safety problem for $\mathcal{S}$ (the original version of the backward reachability algorithm can be found in Ghilardi et al. 2008a; Ghilardi and Ranise 2010a). An integral part of Algorithm 1 is to compute preimages. For that purpose, we define for any $\phi_1(\underline{z}, \underline{z}')$ and $\phi_2(\underline{z})$ (where $\underline{z}'$ are renamed copies of $\underline{z}$), $Pre(\phi_1, \phi_2)$ as the formula $\exists \underline{z}'(\phi_1(\underline{z}, \underline{z}') \wedge \phi_2(\underline{z}'))$. The *preimage* of the set of states described by a state formula $\phi(\underline{x})$ is the set of states described by $Pre(\tau, \phi)$. Notice that, when $\tau = \bigvee \hat{\tau}$, then $Pre(\tau, \phi) = \bigvee Pre(\hat{\tau}, \phi)$. The modified algorithm presents a new subprocedure QE(T^*, ϕ) in Line 6 for computing quantifier elimination in the model completion T^*. The subprocedure QE(T^*, ϕ) applies the quantifier elimination algorithm of T^* to the existential formula ϕ. Algorithm 1 computes iterated preimages of υ 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 can be discharged via proof obligations to be handled by SMT solvers. The fixpoint is reached when the test in Line 2 returns *unsat*, which means that the preimage of the set of the current states is included in the set of states reached by the backward search so far.

In the following, *partial correctness* means that, when the algorithm terminates, it gives a correct answer, whereas *effectiveness* means that all subprocedures in the algorithm can be effectively executed. We state now the main result of this subsection:

Theorem 4.2. *Let $\langle \Sigma, T \rangle$ be a DB schema. Then, for every SAS $\mathcal{S} = \langle \Sigma, T, \underline{x}, \iota, \tau \rangle$, the following hold: (1) backward search is effective and partially correct for solving safety problems for $\mathcal{S}$; (2) if Σ is acyclic, backward search terminates and decides safety problems for $\mathcal{S}$ in* PSPACE *in the combined size of $\underline{x}$, ι, and τ.*

Proof. **Part (1).** Recall formula (7)

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

By definition, $\mathcal{S}$ is unsafe iff for some n, the formula (7) is satisfiable in a DB instance of $\langle \Sigma, T \rangle$. Thanks to Assumption 3.4, T has the finite model property, and consequently, as (7) is an existential Σ-formula, $\mathcal{S}$ is unsafe iff for some n, formula (7) is satisfiable in a model of T; furthermore, again by Assumption 3.4, $\mathcal{S}$ is unsafe iff for some n, formula (7) is satisfiable in a model of T^*. Thus, we shall concentrate on satisfiability in models of T^* in order to prove the theorem.

Let us call B_n (respectively, ϕ_n), with $n \geq 0$, the status of the variable B (respectively, ϕ) after n executions in Line 4 (respectively, Line 6) of Algorithm 1 ($n = 0$ corresponds to the status of the variables in Line 1). Notice that we have

$$T^* \models \phi_{j+1} \leftrightarrow Pre(\tau, \phi_j) \tag{8}$$

for all j and that

$$T \models B_n \leftrightarrow \bigvee_{0 \leq j < n} \phi_j \tag{9}$$

is an invariant of the algorithm.

Since we are considering satisfiability in models of T^*, we can apply quantifier elimination and so the satisfiability of (7) is equivalent to the satisfiability of $\iota \wedge \phi_n$: this is a quantifier-free formula (because of line 6 of Algorithm 1), whose satisfiability (w.r.t. T or equivalently w.r.t. T^*)[2] is decidable by Assumption 1, so if Algorithm 1 terminates with an unsafe outcome, then $\mathcal{S}$ is really unsafe.

Now consider the satisfiability test in Line 2. This is again a satisfiability test for a quantifier-free formula, thus it is decidable. In case of a safe outcome, we have that $T \models \phi_n \rightarrow B_n$; we claim that, if we continued executing the loop of Algorithm 1, we would nevertheless get that:

$$T^* \models B_m \leftrightarrow B_n \tag{10}$$

for all $m \geq n$. We justify Claim (10) below.

From $T \models \phi_n \rightarrow B_n$, taking into consideration that $T^* \supseteq T$ and that Formula (8) holds, we get $T^* \models \phi_{n+1} \rightarrow Pre(\tau, B_n)$. Since Pre commutes with disjunctions (i.e., $Pre(\tau, \bigvee_j \phi_j)$ is logically equivalent to $\bigvee_j Pre(\tau, \phi_j)$), we also have $T^* \models Pre(\tau, B_n) \leftrightarrow \bigvee_{1 \leq j \leq n} \phi_j$ by the Invariant (9) and by Formula (8) again. By using the entailment $T \models \phi_n \rightarrow B_n$ once more, we get $T^* \models \phi_{n+1} \rightarrow B_n$ and also that $T^* \models B_{n+1} \leftrightarrow B_n$; thus, we finally obtain that $T^* \models \phi_{n+1} \rightarrow B_{n+1}$. Since $\phi_{n+1} \rightarrow B_{n+1}$ is quantifier-free, $T^* \models \phi_{n+1} \rightarrow B_{n+1}$ implies $T \models \phi_{n+1} \rightarrow B_{n+1}$. This argument can be repeated for all $m \geq n$, obtaining that $T^* \models B_m \leftrightarrow B_n$ for all $m \geq n$, that is, Claim (10).

This would entail that $\iota \wedge \phi_m$ is always unsatisfiable (because of (9) and because $\iota \wedge \phi_j$ was unsatisfiable for all $j < n$), which is the same (as remarked above) as saying that all formulae (7) are unsatisfiable. Thus $\mathcal{S}$ is safe.

Part (2). In case Σ is acyclic, there are only finitely many quantifier-free formulae (in which the finite set of variables $\underline{x}$ occur), so it is evident that the algorithm must terminate: because of (9), the unsatisfiability test of Line 2 must eventually succeed, if the unsatisfiability test of Line 3 never does so.

Concerning complexity, we need to modify Algorithm 1 (we make it nondeterministic and use Savitch's Theorem saying that PSPACE = NPSPACE).

Since Σ is acyclic, there are only finitely many terms involving a single variable: let this number be k_Σ (we consider T, Σ and hence k_Σ constant for our problems). Then, since all function symbols are unary, it is clear that we have at most $2^{O(n^2)}$ conjunctions of sets of literals involving at most n variables and that if the system is unsafe, unsafety can be detected with a run whose length is at most $2^{O(n^2)}$. Thus, we introduce a counter to be incremented during the main loop (lines 2–6) of Algorithm 1. The fixpoint test in line 2 is removed, and loop is executed only until the maximum length of an unsafe run is not exceeded (notice that an exponential counter requires polynomial space).

Inside the loop, line 4 is removed (we do not need anymore the variable B) and line 6 is modified as follows. We replace line 6 of the algorithm by

$$6'. \quad \phi \longleftarrow \alpha(\underline{x});$$

where α is a nondeterministically chosen conjunction of literals implying $\mathsf{QE}(T^*, \phi)$. Notice that to check the latter, there is no need to compute $\mathsf{QE}(T^*, \phi)$: recalling the proof of Proposition 3.2

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

and Remark 3.5, it is sufficient to check that $T \models \alpha \to C$ holds for every clause $C(\underline{x})$ such that $T \models \phi \to C$.

The algorithm is now in PSPACE, because all the satisfiability tests we need are, as a consequence of the proof of Proposition 3.1, in NP (Nondeterministic Polynomial Time): all such tests are reducible to T-satisfiability tests for quantifier-free Σ-formulae involving the variables $\underline{x}$ and the additional (skolemized) quantified variables occurring in the transitions.[3] In fact, all these satisfiability tests are applied to formulae whose length is polynomial in the size of $\underline{x}$, of ι and of τ. □

The proof of Theorem 4.2 shows that, whenever Σ is not acyclic, backward search is still a semi-decision procedure: if the system is unsafe, backward search always terminates and discovers it; if the system is safe, the procedure can diverge (but it is still correct).

Remark 4.1. We remark that Theorem 4.2 holds also for DB extended-schemas (so, even adding "free relations" to the DB signatures). Moreover, notice that it can be shown that every existential formula $\phi(\underline{x}, \underline{x}')$ can be turned into the form of Formula (6). Furthermore, we highlight that the proof of the decidability result of Theorem 4.2 requires that the considered background theory T: *(i)* admits a model completion; *(ii)* is *locally finite*, that is, up to T-equivalence, there are only finitely many atoms involving a fixed finite number of variables (this condition is implied by acyclicity); *(iii)* is universal; and *(iv)* enjoys decidability of constraint satisfiability. Conditions *(iii)* and *(iv)* imply that one can decide whether a finite structure is a model of T. If *(ii)* and *(iii)* hold, it is well known that *(i)* is equivalent to amalgamation (Lipparini 1982; Wheeler 1976). Moreover, *(ii)* alone always holds for relational signatures and *(iii)* is equivalent to T being closed under substructures (this is a standard preservation theorem in model theory Chang and Keisler 1990). It follows that *arbitrary relational signatures* (or *locally finite theories* in general, even allowing n-ary relation and n-ary function symbols) require only amalgamability and closure under substructures. Thanks to these observations, Theorem 4.2 is reminiscent of an analogous result in Bojańczyk et al. (2013), that is, Theorem 5, the crucial hypotheses of which are exactly amalgamability and closure under substructures, although the setting in that paper is different (there, key dependencies are not discussed, whereas we are interested only in DB (extended-) theories).

In our first-order setting, we can perform verification in a *purely symbolic* way, by using (semi-)decision procedures provided by SMT-solvers, even when local finiteness fails. As mentioned before, local finiteness is guaranteed in the relational context, but it does not hold anymore when *arithmetic operations* are introduced. Note that the theory of a single uninterpreted binary relation (i.e., the theory of directed graphs) has a model completion, whereas it can be easily seen that the theory of one binary relation which is a partial function *does not* (since it is *not* amalgamable). If primary key dependencies are formalized using partial functions, model completability is compromized. So, the second distinctive feature of our setting naturally follows from this observation: thanks to our *many-sorted functional representation* of DB schemas (with keys), the amalgamation property, required by Theorem 4.2, holds, witnessing that our framework remains well behaved even in the presence of key dependencies.

4.2 Relational artifact systems

The RAS formal model. Following the tradition of artifact-centric systems (Deutsch et al. 2009, 2016), a RAS consists of a read-only DB, a read–write working memory for artifacts, and a finite set of actions (also called services) that inspect the relational database and the working memory and determine the new configuration of the working memory. In an RAS , the working memory consists of *individual* and *higher order* variables. These higher order variables (usually called *arrays*) are supposed to model evolving relations, so-called *artifact relations* in Deutsch et al. (2016), Li et al. (2017). The idea is to treat artifact relations in a uniform way as we did for the read-only

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

DB: we need extra sort symbols (recall that each sort symbol corresponds to a DB relation symbol) and extra unary function symbols, the latter being treated as second-order variables.

Given a DB schema Σ, an *artifact extension* of Σ is a signature Σ_{ext} obtained from Σ by adding to it some extra sort symbols.[4] These new sorts (usually indicated with $E, E_1, E_2 \ldots$) are called *artifact sorts* (or *artifact relations* by some abuse of terminology), whereas the old sorts from Σ are called *basic sorts*. In RAS , artifacts and basic sorts correspond, respectively, to the index and the elements sorts mentioned in the literature on array-based systems. Below, given $\langle \Sigma, T\rangle$ and an artifact extension Σ_{ext} of Σ, when we speak of a Σ_{ext}-model of T, a DB instance of $\langle \Sigma_{ext}, T\rangle$, or a Σ_{ext}-model of T^*, we mean a Σ_{ext}-structure $\mathcal{M}$ whose reduct to Σ, respectively, is a model of T, a DB instance of $\langle \Sigma, T\rangle$, or a model of T^*.

An *artifact setting* over Σ_{ext} is a pair $(\underline{x}, \underline{a})$ given by a finite set $\underline{x}$ of individual variables and a finite set $\underline{a}$ of unary function variables: *the latter are required to have an artifact sort as source sort and a basic sort as target sort*. Variables in $\underline{x}$ are called (as before) *artifact variables*, and variables in $\underline{a}$ *artifact components*. Given a DB instance $\mathcal{M}$ of Σ_{ext}, an *assignment* to an artifact setting $(\underline{x}, \underline{a})$ over Σ_{ext} is a map α assigning to every artifact variable $x_i \in \underline{x}$ of sort S_i an element $x^\alpha \in S_i^{\mathcal{M}}$ and to every artifact component $a_j : E_j \longrightarrow U_j$ (with $a_j \in \underline{a}$) a set-theoretic function $a_j^\alpha : E_j^{\mathcal{M}} \longrightarrow U_j^{\mathcal{M}}$.

We can view an assignment to an artifact setting $(\underline{x}, \underline{a})$ as a DB instance *extending* the DB instance $\mathcal{M}$ as follows. Let all the artifact components in $(\underline{x}, \underline{a})$ having source E be $a_{i_1} : E \longrightarrow S_1, \cdots, a_{i_n} : E \longrightarrow S_n$. Viewed as a relation in the artifact assignment $(\mathcal{M}, \alpha)$, the artifact relation E "consists" of the set of tuples

$$\left\{\left\langle e, a_{i_1}^\alpha(e), \ldots, a_{i_n}^\alpha(e)\right\rangle \mid e \in E^{\mathcal{M}}\right\}.$$

Thus, each element of E is formed by an "entry" $e \in E^{\mathcal{M}}$ (uniquely identifying the tuple) and by "data" $\underline{a}_i^\alpha(e)$ taken from the read-only DB $\mathcal{M}$. When the system evolves, the set $E^{\mathcal{M}}$ of entries remains fixed, whereas the components $\underline{a}_i^\alpha(e)$ may change: typically, we initially have $\underline{a}_i^\alpha(e) = \mathtt{undef}$, but these values are changed when some defined values are inserted into the relation modeled by E; the values are then repeatedly modified (and possibly also reset to undef, if the tuple is removed and e is reset to point to undefined values). In accordance with MCMT conventions, we denote the application of an artifact component a to a term (i.e., constant or variable) v also as $a[v]$ (standard notation for arrays), instead of $a(v)$.

To introduce RASs, we discuss the kind of formulae we use. In such formulae, we use notations like $\phi(\underline{z}, \underline{b})$ to mean that ϕ is a formula whose free individual variables are among the $\underline{z}$ and whose free unary function variables are among the $\underline{b}$.

Let $(\underline{x}, \underline{a})$ be an artifact setting over Σ_{ext}, where $\underline{x} = x_1, \ldots, x_n$ are the artifact variables and $\underline{a} = a_1, \ldots, a_m$ are the artifact components (their source and target sorts are left implicitly specified). We list the kind of formulae we shall use:

- An *initial formula* is a formula $\iota(\underline{x}, \underline{a})$ of the form

$$\left(\bigwedge_{i=1}^n x_i = c_i\right) \wedge \left(\bigwedge_{j=1}^m a_j = \lambda y.d_j\right) \tag{11}$$

  where c_i and d_j are constants from Σ (typically, c_i and d_j are undef). Recall that $a_j = \lambda y.d_j$ abbreviates $\forall y\, a_j(y) = d_j$.
- A *state formula* has the form

$$\exists \underline{e}\, \phi(\underline{e}, \underline{x}, \underline{a}) \tag{12}$$

  where ϕ is quantifier-free and the $\underline{e}$ are individual variables of artifact sorts.

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

- A *transition formula* $\hat{\tau}$ has the form

$$\exists \underline{e} \left(\begin{array}{l} \gamma(\underline{e}, \underline{x}, \underline{a}) \wedge \bigwedge_i x'_i = F_i(\underline{e}, \underline{x}, \underline{a}) \\ \qquad\qquad \wedge \bigwedge_j a'_j = \lambda y. G_j(y, \underline{e}, \underline{x}, \underline{a}) \end{array} \right) \tag{13}$$

  where $\underline{e}$ are individual variables (of *both* basic and artifact sorts), γ (the "guard") is quantifier-free, $\underline{x}'$ and $\underline{a}'$ are the renamed copies of $\underline{x}$ and $\underline{a}$, and the F_i, G_j (the "updates") are case-defined functions.

Note that transition formulae as above can express, e.g., *(i)* insertion (with/without duplicates) of a tuple in an artifact relation, *(ii)* removal of a tuple from an artifact relation, *(iii)* transfer of a tuple from an artifact relation to artifact variables (and vice versa), and *(iv)* removal/modification of *all* the tuples satisfying a certain condition from an artifact relation. All the above operations can also be constrained; the formalization of the above operations in the formalism of our transition is straightforward (the reader can see all the details in Appendix F from Calvanese et al. 2018b).

Definition 4.3. *A* Relational Artifact System *(RAS) has the form*

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

where (i) $\langle \Sigma, T \rangle$ *is a (read-only) DB schema, (ii)* Σ_{ext} *is an artifact extension of* Σ, *(iii)* $(\underline{x}, \underline{a})$ *is an artifact setting over* Σ_{ext}, *(iv)* ι *is an initial formula, and (v)* τ *is a disjunction of transition formulae of the type* (13).

Notice that SASs are a particular class of RASs where the working memory consists *only* of artifact variables (without artifact relations).

Example 4.3. We transform the SAS of Example 4.1 into an RAS $\mathcal{S}_{hr}$ containing a multi-instance artifact accounting for the evolution of *job applications*. Each job category may receive multiple applications from registered users. Such applications are then evaluated, finally deciding which are accepted and which are rejected. The example is inspired by the job hiring process presented in Silver (2011) to show the intrinsic difficulties of capturing real-life processes with many-to-many interacting business entities using conventional process modeling notations (such as Business Process Model and Notation (BPMN)). An extended version of this example, capturing the co-evolution of multiple instances of two different artifacts, is presented in Appendix A.1 of Calvanese et al. (2018b).

As for the read-only DB, $\mathcal{S}_{hr}$ works over the DB schema of Example 3.1, extended with a further value sort Score used to score job applications. Score contains 102 values in the range [−1, 100], where −1 denotes the non-eligibility of the application, and a score from 0 to 100 indicates the actual one assigned after evaluating the application. For the sake of readability, we use usual predicates $<$, $>$, and $=$ to compare variables of type Score. This is syntactic sugar and does not require to introduce *rigid* predicates (i.e., first-order logic relational symbols that have a predetermined interpretation in an assigned quantified domain) in our framework: in the following, we do not need to introduce a new relational symbol $>$ that has to be interpreted in the model of natural numbers as the standard numerical relation $>$. Indeed, since in our case $>$ is applied to a fixed finite number of elements (i.e., the natural numbers in [−1, 100]), a literal of the form $s > 80$ can be substituted by a disjunction of a finite number of equalities (i.e., $s > 80$ iff ($s = 81$ or $s = 82$ or...or $s = 100$)).

As for the working memory, $\mathcal{S}_{hr}$ consists of two artifacts: the single-instance *job hiring* artifact tracking the three main phases of the overall process (and described in Example 4.1) and a multi-instance artifact accounting for the evolution of *user applications*. To model applications,

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

we take the DB signature Σ_{hr} of the read-only DB of HRs and enrich it with an artifact extension containing an artifact sort $\mathsf{appIndex}$ used to *index* (i.e., *"internally" identify*) job applications. The management of job applications is then modeled by an artifact setting with: *(i)* artifact components with domain $\mathsf{appIndex}$ capturing the artifact relation that stores the different job applications and *(ii)* additional individual variables as a temporary memory to manipulate the artifact relation. Specifically, each application consists of a job category, the identifier of the applicant user and that of an HR employee responsible for the application, the application score, and the final result (indicating whether the application is among the winners or the losers for the job offer). These information slots are encapsulated into dedicated artifact components, that is, function variables with domain $\mathsf{appIndex}$ that collectively realize the application artifact relation:

$$\begin{array}{lcl} appJobCat & : & \mathsf{appIndex} \longrightarrow \mathsf{JobCatId} \\ applicant & : & \mathsf{appIndex} \longrightarrow \mathtt{UserId} \\ appResp & : & \mathsf{appIndex} \longrightarrow \mathsf{EmpId} \\ appScore & : & \mathsf{appIndex} \longrightarrow \mathsf{Score} \\ appResult & : & \mathsf{appIndex} \longrightarrow \mathsf{String} \end{array}$$

We now discuss the relevant transitions for inserting and evaluating job applications. When writing transition formulae, we make the following assumption: if an artifact variable/component is not mentioned at all, it is meant that it is updated identically; otherwise, the relevant update function will specify how it is updated. Notice that, as mentioned also in the introduction, nondeterministic updates can be formalized using existentially quantified variables in the transition. The insertion of an application into the system can be executed when the hiring process is enabled (cf. Example 4.1) and consists of two consecutive steps. To indicate when a step can be applied, also ensuring that the insertion of an application is not interrupted by the insertion of another one, we manipulate a string artifact variable *aState*. The first step is executable when *aState* is `undef` and aims at loading the application data into dedicated artifact variables through the following simultaneous effects: *(i)* the identifier of the user who wants to submit the application and that of the targeted job category are selected and, respectively, stored into variables *uId* and *jId*; *(ii)* the identifier of an HR employee who becomes responsible for the application is selected and stored into variable *eId*, with the requirement that such an employee must be competent in the job category targeted by the application; *(iii) aState* evolves into state `received`. Formally:

$$\exists u{:}\mathtt{UserId}, j{:}\mathsf{JobCatId}, e{:}\mathsf{EmpId}, c{:}\mathsf{CompInId} \\ \left(\begin{array}{l} pState = \mathtt{enabled} \wedge aState = \mathtt{undef} \\ \wedge\, u \neq \mathtt{undef} \wedge j \neq \mathtt{undef} \wedge e \neq \mathtt{undef} \wedge c \neq \mathtt{undef} \\ \wedge\, who(c) = e \wedge what(c) = j \\ \wedge\, pState' = \mathtt{enabled} \wedge aState' = \mathtt{received} \\ \wedge\, uId' = u \wedge = j \wedge eId' = e \wedge cId' = c \end{array} \right)$$

The second step transfers the application data into the application artifact relation, using its corresponding function variables, at the same time resetting all application-related artifact variables to `undef` (including *aState*, so that new applications can be inserted). For the insertion, a "free" index (i.e., an index pointing to an undefined applicant) is picked. The newly inserted application gets a default score of `-1` (thus initializing it to "not eligible"), while the final result is `undef`:

$$\exists i{:}\mathsf{appIndex}\left(\begin{array}{l} pState = \mathtt{enabled} \wedge aState = \mathtt{received} \\ \wedge\, applicant[i] = \mathtt{undef} \\ \wedge\, pState' = \mathtt{enabled} \wedge aState' = \mathtt{undef} \wedge cId' = \mathtt{undef} \\ \wedge\, appJobCat' = \lambda j.\,\big(\mathsf{if}\ j = i\ \mathsf{then}\ jId\ \mathsf{else}\ appJobCat[j]\big) \\ \wedge\, applicant' = \lambda j.\,\big(\mathsf{if}\ j = i\ \mathsf{then}\ uId\ \mathsf{else}\ applicant[j]\big) \\ \wedge\, appResp' = \lambda j.\,\big(\mathsf{if}\ j = i\ \mathsf{then}\ eId\ \mathsf{else}\ appResp[j]\big) \\ \wedge\, appScore' = \lambda j.\,\big(\mathsf{if}\ j = i\ \mathsf{then}\ \mathtt{-1}\ \mathsf{else}\ appScore[j]\big) \\ \wedge\, appResult' = \lambda j.\,\big(\mathsf{if}\ j = i\ \mathsf{then}\ \mathtt{undef}\ \mathsf{else}\ appResult[j]\big) \\ \wedge\, jId' = \mathtt{undef} \wedge uId' = \mathtt{undef} \wedge eId' = \mathtt{undef} \end{array}\right)$$

Notice that such a transition does not prevent the possibility of inserting exactly the same application twice, at different indexes. If this is not wanted, the transition can be suitably changed so as to guarantee that no two identical applications can coexist in the same artifact relation (see Appendix A.1 of Calvanese et al. 2018b for an example).

Each application currently considered as not eligible can be made eligible by assigning a proper score to it:

$$\exists i{:}\mathsf{appIndex}, s{:}\mathsf{Score}\left(\begin{array}{l} pState = \mathtt{enabled} \wedge appScore[i] = \mathtt{-1} \wedge s \geq 0 \\ \wedge\, pState' = \mathtt{enabled} \wedge appScore'[i] = s \end{array}\right)$$

Finally, application results are computed when the process moves to state `notified`. This is handled by the *bulk* transition:

$$\begin{array}{l} pState = \mathtt{enabled} \wedge pState' = \mathtt{notified} \\ \wedge\, appResult' = \lambda j.\left(\begin{array}{l} \mathsf{if}\ appScore[j] > 80\ \mathsf{then}\ \mathtt{winner} \\ \mathsf{else}\ \mathtt{loser} \end{array}\right) \end{array}$$

which declares applications with a score above 80 as winning, and the others as losing. ◁

Parameterized safety via backward reachability for RAS. As for SAS, a *safety* formula for a RAS $\mathcal{S}$ is a state formula $\upsilon(\underline{x})$. We say that $\mathcal{S}$ is *safe with respect to* υ if there is no DB instance $\mathcal{M}$ of $\langle \Sigma_{ext}, T\rangle$, no $k \geq 0$, and no assignment in $\mathcal{M}$ to the variables $\underline{x}^0, \underline{a}^0 \ldots, \underline{x}^k, \underline{a}^k$ such that the formula

$$\begin{aligned} &\iota(\underline{x}^0, \underline{a}^0) \wedge \tau(\underline{x}^0, \underline{a}^0, \underline{x}^1, \underline{a}^1) \\ &\qquad \wedge \cdots \wedge \tau(\underline{x}^{k-1}, \underline{a}^{k-1}, \underline{x}^k, \underline{a}^k) \wedge \upsilon(\underline{x}^k, \underline{a}^k) \end{aligned} \tag{14}$$

is true in $\mathcal{M}$ (here $\underline{x}^i, \underline{a}^i$ are renamed copies of $\underline{x}, \underline{a}$). The safety problem is defined as for SAS.

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

Example 4.4. We consider a safety property for the RAS from Example 4.3 that checks whether, after having received the evaluation notification, there are no applicants left without winner or loser status being assigned:

$$\exists i{:}\mathsf{appIndex}\left(\begin{array}{l} pState = \mathtt{notified} \land applicant[i] \neq \mathtt{undef} \\ \land\ appResult[i] \neq \mathtt{winner} \land appResult[i] \neq \mathtt{loser} \end{array}\right)$$

The job hiring RAS $\mathcal{S}_{hr}$ turns out to be safe with respect to this property (cf. Section 6). ◁

Interestingly, we can still run backward search for handling safety problems in RASs. In fact, Algorithm 1 presents the same structure. Notice that in this case the definition of $Pre(\tau, \phi)$ gives us $\exists \underline{x}' \exists \underline{a}' (\tau(\underline{x}, \underline{a}, \underline{x}', \underline{a}') \land \phi(\underline{x}', \underline{a}'))$. The subprocedure $\mathsf{QE}(T^*, \phi)$ mentioned on Line 6 is extended so as to convert the preimage $Pre(\tau, \phi)$ of a state formula ϕ into a state formula (equivalent to it modulo the axioms of T^*), witnessing its *regressability*: this is possible since T^* eliminates from primitive formulae the existentially quantified variables over the basic sorts, whereas elimination of quantified variables over artifact sorts is not possible, because these variables occur as arguments of artifact components (see Lemmas 4.1 and 4.2 below for details). In addition, the satisfiability tests from Lines 2–3 can still be discharged (in fact, we prove in Lemma 4.3 below that the entailment between state formulae can be decided via instantiation techniques). In the following, when we refer to Algorithm 1, *we mean Algorithm 1 adapted to RASs as explained above.*

In analogy to Statement (1) of Theorem 4.2, we obtain the following.

Theorem 4.4. *Backward search (cf. Algorithm 1) is effective and partially correct for solving safety problems for RASs.*

The proof of the above result occupies next paragraph.

Proof of Theorem 4.4. When introducing our transition formulae in (6) and (13), we made use of definable extensions and also of some function definitions via λ-abstraction. We already observed that such uses are due to notational convenience and do not really go beyond first-order logic. We are clarifying one more point now, before going into formal proofs. The lambda-abstraction definitions in (13) will make the proof of Lemma 4.1 below smooth. Recall that an expression like

$$b = \lambda y.F(y, \underline{z})$$

can be seen as a mere abbreviation of $\forall y\ b(y) = F(y, \underline{z})$. However, the use of such abbreviation makes clear that, for example, a formula like

$$\exists b\ (b = \lambda y.F(y, \underline{z}) \land \phi(\underline{z}, b))$$

is equivalent to

$$\phi(\underline{z}, \lambda y.F(y, \underline{z})/b). \tag{15}$$

Since our $\phi(\underline{z}, b)$ is in fact a first-order formula, our b can occur in it only in terms like $b(t)$, so that in (15) all occurrences of λ can be eliminated by the so-called β-conversion: replace $\lambda y F(y, \underline{z})(t)$ by $F(t, \underline{z})$. Thus, in the end, either we use definable extensions or definitions via lambda abstractions, *the formulae we manipulate can always be converted into plain first-order* Σ*- or* Σ_{ext}*-formulae.*

Let us call *extended state formulae* the formulae of the kind $\exists \underline{e}\ \phi(\underline{e}, \underline{x}, \underline{a})$, where ϕ is quantifier-free and the $\underline{e}$ are individual variables of both artifact and basic sorts.

Lemma 4.1. *The preimage of an extended state formula is logically equivalent to an extended state formula.*

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

Proof. We manipulate the formula

$$\exists \underline{x}' \, \exists \underline{a}' \, (\tau(\underline{x}, \underline{a}, \underline{x}', \underline{a}') \wedge \exists \underline{e} \, \phi(\underline{e}, \underline{x}', \underline{a}')) \tag{16}$$

up to logical equivalence, where τ is given by[5]

$$\exists \underline{e}_0 \left(\gamma(\underline{e}_0, \underline{x}, \underline{a}) \wedge \underline{x}' = \underline{F}(\underline{e}_0, \underline{x}, \underline{a}) \wedge \underline{a}' = \lambda y. \underline{G}(y, \underline{e}_0, \underline{x}, \underline{a}) \right) \tag{17}$$

(here, we used plain equality for conjunctions of equalities, for example, $\underline{x}' = \underline{F}(\underline{e}_0, \underline{x}, \underline{a})$ stands for $\bigwedge_i x_i' = F_i(\underline{e}, \underline{x}, \underline{a})$). Repeated substitutions show that (16) is equivalent to

$$\exists \underline{e} \, \exists \underline{e}_0 \, \left(\gamma(\underline{e}_0, \underline{x}, \underline{a}) \wedge \phi(\underline{e}, \underline{F}(\underline{e}_0, \underline{x}, \underline{a})/\underline{x}', \lambda y. \underline{G}(y, \underline{e}_0, \underline{x}, \underline{a})/\underline{a}') \right) \tag{18}$$

which is an extended state formula. □

Lemma 4.2. *For every extended state formula, there is a state formula equivalent to it in all Σ_{ext}-models of T^*.*

Proof. Let $\exists \underline{e} \, \exists \underline{y} \, \phi(\underline{e}, \underline{y}, \underline{x}, \underline{a})$ be an extended state formula, where ϕ is quantifier-free, the $\underline{e}$ are variables whose sort is an artifact sort, and the $\underline{y}$ are variables whose sort is a basic sort.

Now observe that, according to our definitions, the artifact components have an artifact sort as source sort and a basic sort as target sort; since equality is the only predicate, the literals in ϕ can be divided into equalities/inequalities between variables from $\underline{e}$ and literals where the $\underline{e}$ can only occur as arguments of an artifact component. Let $\underline{a}[\underline{e}]$ be the tuple of the terms among the terms of the kind $a_j[e_s]$ which are well typed; using disjunctive normal forms, our extended state formula can be written as a disjunction of formulae of the kind

$$\exists \underline{e} \, \exists \underline{y} \, (\phi_1(\underline{e}) \wedge \phi_2(\underline{y}, \underline{x}, \underline{a}[\underline{e}]/\underline{z})) \tag{19}$$

where ϕ_1 is a conjunction of equalities/inequalities, $\phi_2(\underline{y}, \underline{x}, \underline{z})$ is a quantifier-free Σ-formula, and $\phi_2(\underline{y}, \underline{x}, \underline{a}[\underline{e}]/\underline{z})$ is obtained from ϕ_2 by replacing the variables $\underline{z}$ by the terms $\underline{a}[\underline{e}]$. Moving inside the existential quantifiers $\underline{y}$, we can rewrite (19) to

$$\exists \underline{e} \, (\phi_1(\underline{e}) \wedge \, \exists \underline{y} \, \phi_2(\underline{y}, \underline{x}, \underline{a}[\underline{e}]/\underline{z})) \tag{20}$$

Since T^* has quantifier elimination, we have that there is $\psi(\underline{x}, \underline{z})$ which is equivalent to $\exists \underline{y} \, \phi_2(\underline{y}, \underline{x}, \underline{z}))$ in all models of T^*; thus in all Σ_{ext}-models of T^*, the formula (20) is equivalent to

$$\exists \underline{e} \, (\phi_1(\underline{e}) \wedge \, \psi(\underline{x}, \underline{a}[\underline{e}]/\underline{z}))$$

which is a state formula. □

We underline that Lemmas 4.1 and 4.2 both give an explicit effective procedure for computing equivalent (extended) state formulae. Used one after the other, such procedures extend the procedure $QE(T^*, \phi)$ in Line 6 of Algorithm 1 to (non-simple) artifact systems. Thanks to such procedure, the only formulae we need to test for satisfiability in lines 2 and 3 of the backward reachability algorithm are the $\exists\forall$-formulae introduced below.

Let us call $\exists\forall$-formulae the formulae of the kind

$$\exists \underline{e} \, \forall \underline{i} \, \phi(\underline{e}, \underline{i}, \underline{x}, \underline{a}) \tag{21}$$

where the variables $\underline{e}$ and $\underline{i}$ are variables whose sort is an artifact sort and ϕ is quantifier-free. The crucial point for the following lemma to hold is that the *universally* quantified variables in $\exists\forall$-formulae are all of artifact sorts:

Lemma 4.3. *The satisfiability of a $\exists\forall$-formula in a Σ_{ext}-model of T is decidable. Moreover, given a $\exists\forall$-formula χ, the following three statements are equivalent:*

- — *χ is satisfiable in a Σ_{ext}-model of T*
- — *χ is satisfiable in a DB instance of $\langle \Sigma_{ext}, T\rangle$*
- — *χ is satisfiable in a Σ_{ext}-model of T^*.*

Proof. First of all, notice that a $\exists\forall$-formula (21) is equivalent to a disjunction of formulae of the kind

$$\exists \underline{e}\,(\mathrm{AllDiff}(\underline{e}) \wedge \forall \underline{i}\,\phi(\underline{e}, \underline{i}, \underline{x}, \underline{a})) \tag{22}$$

where AllDiff($\underline{e}$) says that any two variables of the same sort from the $\underline{e}$ are distinct (to this aim, it is sufficient to guess a partition and to keep, via a substitution, only one element for each equivalence class).[6] So we can freely assume that $\exists\forall$-formulae are all of the kind (22).

Let us consider now the set of all (sort-matching) substitutions σ mapping the $\underline{i}$ to the $\underline{e}$. The formula (22) is satisfiable (respectively, in a Σ_{ext}-model of T, in a DB instance of $\langle \Sigma_{ext}, T\rangle$, in a Σ_{ext}-model of T^*) iff so it is the formula

$$\exists \underline{e}\,(\mathrm{AllDiff}(\underline{e}) \wedge \bigwedge_{\sigma} \phi(\underline{e}, \underline{i}\sigma, \underline{x}, \underline{a})) \tag{23}$$

(here $\underline{i}\sigma$ means the componentwise application of σ to the $\underline{i}$): this is because, if (23) is satisfiable in $\mathcal{M}$, then we can take as $\mathcal{M}'$ the same Σ_{ext}-structure as $\mathcal{M}$, but with the interpretation of the artifact sorts restricted only to the elements named by the $\underline{e}$ and get in this way a Σ_{ext}-structure $\mathcal{M}'$ satisfying (22) (notice that $\mathcal{M}'$ is still a DB instance of $\langle \Sigma_{ext}, T\rangle$ or a Σ_{ext}-model of T^*, if so was $\mathcal{M}$). Thus, we can freely concentrate on the satisfiability problem of formulae of the kind (23) only.

Now, by the way Σ_{ext} is built, the only atoms occurring in the subformula $\phi(\underline{e}, \underline{i}\sigma, \underline{x}, \underline{a})$) of (23) whose argument terms are terms of artifact sorts are of the kind $e_s = e_j$, so all such atoms can be replaced either by $\top$ or by $\bot$ (depending on whether we have $s = j$ or not). So we can assume that there are no such atoms in $\phi(\underline{e}, \underline{i}\sigma, \underline{x}, \underline{a})$) and as a result, the variables $\underline{e}$ can only occur there as arguments of the $\underline{a}$.

Let now $\underline{a}[\underline{e}]$ be the tuple of the terms among the terms of the kind $a_j[e_s]$ which are well typed. Since in (23), the $\underline{e}$ can only occur as arguments of the artifact components, as observed above, the formula (23) is in fact of the kind

$$\exists \underline{e}\,(\mathrm{AllDiff}(\underline{e}) \wedge \psi(\underline{x}, \underline{a}[\underline{e}]/\underline{z})) \tag{24}$$

where $\psi(\underline{x}, \underline{z})$ is a quantifier-free Σ-formula and $\psi(\underline{x}, \underline{a}[\underline{e}]/\underline{z})$ is obtained from ψ by replacing the variables $\underline{z}$ by the terms $\underline{a}[\underline{e}]$ (notice that the $\underline{z}$ are of basic sorts because the target sorts of the artifact components are basic sorts).

It is now evident that (24) is satisfiable (respectively, in a Σ_{ext}-model of T, in a DB instance of $\langle \Sigma_{ext}, T\rangle$, in a Σ_{ext}-model of T^*) iff the formula

$$\psi(\underline{x}, \underline{z}) \tag{25}$$

is satisfiable (respectively, in a Σ-model of T, in a DB instance of $\langle \Sigma, T\rangle$, in a Σ-model of T^*). In fact, if we are given a Σ-structure $\mathcal{M}$ and an assignment satisfying (25), we can easily expand $\mathcal{M}$ to a Σ_{ext}-structure by taking the e's themselves as the elements of the interpretation of the artifact sorts; in the so-expanded Σ_{ext}-structure, we can interpret the artifact components $\underline{a}$ by taking the $\underline{a}[\underline{e}]$ to be the elements assigned to the $\underline{z}$ in the satisfying assignment for (25).

Thanks to Assumption 3.4, the satisfiability of (25) in a Σ-model of T, in a DB instance of $\langle \Sigma, T\rangle$, or in a Σ-model of T^* are all equivalent and decidable. □

The instantiation algorithm of Lemma 4.3 can be used to discharge the satisfiability tests in lines 2 and 3 of Algorithm 1 because the conjunction of a state formula and of the negation of a state formula is a $\exists\forall$-formula (notice that ι is itself the negation of a state formula, according to (11)).

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

Theorem 4.4. *The backward search algorithm (cf. Algorithm 1) is effective and partially correct for solving the safety problem for RASs.*

Proof. Recall that $\mathcal{S}$ is unsafe iff there is no DB instance $\mathcal{M}$ of $\langle \Sigma_{ext}, T\rangle$, no $k \geq 0$, and no assignment in $\mathcal{M}$ to the variables $\underline{x}^0, \underline{a}^0 \ldots, \underline{x}^k, \underline{a}^k$ such that the formula (14)

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

is true in $\mathcal{M}$. It is sufficient to show that this is equivalent to saying that there is no Σ_{ext}-model $\mathcal{M}$ of T^*, no $k \geq 0$, and no assignment in $\mathcal{M}$ to the variables $\underline{x}^0, \underline{a}^0 \ldots, \underline{x}^k, \underline{a}^k$ such that (14) is true in $\mathcal{M}$ (once this is shown, the proof goes in the same way as the proof of Theorem 4.2).

Now, the formula (14) is satisfiable in a Σ_{ext}-structure $\mathcal{M}$ under a suitable assignment iff the formula

$$\begin{aligned} &\iota(\underline{x}^0, \underline{a}^0) \ \wedge \ \exists \underline{a}^1 \exists \underline{x}^1 (\tau(\underline{x}^0, \underline{a}^0, \underline{x}^1, \underline{a}^1) \wedge \cdots \\ &\qquad \cdots \wedge \exists \underline{a}^k \exists \underline{x}^k (\tau(\underline{x}^{k-1}, \underline{a}^{k-1}, \underline{x}^k, \underline{a}^k) \wedge \upsilon(\underline{x}^k, \underline{a}^k)) \cdots) \end{aligned}$$

is satisfiable in $\mathcal{M}$ under a suitable assignment; by Lemma 4.1, the latter is equivalent to a formula of the kind

$$\iota(\underline{x}, \underline{a}) \ \wedge \ \exists \underline{e}\, \exists \underline{y}\, \phi(\underline{e}, \underline{y}, \underline{x}, \underline{a}) \tag{26}$$

where $\exists \underline{e}\, \exists \underline{y}\, \phi(\underline{e}, \underline{y}, \underline{x}, \underline{a})$ is an extended state formula (thus ϕ is quantifier-free, the $\underline{e}$ are variables of artifact sorts, and the $\underline{y}$ are variables of basic sorts - we renamed $\underline{x}^0, \underline{a}^0$ as $\underline{x}, \underline{a}$). However, the satisfiability of (26) is the same as the satisfiability of $\exists \underline{e}\, (\iota(\underline{x}, \underline{a}) \wedge \phi(\underline{e}, \underline{y}, \underline{x}, \underline{a}))$; the latter, in view of (11), is a $\exists\forall$-formula and so Lemma 4.3 applies and shows that its satisfiability in a DB instance of $\langle \Sigma_{ext}, T\rangle$ is the same as its satisfiability in a Σ_{ext}-model of T^*. □

To sum up, in this subsection, we remarked that for Algorithm 1, to be effective, we need decision procedures for discharging the satisfiability tests in Lines 2–3. Thanks to the subprocedure $\mathsf{QE}(T^*, \phi)$, the only formulae we need to test in these lines have a specific form (i.e., they are $\exists\forall$-formulae). In fact, by our hypotheses in Assumption 3.4, we can freely assume that all the runs we are interested in take place inside models of T^* (where we can eliminate quantifiers binding variables of basic sorts). Then, in two first technical lemmas (Lemmas 4.1 and 4.2), we show that the preimage of a state formula is an extended state formula and that such an extended state formula can be converted back (modulo T^*) into a state formula; finally, in a third technical lemma (Lemma 4.3), we show that entailments between state formulae (more generally, satisfiability of formulae of the kind $\exists\forall$) can be decided via finite instantiation techniques. These observations make both safety and fixpoint tests effective and constitute the skeleton of the proof of Theorem 4.4.

Remark 4.2. Notice that the role of quantifier elimination (Line 6 of Algorithm 1) is twofold: *(i)* It allows to discharge the fixpoint test of Line 2 (see Lemma 4.3). *(ii)* It ensures termination in significant cases, namely those where *(strongly) local formulae*, introduced in the next section, are involved.

5. Termination Results for RASs

Theorem 4.4 gives a semi-decision procedure for unsafety: if the system is unsafe, the procedure discovers it, but if the system is safe, the procedure (still correct) may not terminate. Termination is much more difficult to achieve for RASs, since acyclicity of Σ seems not to be sufficient to guarantee it. We present two termination results for RASs, both obtained via the use of well-quasi-orders. The strategy for proving termination consists of isolating sufficient conditions that

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

imply that the embeddability relation between DB instances is a well-quasi-ordering. Since there is no guarantee that this fact holds in general, RASs are *not* well-structured transition systems.

5.1 Termination with local updates

Consider an acyclic signature Σ, a theory T (satisfying our Assumption 3.4), and an artifact setting $(\underline{x}, \underline{a})$ over an artifact extension Σ_{ext} of Σ. We call a state formula *local* if it is a disjunction of the formulae

$$\exists e_1 \cdots \exists e_k\, (\delta(e_1, \ldots, e_k) \wedge \textstyle\bigwedge_{i=1}^{k} \phi_i(e_i, \underline{x}, \underline{a})), \tag{27}$$

and *strongly local* if it is a disjunction of the formulae

$$\exists e_1 \cdots \exists e_k\, (\delta(e_1, \ldots, e_k) \wedge \psi(\underline{x}) \wedge \textstyle\bigwedge_{i=1}^{k} \phi_i(e_i, \underline{a})). \tag{28}$$

In (27) and (28), δ is a conjunction of variable equalities and disequalities, ϕ_i and ψ are quantifier-free, and $e_1, \ldots, e_k$ are individual variables varying over artifact sorts. The key expressivity limitation of local state formulae is that they cannot compare entries belonging to different tuples of artifact relations: in fact, each ϕ_i in (27) and (28) can contain only the existentially quantified variable e_i.

A transition formula $\hat{\tau}$ is *local* (respectively, *strongly local*) if whenever a formula ϕ is local (respectively, strongly local), so is $Pre(\hat{\tau}, \phi)$ (modulo the axioms of T^*).

Below in Theorem 5.4, we show that (for acyclic Σ) Algorithm 1 terminates when applied to a local safety formula in an RAS whose τ is a disjunction of local transition formulae. Note that Theorem 5.4 can be used to reconstruct (restricted to safety problems) the essence of the decidability results of Li et al. (2017). Specifically, it can be shown by a direct computation that transitions in Li et al. (2017) are strongly local which, in turn, can be shown using quantifier elimination (see Appendix F in Calvanese et al. 2018b for all the details, where we also show how to represent transitions from Li et al. 2017 by the means of existentially quantified data variables). Interestingly, Theorem 5.4 can be applied to more cases not covered in Li et al. (2017). For example, one can provide transitions enforcing *updates over unboundedly many* tuples (bulk updates) that are strongly local (cf. Appendix F in Calvanese et al. 2018b).

Example 5.1. By inspecting the transitions of our running example, one can see that all of them are strongly local. Consequently, it is decidable to check safety of local state formulae. For example, we show that the first transition of Example 4.3 is strongly local (the computations for all the other transitions are analogous, and all the details about the format of transitions that are (strongly) local can be found in Appendix F of Calvanese et al. 2018b).

The first transition of Example 4.3 represents the first step of the insertion of an application into the system. This step is executable when the artifact variable *aState* is `undef`, aims at loading the application data (user ID, job ID, and employee ID) into dedicated artifact variables (*uId*, *jId*, *eId*, respectively), and evolves *aState* into state `received`. Formally, we have

$$\begin{array}{l}\exists u{:}\mathtt{UserId}, j{:}\mathsf{JobCatId}, e{:}\mathsf{EmpId}, c{:}\mathsf{CompInId} \\ \left(\begin{array}{l} pState = \mathtt{enabled} \wedge aState = \mathtt{undef} \\ \wedge\, u \neq \mathtt{undef} \wedge j \neq \mathtt{undef} \wedge e \neq \mathtt{undef} \wedge c \neq \mathtt{undef} \\ \wedge\, who(c) = e \wedge what(c) = j \\ \wedge\, pState' = \mathtt{enabled} \wedge aState' = \mathtt{received} \\ \wedge\, uId' = u \wedge = j \wedge eId' = e \wedge cId' = c \end{array}\right)\end{array} \tag{29}$$

For simplicity, we can rewrite Formula (29) into the following equivalent but more succinct formula:

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

$$\exists \underline{d} \begin{pmatrix} \pi(\underline{x}_1, \underline{x}_2) \wedge \psi(\underline{d}) \wedge d_1 = \texttt{enabled} \wedge d_2 = \texttt{received} \\ \wedge (\underline{x}_1' := \underline{x}_1 \wedge \underline{x}_2' := \underline{d} \wedge \underline{a}' := \underline{a}) \end{pmatrix} \tag{30}$$

where $\underline{d} := \langle d_1, d_2, u, j, e, c \rangle$, $\underline{x}_1$ are the artifact variables of the system that are *not* updated, $\underline{x}_2$ are the artifact variables of the system that are updated, $\pi(\underline{x}_1, \underline{x}_2)$ and $\psi(\underline{d})$ are quantifier-free Σ-formulae, and $\underline{a}$ are the artifact components of the systems.

We show that the preimage along (30) of a strongly local formula is strongly local.

Given a strongly local state formula ϕ, we can easily suppose that ϕ has the following format:

$$\phi := \psi'(\underline{x}) \wedge \exists \underline{i} \left(\mathrm{AllDiff}(\underline{i}) \wedge \Theta(\underline{a}) \right)$$

where $\underline{x}$ are all the artifact variables of the system, $\underline{i}$ are variables of artifact sorts, and Θ is a formula involving all the artifact components $\underline{a}$.

We compute the preimage $Pre(30, \phi)$:

$$\exists \underline{d} \begin{pmatrix} \pi(\underline{x}_1, \underline{x}_2) \wedge \psi(\underline{d}) \wedge d_1 = \texttt{enabled} \wedge d_2 = \texttt{received} \\ \wedge (\underline{x}_1' := \underline{x}_1 \wedge \underline{x}_2' := \underline{d} \wedge \underline{a}' := \underline{a}) \\ \wedge \psi'(\underline{x}') \wedge \exists \underline{i} \left(\mathrm{AllDiff}(\underline{i}) \wedge \Theta(\underline{a}') \right) \end{pmatrix} \tag{31}$$

which can be rewritten as follows:

$$\exists \underline{d} \begin{pmatrix} \pi(\underline{x}_1, \underline{x}_2) \wedge \psi(\underline{d}) \wedge d_1 = \texttt{enabled} \wedge d_2 = \texttt{received} \\ \wedge \psi'(\underline{x}_1, \underline{d}) \wedge \exists \underline{i} \left(\mathrm{AllDiff}(\underline{i}) \wedge \Theta(\underline{a}) \right) \end{pmatrix} \tag{32}$$

Now, we can move the existential quantifier $\exists \underline{d}$ in front of $\chi(\underline{d}, \underline{x}_1) := (\psi(\underline{d}) \wedge d_1 = \texttt{enabled} \wedge d_2 = \texttt{received} \wedge \psi'(\underline{x}_1, \underline{d}))$. We eliminate the quantifiers (applying the quantifier elimination procedure for T^*) from the subformula $\exists \underline{d}(\chi(\underline{d}, \underline{x}_1))$ obtaining a formula of the kind $\theta(\underline{x}_1)$.

The final result is

$$\pi(\underline{x}_1, \underline{x}_2) \wedge \theta(\underline{x}_1) \wedge \exists \underline{i} \left(\mathrm{AllDiff}(\underline{i}) \wedge \Theta(\underline{a}) \right) \tag{33}$$

which is a strongly local formula.

The interested reader can find additional details about applications of (strongly) local RASs to data-aware business processes in Calvanese et al. (2019a). Specifically, this paper contains a running example (verified against several properties) that can be represented using a RAS that is strongly local.

In addition, Theorem 5.4 covers also problems coming from a different source, like coverability problems for broadcast protocols (Delzanno et al. 1999; Esparza et al. 1999): these problems can be encoded using local formulae over the trivial one-sorted signature containing just one basic sort, finitely many constants and one artifact sort with one artifact component. We remark that coverability for broadcast protocols can be decided with a non-primitive recursive lower bound (Schmitz and Schnoebelen 2013); this proves that our framework is quite expressive (the problems in Li et al. 2017 have, for instance, an ExpSpace upper bound). Recalling that Li et al. (2017) handles verification of LTL-FO (First Order Linear Temporal Logic), thus going beyond safety problems, this shows that the two settings are incomparable. Finally, notice that Theorem 5.4 implies also the decidability of the safety problem for SASs, in case of Σ acyclic.

Before stating and proving Theorem 5.4, we need to recall some basic facts about well-quasi-orders. Recall that a *well-quasi-order* (wqo) is a set W endowed with a reflexive-transitive relation $\leq$ having the following property: for every infinite succession

$$w_0, w_1, \ldots, w_i, \ldots$$

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

of elements from W, there are i, j such that $i < j$ and $w_i \leq w_j$ The fundamental result about wqo's is the following theorem, which is a recursive version of Higman's Lemma (Higman 1952), and is a special case of the well-known Kruskal's Tree Theorem (Kruskal 1960):

Theorem 5.1. *If* $(W, \leq)$ *is a wqo, then so is the partial order of the finite lists over* W*, ordered by componentwise subword comparison (i.e.,* $w \leq w'$ *iff there is a subword* w_0 *of* w' *of the same length as* w*, such that the* i*-th entry of* w *is less or equal to – in the sense of* $(W, \leq)$ *– the* i*-th entry of* w_0*, for all* $i = 0, \ldots |w|$*).*

Various wqo's can be recognized by applying the above theorem; in particular, the theorem implies that the cartesian product of wqo's is a wqo. As an application, notice that $\mathbb{N}$ is a wqo, hence the following corollary (known as Dickson's Lemma) follows:

Corollary 5.2. *The cartesian product of* k*-copies of* $\mathbb{N}$ *(and also of* $\mathbb{N} \cup \{\infty\}$*), with componentwise ordering, is a wqo.*

Let $\tilde{\Sigma}$ be $\Sigma_{ext} \cup \{\underline{a}, \underline{x}\}$, that is, Σ_{ext} expanded with function symbols $\underline{a}$ and constants $\underline{x}$ (thus, a $\tilde{\Sigma}$-structure is a Σ_{ext}-structure endowed with an assignment to $\underline{x}$ and $\underline{a}$, which were variables and now are treated as symbols of $\tilde{\Sigma}$). For the following, we need the following definition:

Definition 5.3. *A* $\tilde{\Sigma}$*-structure* $\mathcal{M}$ *is called* cyclic[7] *if it is generated by a* single *element* $e \in E^{\mathcal{M}}$ *(called* generator *of* $\mathcal{M}$*), where* E *is an artifact sort (i.e.,* e *belongs to the interpretation of an artifact sort* E*).*

The previous definition intuitively means that *all* the elements of the cyclic structures are obtained from the generator by applying the function symbols of $\tilde{\Sigma}$ to the generator.

Since Σ is acyclic, so is $\tilde{\Sigma}$, and then one can show that there are only finitely many cyclic $\tilde{\Sigma}$-structures $\mathcal{C}_1, \ldots, \mathcal{C}_N$ up to isomorphism. With a $\tilde{\Sigma}$-structure $\mathcal{M}$, we associate the tuple of numbers $k_1(\mathcal{M}), \ldots, k_N(\mathcal{M}) \in \mathbb{N} \cup \{\infty\}$ counting the numbers of elements generating (as singletons) cyclic substructures isomorphic to $\mathcal{C}_1, \ldots, \mathcal{C}_N$, respectively.

Now, we show that, if the tuple associated with $\mathcal{M}$ is componentwise bigger than the one associated with $\mathcal{N}$, then $\mathcal{M}$ satisfies all the local formulae satisfied by $\mathcal{N}$.

Lemma 5.1. *Let* $\mathcal{M}, \mathcal{N}$ *be* $\tilde{\Sigma}$*-structures. If the inequalities*

$$k_1(\mathcal{M}) \leq k_1(\mathcal{N}), \ldots, k_N(\mathcal{M}) \leq k_N(\mathcal{N})$$

hold, then all local formulae true in $\mathcal{M}$ *are also true in* $\mathcal{N}$*.*

Proof. Notice that local formulae (viewed in $\tilde{\Sigma}$) are sentences, because they do not have free variable occurrences – the $\underline{a}, \underline{x}$ are now constant function symbols and individual constants, respectively. The proof of the lemma is fairly obvious: notice that, once we assigned some $\alpha(e_i)$ in $\mathcal{M}$ to the variable e_i, the truth of a formula like $\phi(e_i, \underline{x}, \underline{a})$ under such an assignment depends only on the $\tilde{\Sigma}$-substructure generated by $\alpha(e_i)$, because ϕ is quantifier-free and e_i is the only $\tilde{\Sigma}$-variable occurring in it. In fact, if a local state formula $\exists e_1 \cdots \exists e_k \left(\delta(e_1, \ldots, e_k) \wedge \bigwedge_{i=1}^{k} \phi_i(e_i, \underline{x}, \underline{a})\right)$ is true in $\mathcal{M}$, then there exist elements $\bar{e}_1, \ldots, \bar{e}_k$ (in the interpretation of some artifact sorts), each of which makes ϕ_i true. Hence, ϕ_i is also true in the corresponding cyclic structure generated by $\bar{e}_i$. Since $k_1(\mathcal{M}) \leq k_1(\mathcal{N}), \ldots, k_N(\mathcal{M}) \leq k_N(\mathcal{N})$ hold, then also in $\mathcal{N}$ there are at least as many elements in the interpretation of artifact sorts as there are in $\mathcal{M}$ that validate all the ϕ_i . Thus, we get that the formula $\exists e_1 \cdots \exists e_k \left(\delta(e_1, \ldots, e_k) \wedge \bigwedge_{i=1}^{k} \phi_i(e_i, \underline{x}, \underline{a})\right)$ is true also in $\mathcal{N}$, as wanted. □

Now we are ready to prove our first termination and decidability result.

Theorem 5.4. *If* Σ *is acyclic, backward search (cf. Algorithm 1) terminates when applied to solve the safety problem, with respect to a* (strongly) local *safety formula* $\upsilon(\underline{x}, \underline{a})$*, for an*

RAS $\langle \Sigma, T, \Sigma_{ext}, \underline{x}, \underline{a}, \iota(\underline{x}, \underline{a}), \tau(\underline{x}, \underline{a}, \underline{x}', \underline{a}')\rangle$, *where* τ *is a disjunction of* (strongly) local *transition formulae.*

Proof. Suppose the algorithm does not terminate. Then the fixpoint test of Line 2 fails infinitely often. Recalling that the T-equivalence of B_n and of $\bigvee_{0\leq k<n} \phi_k$ is an invariant of the algorithm (here ϕ_n, B_n are the status of the variables ϕ and B after n execution of the main loop), this means that there are models

$$\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_j, \ldots$$

such that for all j, we have that $\mathcal{M}_j \models \phi_j$ and $\mathcal{M}_j \not\models \phi_i$ (all $i < j$). But the ϕ_j are all local formulae, so considering the tuple of cardinals $k_1(\mathcal{M}_j), \ldots, k_N(\mathcal{M}_j)$ and Lemma 5.1, we get a contradiction, in view of Dickson's Lemma. This is because, by Dickson's Lemma, $(\mathbb{N} \cup \{\infty\})^N$ is a wqo, so there exist i, j such that $i < j$ and $k_1(\mathcal{M}_i) \leq k_1(\mathcal{M}_j), \ldots, k_N(\mathcal{M}_i) \leq k_N(\mathcal{M}_j)$. Using Lemma 5.1, we get that ϕ_i, which is local and true in $\mathcal{M}_i$, is also true in $\mathcal{M}_j$, which is a contradiction. □

5.2 Termination with tree-like signatures

Σ is *tree-like* if it is acyclic and all non-leaf nodes have outdegree 1. An artifact setting over Σ is tree-like if $\tilde{\Sigma} := \Sigma_{ext} \cup \{\underline{a}, \underline{x}\}$ is tree-like. In tree-like artifact settings, artifact relations have a single "data" component, and basic relations are unary or binary.

Proving termination for RAS with a tree-like artifact setting is more complex but follows a similar schema as in the case of local transition formulae.

If $(W, \leq)$ is a partial order, we consider the set $M(W)$ of finite multisets of W as a partial order in the following way:[8] say that $M \leq N$ holds iff there is an injection $p : M \longrightarrow N$ such that $m \leq p(m)$ holds for all $m \in M$ (in other words, p associates with every occurrence of an element m of M an occurrence $p(m)$ of an element of N such that $p(m) \geq m$ – this is moreover done injectively, that is, in such a way that different occurrences are associated to different occurrences).

Corollary 5.5. *If* $(W, \leq)$ *is a wqo, then so is* $(M(W), \leq)$ *as defined above.*

Proof. This is due to the fact that one can convert a multiset M to a list $L(M)$ so that if $L(M) \leq L(N)$ holds, then also $M \leq N$ holds (such a conversion L can be obtained by ordering the occurrences of elements in M in any arbitrarily chosen way). □

We assume that the graph $G(\tilde{\Sigma})$ associated to $\tilde{\Sigma}$ is a tree (the generalization to the case where such a graph is a forest is trivial). This means in particular that each sort is the domain of at most one function symbol and that there just one sort which is not the domain of any function symbol (let us call it the *root sort* of $\tilde{\Sigma}$ and let us denote it with S_r).

By induction on the height of a sort S (defined as the length of the longest path from S to a leaf) in the above graph, we define a wqo $w(S)$ (in the definition, we use the fact that the cartesian product of wqo's is a wqo and Corollary 5.5). Let $S_1, \ldots, S_n$ be the sons of S in the tree; we put

$$w(S) := M(w(S_1)) \times \cdots \times M(w(S_n)) \tag{34}$$

(thus, if S is a leaf, $w(S)$ is the trivial one-element wqo – its only element is the empty tuple).

Let now $\mathcal{M}$ be a finite $\tilde{\Sigma}$-structure; we indicate with $S^{\mathcal{M}}$ the interpretation in $\mathcal{M}$ of the sort S (it is a finite set). For $a \in S^{\mathcal{M}}$, we define $M_{\mathcal{M}}(a) \in w(S)$, again by induction on the height of S. Suppose that $S_1, \ldots, S_n$ are the sons of S and that the arc from S_i to S is labeled by the function symbol f_i; then, we put

$$\begin{aligned} M_{\mathcal{M}}(a) := \langle &\{M_{\mathcal{M}}(b_1) \mid b_1 \in S_1^{\mathcal{M}} \text{ and } f_1^{\mathcal{M}}(b_1) = a\}, \ldots \\ &\ldots, \{M_{\mathcal{M}}(b_n) \mid b_n \in S_n^{\mathcal{M}} \text{ and } f_n^{\mathcal{M}}(b_n) = a\}\rangle \end{aligned}$$

where $f_i^{\mathcal{M}}$ $(i = 1, \ldots, n)$ is the interpretation of the symbol f_i in $\mathcal{M}$.

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

Moreover, for every sort S, we let

$$M_{\mathcal{M}}(S) := \{M_{\mathcal{M}}(a) \mid a \in S^{\mathcal{M}}\}. \tag{35}$$

Finally, we define

$$M(\mathcal{M}) := M_{\mathcal{M}}(S_r). \tag{36}$$

For termination, the relevant lemma is the following:

Lemma 5.2. *Suppose that $\tilde{\Sigma}$ is tree-like and does not contain constant symbols; given two finite $\tilde{\Sigma}$-structures $\mathcal{M}$ and $\mathcal{N}$, we have that if $M(\mathcal{M}) \leq M(\mathcal{N})$, then $\mathcal{M}$ embeds into $\mathcal{N}$. As a consequence, the finite $\tilde{\Sigma}$-structures are a wqo with respect to the embeddability quasi-order.*

Proof. Again, we make an induction on the height of S, proving the claim for the subsignature of $\tilde{\Sigma}$ having S as a root (let us call this the S-subsignature).

Let $\mathcal{M}$ be a model over the S-subsignature. For every $a \in S^{\mathcal{M}}$, and for every $f_i : S_i \longrightarrow S$, if we restrict $\mathcal{M}$ to the elements in the f_i-fibers of a, we get a model $\mathcal{M}_{f_i,a}$ for the S_i-subsignature (an element $c \in \tilde{S}^{\mathcal{M}}$ is in the f_i-fiber of a if, taking the term t corresponding to the composition of the functions symbols going from $\tilde{S}$ to S_i, we have that $f_i^{\mathcal{M}}(t^{\mathcal{M}}(c)) = a$). In addition, if $M_{\mathcal{M}}(a) = (M_1, \ldots, M_n)$, then $M_i = M(\mathcal{M}_{f_i,a})$ by definition. Finally, observe that the restriction of $\mathcal{M}$ to the S_i-subsignature is the disjoint union of the f_i-fibers models $\mathcal{M}_{f_i,a}$, varying $a \in S^{\mathcal{M}}$.

Suppose now that $\mathcal{M}, \mathcal{N}$ are models over the S-subsignature such that $M(\mathcal{M}) \leq M(\mathcal{N})$; this means that we can find an injective map μ mapping $S^{\mathcal{M}}$ into $S^{\mathcal{N}}$ so that $M_{\mathcal{M}}(a) \leq M_{\mathcal{N}}(\mu(a))$. If $M_{\mathcal{M}}(a) = (M_1, \ldots, M_n)$ and $M_{\mathcal{N}}(\mu(a)) = (N_1, \ldots, N_n)$, we then have that $M_i \leq N_i$ for every $i = 1, \ldots, n$. Considering that, as noticed above, $M_i = \mathcal{M}_{f_i,a}$ and $N_i = \mathcal{N}_{f_i,\mu(a)}$, by induction hypothesis, we have embeddings $\nu_{i,a}$ for the f_i-fibers models of a and $\mu(a)$ (for every $a \in S^{\mathcal{M}}$ and $i = 1, \ldots, n$). Glueing these embeddings to the disjoint union (varying i, a) and adding them μ as S-component, we get the desired embedding of $\mathcal{M}$ into $\mathcal{N}$. □

Theorem 5.6. *Backward search (cf. Algorithm 1) terminates when applied to a safety problem in an RAS with a tree-like artifact setting.*

Proof. For simplicity, we start giving the argument for the case where we do not have constants and artifact variables. Similarly to the proof of Theorem 5.4, suppose the algorithm does not terminate. Then, the fixpoint test of Line 2 fails infinitely often. Recalling that the T-equivalence of B_n and of $\bigvee_{0 \leq k < n} \phi_k$ is an invariant of the algorithm (here ϕ_n and B_n are the status of the variables ϕ, B after n execution of the main loop), this means that there are models

$$\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_j, \ldots$$

such that for all j, we have that $\mathcal{M}_j \models \phi_j$ and $\mathcal{M}_j \not\models \phi_i$ (all $i < j$). The models can be taken to be all finite, by Lemma 4.3. But the ϕ_j are all existential sentences in $\tilde{\Sigma}$, so this is incompatible to the fact that, by Lemma 5.2, there are $i < j$ with $\mathcal{M}_i$ embeddable into $\mathcal{M}_j$.

Concerning the general case, it is sufficient to consider the following observation that shows how to extend the proof to the case where we have constants and artifact variables. Recall that in $\tilde{\Sigma}$, the artifact variables are seen as constants, so we need to consider only the case of constants. Let $\tilde{\Sigma}^+$ be $\tilde{\Sigma}$ where each constant symbol c of sort S is replaced by a new sort S_c and a new function symbol $f_c : S_c \longrightarrow S$. Now every model $\mathcal{M}$ of $\tilde{\Sigma}$ can be transformed into a model $\mathcal{M}^+$ of $\tilde{\Sigma}^+$ by interpreting S_c as a singleton set $\{*\}$ and f_c as the map sending $*$ to $c^{\mathcal{M}}$. This transformation has the following property: $\tilde{\Sigma}$-embeddings of $\mathcal{M}$ into $\mathcal{N}$ are in bijective correspondence with $\tilde{\Sigma}^+$-embeddings of $\mathcal{M}^+$ into $\mathcal{N}^+$. Since $\tilde{\Sigma}^+$ is still tree-like and does not have constant symbols, this shows that Theorem 5.6 holds for $\tilde{\Sigma}$ too. □

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

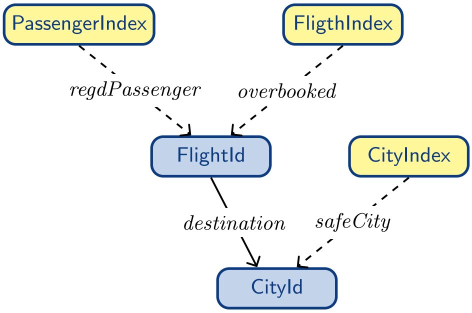

Figure 2. A characteristic graph of the flight management process, where blue and yellow boxes respectively represent basic and artifact sorts.

While tree-like RAS restricts artifact relations to be unary, their transitions are not subject to any locality restriction. This allows for expressing rich forms of updates, including general bulk updates (which allow us to capture non-primitive recursive verification problems[9]) and transitions comparing at once different tuples in artifact relations. The flight management process presented in the following example shows these advanced features, with a tree-like RAS whose safety verification is indeed decidable. Finally, notice that tree-like RASs are incomparable: *(i)* with the "tree" classes of Bojańczyk et al. (2013), since the former use artifact relations, whereas the latter only individual variables; *(ii)* with the decidability class of Li et al. (2017), since tree-like RASs express transitions able to compare at once values stored in different tuples in artifact relations.

Example 5.2. We consider a simple RAS that falls in the scope of the tree-like decidability result. Specifically, this example has a tree-like artifact setting (see Figure 2), thus assuring that, when solving the safety problem for it, the backward search algorithm is guaranteed to terminate. Note, however, that the termination result adopted here is the one of Theorem 5.6 due to the non-locality of certain transitions, as explained in detail below.

The flight management process represents a simplified version of a flight management system adopted by an airline. To prepare a flight, the company picks a corresponding destination (that meets the aviation safety compliance indications) and consequently reports on a number of passengers that are going to attend the flight. Then, an airport dispatcher may pick a manned flight and put it in the airports flight plan. In case the flight destination becomes unsafe (e.g., it was struck by a hurricane or the hosting airport had been seized by terrorists), the dispatcher uses the system to inform the airline about this condition. In turn, the airline notifies all the passengers of the affected destination about the contingency, and temporary cancels their flights.

To formalize these different aspects, we make use of a DB signature Σ_{fm} that consists of: *(i)* two id sorts, used to identify flights and cities; *(ii)* one function symbol $\mathit{destination} : \mathsf{FlightId} \longrightarrow \mathsf{CityId}$ mapping flight identifiers to their corresponding destinations (i.e., city identifiers). Note that, in a classical relational model (cf. Section 3.1), our signature would contain two relations: one binary R_{FlightId} that defines flights and their destinations, and another unary R_{CityId} identifying cities, that are referenced by R_{FlightId} using *destination*.

We assume that the read-only flight management database contains data about at least one flight and one city. To start the process, one needs at least one city to meet the aviation safety compliances. It is assumed that, initially, all the cities are unsafe. An airport dispatcher, at once, may change the safety status only of one city.

We model this action by performing two consequent actions. First, we select the city identifier and store it in the designated artifact variable *safeCitytId*:

$$\exists c{:}\mathsf{CityId}\,\big(c \neq \mathtt{undef} \wedge \mathit{safeCitytId} = \mathtt{undef} \wedge \mathit{safeCitytId}' = c\big)$$

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

Then, we place the extracted city identifier into a unary artifact relation $\mathit{safeCity} : \mathsf{CityIndex} \longrightarrow \mathsf{CityId}$, that is used to represent safe cities and where $\mathsf{CityIndex}$ is its artifact sort.

$$\exists i{:}\mathsf{CityIndex}$$
$$\left(\begin{array}{l} \mathit{safeCity}[i] = \mathtt{undef} \wedge \mathit{safeCitytId} \neq \mathtt{undef} \wedge \mathit{safeCitytId}' = \mathtt{undef} \\ \wedge\ \mathit{safeCity}' = \lambda j. \left(\begin{array}{l} \text{if } j = i \text{ then } \mathit{safeCitytId} \\ \text{else if } \mathit{safeCity}[j] = \mathit{safeCitytId} \text{ then } \mathtt{undef} \\ \qquad \text{else } \mathit{safeCity}[j] \end{array}\right) \end{array}\right)$$

Note that the two previous transitions can be rewritten as a unique one, hence showing a more compact way of specifying RAS transitions. This, in turn, can augment the performance of the verifier while working with large-scale cases. The unified transition actually looks as follows:

$$\exists c{:}\mathsf{CityId}, \exists i{:}\mathsf{CityIndex}$$
$$\left(\begin{array}{l} c \neq \mathtt{undef} \wedge \mathit{safeCity}[i] = \mathtt{undef} \\ \wedge\ \mathit{safeCity}' = \lambda j. \left(\begin{array}{l} \text{if } j = i \text{ then } c \\ \text{else if } \mathit{safeCity}[j] = c \text{ then } \mathtt{undef} \\ \qquad \text{else } \mathit{safeCity}[j] \end{array}\right) \end{array}\right)$$

Then, to register passengers with booked tickets on a flight, the airline needs to make sure that a corresponding flight destination is actually safe. To perform the passenger registration, the airline selects a flight identifier that is assigned to the route and uses it to populate entries in an unary artifact relation $\mathit{regdPassenger} : \mathsf{PassengerIndex} \longrightarrow \mathsf{FlightId}$. Note that there may be more than one passenger taking the flight, and therefore, more than one entry in *regdPassenger* with the same flight identifier.

$$\exists i{:}\mathsf{CityIndex}, f{:}\mathsf{FlightId}, p{:}\mathsf{PassengerIndex}$$
$$\left(\begin{array}{l} f \neq \mathtt{undef} \wedge \mathit{destination}(f) = \mathit{safeCity}[i] \wedge \mathit{regdPassenger}[p] = \mathtt{undef} \\ \wedge\ \mathit{regdPassenger}' = \lambda j. \left(\begin{array}{l} \text{if } j = p \text{ then } f \\ \text{else } \mathit{regdPassenger}[j] \end{array}\right) \end{array}\right)$$

We also assume that the airline owns aircraft of one type that can contain no more than k passengers. In case there were more than k passengers registered on the flight, the airline receives a notification about its overbooking and temporary suspends all passenger registrations associated to this flight. This is modelled by checking whether there are at least $k+1$ entries in *regdPassenger*. If so, the flight identifier is added to a unary artifact relation $\mathit{overbooked} : \mathsf{FligthIndex} \longrightarrow \mathsf{FlightId}$ and all the passenger registrations in *regdPassenger* that reference this flight identifier are nullified by updating unboundedly many entries in the corresponding artifact relation:[10]

$$\exists p_1{:}\mathsf{PassengerIndex}, \ldots p_{k+1}{:}\mathsf{PassengerIndex}, m{:}\mathsf{FligthIndex}$$
$$\left(\begin{array}{l} \bigwedge_{i,i' \in \{1,\ldots,k+1\}, i \neq i'} \left(p_i \neq p_{i'} \wedge \mathit{regdPassenger}[p_i] \neq \mathtt{undef} \wedge \mathit{regdPassenger}[p_i] = \mathit{regdPassenger}[p_{i'}]\right) \\ \wedge\ \mathit{overbooked}[m] = \mathtt{undef} \\ \wedge\ \mathit{regdPassenger}' = \lambda j. \left(\begin{array}{l} \text{if } \mathit{regdPassenger}[j] = \mathit{regdPassenger}[p_1] \text{ then } \mathtt{undef} \\ \text{else } \mathit{regdPassenger}[j] \end{array}\right) \\ \wedge\ \mathit{overbooked}'[m] = \mathit{regdPassenger}[p_1] \end{array}\right)$$

Notice that this transition is not local, since its guard contains literals of the form $regdPassenger[p_i] = regdPassenger[p_{i'}]$ (with $p_i \neq p_{i'}$), which involve more than one element of one artifact sort.

In case of any contingency, the airport dispatcher may change the city status from *safe* to *unsafe*. To do it, we first select one of the safe cities, make it unsafe (i.e., remove it from *safeCity* relation) and store its identifier in the artifact variable *unsafeCityId*:

$$\exists i{:}\mathsf{CityIndex} \begin{pmatrix} unsafeCityId = \mathtt{undef} \wedge safeCity[i] \neq \mathtt{undef} \wedge \\ \wedge\ unsafeCityId' = safeCity[i] \wedge safeCity'[i] = \mathtt{undef} \end{pmatrix}$$

Then, we use the remembered city identifier to cancel all the passenger registrations for flights that use this city as their destination:

$$\begin{pmatrix} unsafeCityId \neq \mathtt{undef} \wedge unsafeCityId' = \mathtt{undef} \wedge \\ \wedge\ regdPassenger' = \lambda j. \begin{pmatrix} \text{if } destination(regdPassenger[j]) = unsafeCityId \text{ then } \mathtt{undef} \\ \text{else } regdPassenger[j] \end{pmatrix} \end{pmatrix}$$

Similarly to the previous case, this transition performs the intended action by updating unboundedly many entries in the artifact relation.

Also in this case, we can shrink the last two transitions into a single transition:

$$\exists i{:}\mathsf{CityIndex} \begin{pmatrix} safeCity[i] \neq \mathtt{undef} \wedge \\ \wedge regdPassenger' = \lambda j. \begin{pmatrix} \text{if } destination(regdPassenger[j]) = safeCity[i] \text{ then } \mathtt{undef} \\ \text{else } regdPassenger[j] \end{pmatrix} \end{pmatrix}$$

However, as in the previous case, the transition turns out to be not local. Specifically, it is due to the literal $destination(regdPassenger[j]) = safeCity[i]$ that involves more than one element of (different) artifact sorts.

6. First Experiments

We implemented a prototype of our backward reachability algorithm for artifact systems on top of the MCMT model checker, extending it with the features required to formalize and verify RASs. MCMT manages verification in the infinite-state case by exploiting as its model-theoretic framework the declarative formalism of array-based systems. Since their first introduction in Ghilardi et al. (2008a), Ghilardi and Ranise (2010a), array-based systems have been provided with various implementations of the standard backward reachability algorithms (including more sophisticated variants and heuristics). Starting from its first version (Ghilardi and Ranise 2010b), MCMT was successfully applied to cache coherence and mutual exclusions protocols (Ghilardi and Ranise 2010a), timed (Carioni et al. 2010), and fault-tolerant (Alberti et al. 2010, 2012b) distributed systems, and then to imperative programs (Alberti et al. 2014b, 2017); interesting case studies concerned waiting time bounds synthesis in parameterized timed networks (Bruttomesso et al. 2012) and internet protocols (Bruschi et al. 2017). Further related tools include SAFARI (Alberti et al. 2012a) and ASASP (Alberti et al. 2011); finally, Conchon et al. (2012, 2013, 2015, 2018a) implement the array-based setting on a parallel architecture with further powerful extensions.

The work principle of MCMT is rather simple: the tool generates the proof obligations arising from the safety and fixpoint tests in backward search (Lines 2–3 of Algorithm 1) and passes them to the background SMT-solver (currently it is YICES Dutertre and De Moura 2006). In practice, the situation is more complicated because SMT-solvers are quite efficient in handling satisfiability problems in combined theories at quantifier-free level but may encounter difficulties with

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

Table 1. Summary of the experimental examples

	Example	#(AC)	#(AV)	#(T)
E1	JobHiring	9	18	15
E2	Acquisition-following-RFQ	6	13	28
E3	Book-Writing-and-Publishing	4	14	13
E4	Customer-Quotation-Request	9	11	21
E5	Patient-Treatment-Collaboration	6	17	34
E6	Property-and-Casualty-Insurance-Claim-Processing	2	7	15
E7	Amazon-Fulfillment	2	28	38
E8	Incident-Management-as-Collaboration	3	20	19

quantifiers. For this reason, MCMT implements modules for *quantifier elimination* and *quantifier instantiation*. A *specific module* for the quantifier elimination problems mentioned in Line 6 of Algorithm 1 has been added to Version 2.8 of MCMT.

We base our experimental evaluation on the already existing benchmark provided in Li et al. (2017) that samples 32 real-world BPMN workflows published at the official BPM website (http://www.bpmn.org/). Specifically, we select seven examples of varying complexity (see Table 1) and provide their faithful encoding in the array-based specification using MCMT Version 2.8. Moreover, we enrich our experimental set with an extended version of the running example of this paper (see Appendix A.1 of Calvanese et al. 2018b). Each example has been checked against at least one safe and one unsafe conditions. Since MCMT performs safety verification parameterized on the read-only DB, the result is *independent* on the size of specific DB instances. Moreover, MCMT can in principle handle unbounded DB schemas and unboundedly many DB constants: we have ascertained that the size of the DB schema does not affect the performances as much as the number of transitions involved in the verified RAS . We leave for future work a systematic experimental evaluation of those preliminary observations.

Experiments were performed on a machine with Ubuntu 16.04, 2.6 GHz Intel Core i7 and 16 GB RAM. The benchmark set is available as part of the last distribution 2.8 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 (pp. 36–39) giving essential information on how to use the capabilities of the new version of MCMT (by activating the "db_driven" mode), how to encode RASs in MCMT specifications, and how to produce user-defined examples in the DB-driven framework. All the verified examples include transitions with quantified "data" variables and rely on and the algebraic framework of DB theories introduced in the paper. Consequently, the experiments were carried out on the new version of MCMT and could not be verified with the previous versions.

In Table 1, the columns #(AV), #(AC), and #(T) represent, respectively, the number of artifact variables, artifact components, and transitions used in the example specification; in Table 2, the column Time is the MCMT execution time. The most critical measures in Table 2 are #(N), depth and #(SMT-calls) that, respectively, define the number of nodes and the depth of the tree used for the backward reachability procedure adopted by MCMT, and the number of the SMT-solver calls. Indeed, MCMT computes the iterated preimages of the formula describing the unsafe states along the various transitions. Such computation produces a tree, whose nodes are labeled by formulae describing sets of states that can reach an unsafe state and whose arcs are labelled by a transition.

To stress test our encoding, we came up with a few formulae describing unsafe configurations (sets of "bad" states), that is, the configurations that the system should not incur throughout its execution. Property references encodings of examples endowed with specific (un)safety properties

Table 2. Experimental results for safety properties

Example	Property	Result	Time	#(N)	Depth	#(SMT-calls)
E1	E1P1	SAFE	0.06	3	3	1238
	E1P2	UNSAFE	0.36	46	10	2371
	E1P3	UNSAFE	0.50	62	11	2867
	E1P4	UNSAFE	0.35	42	10	2237
E2	E2P1	SAFE	0.72	50	9	3156
	E2P2	UNSAFE	0.88	87	10	4238
	E2P3	UNSAFE	1.01	92	9	4811
	E2P4	UNSAFE	0.83	80	9	4254
E3	E3P1	SAFE	0.05	1	1	700
	E3P2	UNSAFE	0.06	14	3	899
E4	E4P1	SAFE	0.12	14	6	1460
	E4P2	UNSAFE	0.13	18	8	1525
E5	E5P1	SAFE	4.11	57	9	5618
	E5P2	UNSAFE	0.17	13	3	2806
E6	E6P1	SAFE	0.04	7	4	512
	E6P2	UNSAFE	0.08	28	10	902
E7	E7P1	SAFE	1.00	43	7	5281
	E7P2	UNSAFE	0.20	7	4	3412
E8	E8P1	SAFE	0.70	77	11	3720
	E8P2	UNSAFE	0.15	25	7	1652

done in MCMT, whereas Result shows their verification outcome that can be of the two following types: SAFE and UNSAFE. The MCMT tool returns SAFE, if the undesirable property it was asked to verify represents a configuration that the system cannot reach. At the same time, the result is UNSAFE if there exists a path of the system execution that reaches "bad" states.

To conclude, we would like to point out that seemingly high number of SMT solver calls in #(SMT-calls) against relatively small execution time demonstrates that MCMT could be considered as a promising tool supporting the presented line of research. This is due to the following two reasons. On the one hand, the SMT technology underlying solvers like YICES (Dutertre and De Moura 2006) is quite mature and impressively well-performing. On the other hand, the backward reachability algorithm generates proof obligations which are relatively easy to be analyzed as (un)satisfiable by the solver.

A thorough comparison with VERIFAS (Li et al. 2017) is at the moment rather problematic, for various reasons, due, for instance, to the different specification languages and to the different set of benchmarks covered. In fact, the two systems tackle incomparable verification problems: on the one hand, we deal with safety problems, whereas VERIFAS handles more general LTL-FO properties. On the other hand, we tackle features not available in VERIFAS, like bulk updates and comparisons between artifact tuples. We leave this for future, more experimentally oriented, work: the comparison might be interesting because the two tools apply quite different technologies (VERIFAS is based on VASS encoding, whereas MCMT follows a purely declarative paradigm). We just point out that Table 2 shows the very encouraging results: in fact, MCMT seems to effectively handle the benchmark with a similar performance to that shown in other, well-established settings, with verification times below 1s in most cases.

7. Conclusions and Future Work

We have laid the foundations of SMT-based verification for artifact systems, focusing on safety problems and relying on array-based systems as underlying inspiring model. We have shown how to overcome the main technical difficulty arising from this approach, namely reconstructing quantifier elimination techniques in the rich setting of artifact systems, using the model-theoretic machinery of model completions. On top of this framework, we have identified three classes of systems for which safety is decidable, which impose different combinations of restrictions on the form of actions and the shape of DB constraints. The presented techniques have been implemented on top of the well-established MCMT model checker, making our approach fully operational. Notably, the machinery presented in this paper has been already employed to formalize and verify a data-aware extension of the de-facto process modeling standard BPMN (Calvanese et al. 2019a), starting a line of research that aims at transferring our technical results into practical, end user-oriented settings.

It is important to stress that the artifact systems we study here are radically different from other formal models integrating dynamics with data, such as Data Petri Nets (Lazic et al. 2008), ν-PNs (Rosa-Velardo and de Frutos-Escrig 2011), and multiset rewriting systems with data and constraints (Delzanno 2002). Let us consider Data Petri Nets as a representative example of this class of approaches. In Data Petri Nets, one can generate tokens that carry fresh values not already present in the current marking. The requirement that a value is fresh can be encoded in the model (Rosa-Velardo and de Frutos-Escrig 2011). Such values can only be mutually related using the comparison predicates over the underlying domain. In our setting, instead, the working memory of an artifact system may contain data elements arbitrarily taken from value sorts, or extracted from the (active domain of a) read-only DB. When loading a data element from a value sort, this may or not be present in the active domain, and thus it may be possibly fresh (but notice that freshness cannot be enforced in our model). When loading data elements from the read-only DB, it is crucial to consider that they are mutually related via constraints present therein. These constraints are primary keys, foreign keys, and additional axioms present in the DB theory. The read-only DB it is fixed within a run, but model checking of safety properties is studied parametrically with respect to all possible read-only DBs over a given schema. During model checking, we are examining sets of reachable states described by logical formulae, whose validity depends on properties that might happen to be true in the read-only DB, depending on the constraints present therein. To handle data elements coming from the read-only DB and their corresponding constraints, we therefore need a specialized machinery that is different from the one typically used to tame the infinity brought by freshness. In fact, it is not enough to embed the read-only DB in a larger model that admits fresh values to obtain quantifier elimination, which is essential in our model-checking algorithm. Quantifier elimination becomes available only when such a “larger model” possesses suitable model-theoretic properties, which we have studied in the paper. In particular, we have argued that such properties are captured by the well-known model-theoretic notion of existentially closed structure and its intimately related notion of model completion. Notably, resorting to model completion can be seen as the “most natural” way to obtain quantifier elimination, as it is the “closest” theory to the original one that at once admits quantifier elimination and preserves satisfiability of existential formulae. This is precisely what we intensively exploit in our model-checking algorithm.

From the *foundational* point of view, it is an open, non-trivial research question to see whether our framework and Data Petri nets (or similar approaches) can be inter-reduced to each other when we restrict our attention to the three decidable fragments studied in the paper. We are also interested in using the present contribution as the starting point for a full line of research dedicated to SMT-based techniques for the effective verification of data-aware processes, considering richer forms of verification going beyond safety, and richer classes of artifact systems incorporating concrete data types and arithmetic operations. We also intend to investigate more

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

integrity constraints used in database theory: this study should extend decidability and model-completability results beyond the cases covered in this paper. In addition, it would be interesting to study whether the decidable classes considered here are tight, or whether interesting variations can be found for which decidability is preserved, possibly guaranteeing termination of the our backward reachability procedure.

From the *algorithmic* point of view, we intend to develop more sophisticated techniques for the quantifier elimination module required in our backward reachability procedure. A first stepping stone in this direction, relying on a constrained version of the Superposition Calculus (SC) (Nieuwenhuis and Rubio 2001), can be found in Calvanese et al. (2019c): indeed, thanks to our constrained version of the SC, suitably combined with congruence closure, we show that it is possible to obtain a quadratic bound for the complexity of the quantifier elimination procedure in the case of interest for our applications (Calvanese et al. 2018a, 2019c).

As for *experiments*, we aim at building on the encouraging results reported here toward an extensive experimental evaluation of our approach, using the benchmark of Li et al. (2017) and the concrete specification language in Calvanese et al. (2019a) as a starting point. A natural next step is then to study how the computation of over-approximations, abstractions, and invariants (a capability that MCMT already supports but that should be adapted to the "db_driven" mode) and well-established techniques for SMT-based model checking like CEGAR (Alberti et al. 2014a; McMillan 2006) and IC3 (Bradley 2011, 2012; Hoder and Bjørner 2012) can be used to speed up the verification of artifact systems.

Financial Support. This research has been partially supported by the UNIBZ CRC project *REKAP: Reasoning and Enactment for Knowledge-Aware Processes* and by the CHIST-ERA project *PACMEL: Process-Aware Analytics Support Based on Conceptual Models for Event Logs.*

Notes

[1] We underline the fact that free n-ary relations (modeling plain relations without integrity constraints) can be added to our sorted unary functions framework without compromizing our model-checking techniques to apply. We shall briefly mention such trivial extensions in the paper (see, e.g., Definition 3.3).

[2] T-satisfiability and T^*-satisfiability are equivalent, by the definition of T^*, as far as existential (in particular, quantifier-free) formulae are concerned.

[3] For the test in line 3, we just need replace in ϕ the $\underline{x}$ by their values given by ι, conjoin the result with all the ground instances of the axioms of T, and finally decide satisfiability with congruence closure algorithm of a polynomial size ground conjunction of literals.

[4] By "signature" we always mean "signature with equality," so as soon as new sorts are added, the corresponding equality predicates are added too.

[5] Actually, τ is a disjunction of such formulae, but it is easily seen that disjunction can be accommodated by moving existential quantifiers back-and-forth through them.

[6] In the MCMT implementation, state formulae are always maintained so that all existential variables occurring in them are differentiated and there is no need of this expensive computation step.

[7] This is unrelated to cyclicity of Σ defined in Section 3 and comes from universal algebra terminology.

[8] This is not the canonical ordering used for multisets, as introduced, for example, in Baader and Nipkow (1998).

[9] Notice that the artifact setting described above to capture coverability problems for broadcast protocols is both local and tree-like.

[10] For simplicity of presentation, we simply remove such data from the artifact relation. In a real setting, this information would actually be transferred to a dedicated, historical table, so as to reconstruct the status of past, overbooked flights.

References

Abdulla, P. A., Cerans, K., Jonsson, B. and Tsay, Y.-K. (1996). General decidability theorems for infinite-state systems. In: *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS)*, 313–321.

Alberti, F., Armando, A. and Ranise, S. (2011). ASASP: automated symbolic analysis of security policies. In: *Proceedings of the 23rd International Conference on Automated Deduction (CADE)*, LNCS (LNAI), vol. 6803, Springer, 26–33.

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

Alberti, F., Bruttomesso, R., Ghilardi, S., Ranise, S. and Sharygina, N. (2012a). SAFARI: SMT-based abstraction for arrays with interpolants. In: *Proceedings of the 24th International Conference on Computer Aided Verification (CAV)*, LNCS, vol. 7358, Springer, 679–685.

Alberti, F., Bruttomesso, R., Ghilardi, S., Ranise, S. and Sharygina, N. (2014a). An extension of lazy abstraction with interpolation for programs with arrays. *Formal Methods in System Design* **45** (1) 63–109.

Alberti, F., Ghilardi, S., Pagani, E., Ranise, S. and Rossi, G. P. (2010). Brief announcement: automated support for the design and validation of fault tolerant parameterized systems - A case study. In: *Proceeding of 24th International Symposium on Distributed Computing DISC*, LNCS, vol. 6343, Springer, 392–394.

Alberti, F., Ghilardi, S., Pagani, E., Ranise, S. and Rossi, G. P. (2012b). Universal guards, relativization of quantifiers, and failure models in Model Checking Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation* **8** (1/2) 29–61.

Alberti, F., Ghilardi, S. and Sharygina, N. (2014b). Booster: an acceleration-based verification framework for array programs. In: *Proceedings of the 12th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, LNCS, vol. 8837, Springer, 18–23.

Alberti, F., Ghilardi, S. and Sharygina, N. (2017). A framework for the verification of parameterized infinite-state systems. *Fundamenta Informaticae* **150** (1) 1–24.

Baader, F., Ghilardi, S. and Tinelli, C. (2006). A new combination procedure for the word problem that generalizes fusion decidability results in modal logics. *Information and Computation* **204** (10) 1413–1452.

Baader, F. and Nipkow, T. (1998). *Term Rewriting and All That*, Cambridge University Press.

Bojańczyk, M., Segoufin, L. and Toruńczyk, S. (2013). Verification of database-driven systems via amalgamation. In: *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 63–74.

Bradley, A. R. (2011). SAT-based model checking without unrolling. In: *Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation VMCAI*, LNCS, vol. 6538, Springer, 70–87.

Bradley, A. R. (2012). IC3 and beyond: incremental, inductive verification. In: *Proceedings of the 24th International Conference on Computer Aided Verification (CAV)*, LNCS, vol. 7358, Springer, 4.

Bradley, A. R. and Manna, Z. (2007). *The Calculus of Computation - Decision Procedures with Applications to Verification*, Springer.

Bruschi, D., Di Pasquale, A., Ghilardi, S., Lanzi, A. and Pagani, E. (2017). Formal verification of ARP (address resolution protocol) through SMT-based model checking - A case study. In: *Proceedings of the 13th International Conference on Integrated Formal Methods (IFM)*, LNCS, vol. 10510, Springer, 391–406.

Bruttomesso, R., Carioni, A., Ghilardi, S. and Ranise, S. (2012). Automated analysis of parametric timing-based mutual exclusion algorithms. In: *Proceedings of the 4th International Symposium on NASA Formal Methods (NFM)*, LNCS, vol. 7226, Springer, 279–294.

Calvanese, D., De Giacomo, G. and Montali, M. (2013). Foundations of data-aware process analysis: a database theory perspective. In: *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 1–12.

Calvanese, D., Ghilardi, S., Gianola, A., Montali, M. and Rivkin, A. (2018a). *Quantifier Elimination for Database Driven Verification*. Technical Report arXiv:1806.09686, arXiv.org.

Calvanese, D., Ghilardi, S., Gianola, A., Montali, M. and Rivkin, A. (2018b). *Verification of Data-Aware Processes via Array-Based Systems (Extended Version)*. Technical Report arXiv:1806.11459, arXiv.org.

Calvanese, D., Ghilardi, S., Gianola, A., Montali, M. and Rivkin, A. (2019a). Formal modeling and SMT-based parameterized verification of data-aware BPMN. In: *Proceeding of the 17th International Conference on Business Process Management (BPM)*, LNCS, vol. 11675, Springer, 157–175.

Calvanese, D., Ghilardi, S., Gianola, A., Montali, M. and Rivkin, A. (2019b). From model completeness to verification of data aware processes. In: Lutz C., Sattler U., Tinelli C., Turhan A.-Y., and Wolter F. (eds.) *Description Logic, Theory Combination, and All That*, LNCS, vol. 11560, Springer, 212–239.

Calvanese, D., Ghilardi, S., Gianola, A., Montali, M. and Rivkin, A. (2019c). Model completeness, covers and superposition. In: *Proceedings of 27th International Conference on Automated Deduction (CADE)*, LNCS (LNAI), vol. 11716, Springer, 142–160.

Calvanese, D., Ghilardi, S., Gianola, A., Montali, M. and Rivkin, A. (2019d). 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)*, EPTCS, vol. 311.

Carioni, A., Ghilardi, S. and Ranise, S. (2010). MCMT in the land of parametrized timed automata. In: *Proceedings of the 6th International Verification Workshop (VERIFY)*, 47–64.

Chang, C.-C. and Keisler, J. H. (1990). *Model Theory*. North-Holland Publishing Co.

Cimatti, A., Stojic, I. and Tonetta, S. (2018). Formal specification and verification of dynamic parametrized architectures. In: *Proceedings of the 22nd International Symposium on Formal Methods (FM)*, LNCS, vol. 10951, Springer, 625–644.

Conchon, S., Declerck, D. and Zaidi, F. (2018a). Cubicle-W: parameterized model checking on weak memory. In: *Proceedings of the 9th International Joint Conference on Automated Reasoning (IJCAR)*, LNCS (LNAI), vol. 10900, Springer, 152–160.

https://doi.org/10.1017/S0960129520000067 Published online by Cambridge University Press

Conchon, S., Delzanno, G. and Ferrando, A. (2018b). Declarative parameterized verification of topology-sensitive distributed protocols. In: *Proceedings of the 6th International Conference on Networked Systems (NETYS)*, LNCS, vol. 11028, Springer, 209–224.

Conchon, S., Goel, A., Krstic, S., Mebsout, A. and Zaïdi, F. (2012). Cubicle: a parallel SMT-based model checker for parameterized systems - Tool paper. In: *Proceedings of the 24th International Conference on Computer Aided Verification (CAV)*, LNCS, vol. 7358, Springer, 718–724.

Conchon, S., Goel, A., Krstic, S., Mebsout, A. and Zaïdi, F. (2013). Invariants for finite instances and beyond. In: *Proceedings of the International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, 61–68.

Conchon, S., Mebsout, A. and Zaïdi, F. (2015). Certificates for parameterized model checking. In: *Proceeding of the 20th International Symposium on Formal Methods (FM)*, LNCS, vol. 9109, Springer, 126–142.

Damaggio, E., Deutsch, A. and Vianu, V. (2012). Artifact systems with data dependencies and arithmetic. *ACM Transactions on Database Systems* **37** (3) 22:1–22:36.

Delzanno, G. (2002). An overview of MSR(C): a CLP-based framework for the symbolic verification of parameterized concurrent systems. *Electronic Notes in Theoretical Computer Science* **76** 65–82.

Delzanno, G. (2018). Parameterized verification of publish/subcribe protocols via infinite-state model checking. In: *Proceedings of the 33rd Italian Conference on Computational Logic (CILC)*, 97–111.

Delzanno, G., Esparza, J. and Podelski, A. (1999). Constraint-based analysis of broadcast protocols. In: *Proceeding of 13th International Workshop on Computer Science Logic (CSL)*, LNCS, vol. 1683, Springer, 50–66.

Deutsch, A., Hull, R., Li, Y. and Vianu, V. (2018). Automatic verification of database-centric systems. *SIGLOG News* **5** (2) 37–56.

Deutsch, A., Hull, R., Patrizi, F. and Vianu, V. (2009). Automatic verification of data-centric business processes. In: *Proceedings of the 12th International Conference on Database Theory (ICDT)*, 252–267.

Deutsch, A., Li, Y. and Vianu, V. (2016). Verification of hierarchical artifact systems. In: *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, 179–194.

Deutsch, A., Li, Y. and Vianu, V. (2019). Verification of hierarchical artifact systems. *ACM Transactions on Database Systems* **44** (3) 12:1–12:68.

Dutertre, B. and De Moura, L. (2006). *The YICES SMT Solver*. Technical Report, SRI International.

Esparza, J., Finkel, A. and Mayr, R. (1999). On the verification of broadcast protocols. In: *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS)*, 352–359.

Fagin, R. (1976). Probabilities on finite models. *Journal of Symbolic Logic* **41** (1) 50–58.

Ghilardi, S. (2004). Model theoretic methods in combined constraint satisfiability. *Journal of Automated Reasoning* **33** (3–4) 221–249.

Ghilardi, S. and Gianola, A. (2017). Interpolation, amalgamation and combination (the non-disjoint signatures case). In: *Proceedings of the 11th International Symposium on Frontiers of Combining Systems (FroCoS)*, LNCS (LNAI), vol. 10483, Springer, 316–332.

Ghilardi, S. and Gianola, A. (2018). Modularity results for interpolation, amalgamation and superamalgamation. *Annals of Pure and Applied Logic* **169** (8) 731–754.

Ghilardi, S., Nicolini, E., Ranise, S. and Zucchelli, D. (2008a). Towards SMT model checking of array-based systems. In: *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR)*, LNCS (LNAI), vol. 5195, Springer, 67–82.

Ghilardi, S., Nicolini, E. and Zucchelli, D. (2008b). A comprehensive framework for combined decision procedures. *ACM Transactions on Computational Logic* **9** (2) 8:1–8:54.

Ghilardi, S. and Ranise, S. (2010a). Backward reachability of array-based systems by SMT solving: termination and invariant synthesis. *Logical Methods in Computer Science* **6** (4).

Ghilardi, S. and Ranise, S. (2010b). MCMT: a model checker modulo theories. In: *Proceedings of the 5th International Joint Conference on Automated Reasoning (IJCAR)*, LNCS (LNAI), vol. 6173, Springer, 22–29.

Ghilardi, S. and van Gool, S. J. (2016). Monadic second order logic as the model companion of temporal logic. In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 417–426.

Ghilardi, S. and van Gool, S. J. (2017). A model-theoretic characterization of monadic second order logic on infinite words. *Journal of Symbolic Logic* **82** (1) 62–76.

Higman, G. (1952). Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society* **3** (2) 326–336.

Hoder, K. and Bjørner, N. (2012). Generalized property directed reachability. In: *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, LNCS, vol. 7317, Springer, 157–171.

Hull, R. (2008). Artifact-centric business process models: brief survey of research results and challenges. In: *Proceedings of the OTM Confederated International Conferences*, LNCS, vol. 5332, Springer, 1152–1163.

Kruskal, J. B. (1960). Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society* **95** 210–225.

Lazic, R., Newcomb, T. C., Ouaknine, J., Roscoe, A. W. and Worrell, J. (2008). Nets with tokens which carry data. *Fundamenta Informaticae* **88** (3) 251–274.

Li, Y., Deutsch, A. and Vianu, V. (2017). VERIFAS: a practical verifier for artifact systems. *Proceedings of the VLDB Endowment* **11** (3) 283–296.

Lipparini, P. (1982). 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.

McMillan, K. L. (2006). Lazy abstraction with interpolants. In: *Proceedings of the 18th International Conference on Computer Aided Verification (CAV)*, LNCS, vol. 4144, Springer, 123–136.

Nicolini, E., Ringeissen, C. and Rusinowitch, M. (2009a). Data structures with arithmetic constraints: a non-disjoint combination. In: *Proceedings of the 7th International Symposium on Frontiers of Combining Systems (FroCoS)*, LNCS (LNAI), vol. 5749, Springer, 319–334.

Nicolini, E., Ringeissen, C. and Rusinowitch, M. (2009b). Satisfiability procedures for combination of theories sharing integer offsets. In: *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS, vol. 5505, Springer, 428–442.

Nicolini, E., Ringeissen, C. and Rusinowitch, M. (2010). Combining satisfiability procedures for unions of theories with a shared counting operator. *Fundamenta Informaticae* **105** (1–2) 163–187.

Nieuwenhuis, R. and Rubio, A. (2001). Paramodulation-based theorem proving. In: Robinson J. A., and Voronkov A. (eds.) *Handbook of Automated Reasoning (in 2 Volumes)*, MIT Press, 371–443.

Rado, R. (1964). Universal graphs and universal functions. *Acta Arithmetica* **9** 331–340.

Robinson, A. (1951). *On the Metamathematics of Algebra*, North-Holland.

Robinson, A. (1963). *Introduction to Model Theory and to the Metamathematics of Algebra*, Studies in Logic and the Foundations of Mathematics, North-Holland.

Rosa-Velardo, F. and de Frutos-Escrig, D. (2011). Decidability and complexity of Petri nets with unordered data. *Theoretical Computer Science* **412** (34) 4439–4451.

Schmitz, S. and Schnoebelen, P. (2013). The power of well-structured systems. In: *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR)*, LNCS, vol. 8052, Springer, 5–24.

Silver, B. (2011). *BPMN Method and Style*, 2nd edn., Cody-Cassidy.

Sofronie-Stokkermans, V. (2008). Interpolation in local theory extensions. *Logical Methods in Computer Science* **4** (4).

Sofronie-Stokkermans, V. (2016). On interpolation and symbol elimination in theory extensions. In: *Proceedings of the 8th International Joint Conference on Automated Reasoning (IJCAR)*, LNCS (LNAI), vol. 9706, Springer, 273–289.

Sofronie-Stokkermans, V. (2018). On interpolation and symbol elimination in theory extensions. *Logical Methods in Computer Science* **14** (3).

Vianu, V. (2009). Automatic verification of database-driven systems: a new frontier. In: *Proceedings of the 12th International Conference on Database Theory (ICDT)*, 1–13.

Wheeler, W. H. (1976). Model-companions and definability in existentially complete structures. *Israel Journal of Mathematics* **25** (3–4) 305–330.

Cite this article: Calvanese D, Ghilardi S, Gianola A, Montali M and Rivkin A (2020). SMT-based verification of data-aware processes: a model-theoretic approach. *Mathematical Structures in Computer Science* **30**, 271–313. https://doi.org/10.1017/S0960129520000067