Safety Verification and Universal Invariants for Relational Action Bases

Silvio Ghilardi[1] , **Alessandro Gianola**[2] , **Marco Montali**[2] and **Andrey Rivkin**[3]

[1]Dipartimento di Matematica, Università degli Studi di Milano, Milan, Italy
[2]Faculty of Engineering, Free University of Bozen-Bolzano, Bolzano, Italy
[3]DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark
silvio.ghilardi@unimi.it, {gianola,montali}@inf.unibz.it, ariv@dtu.dk

Abstract

Modeling and verification of dynamic systems operating over a relational representation of states are increasingly investigated problems in AI, Business Process Management and Database Theory. To make these systems amenable to verification, the amount of information stored in each state needs to be bounded, or restrictions are imposed on the preconditions and effects of actions. We lift these restrictions by introducing the framework of relational action bases (RABs), which generalizes existing frameworks and in which unbounded relational states are evolved through actions that can *(1)* quantify both existentially and universally over the data, and *(2)* use arithmetic constraints.We then study parameterized safety of RABs via (approximated) SMT-based backward search, singling out essential meta-properties of the resulting procedure, and showing how it can be realized by an off-the-shelf combination of existing verification modules of the state-of-the-art MCMT model checker. We demonstrate the effectiveness of this approach on a benchmark of data-aware business processes. Finally, we show how universal invariants can be exploited to make this procedure fully correct.

1 Introduction

Reasoning about actions and processes has witnessed an important shift in the representation of states and the way actions query and progress them. Relational representations and query/update languages are often used in these systems as a convenient syntactic sugar to compactly represent and evolve propositional states - see, e.g., STRIPS planning. More recently, instead, several works have put emphasis on states captured by full-fledged relational structures, equipping dynamic systems with actions that can create and destroy objects and relations. This *data-awareness* is essential to capture relevant systems in AI [Baral and De Giacomo, 2015] and business process management [Vianu, 2009; Calvanese *et al.*, 2013]. Notable examples are: *(i)* Situation Calculus [De Giacomo *et al.*, 2016] and knowledge-based [Bagheri Hariri *et al.*, 2013b] action theories, *(ii)* relational MDPs [Yang *et al.*, 2022], *(iii)* data-/artifact-centric business processes [Bagheri Hariri *et al.*, 2013a; Deutsch *et al.*, 2016; Calvanese *et al.*, 2019], and *(iv)* Petri nets with identifiers [Polyvyanyy *et al.*, 2019; Ghilardi *et al.*, 2022a] or with advanced interaction mechanisms [Fahland, 2019]. The subtle interplay between, data and processes calls for dedicated techniques for verifying at design-time whether such *relational dynamic systems* behave as expected [Baral and De Giacomo, 2015; Calvanese *et al.*, 2013]. However, verification is extremely difficult, as relational dynamic systems induce infinite state-spaces. This poses two challenges: *foundationally*, the identification of interesting classes of systems for which suitable verification procedures enjoy key (meta-)properties like correctness and termination; *practically*, the development of corresponding efficient verification tools. Typically, this leads to data-aware dynamic systems severely controlling the expressiveness of the data component and/or of the way actions operate over data, resulting in decidable/tractable fragments that are however too restrictive in capturing real-life processes.

In this work, we follow a different route. Modeling-wise, we start from one of the most expressive data-aware process modeling frameworks existing in the literature [Calvanese *et al.*, 2020], and we further enrich it with two essential features toward practical applicability. Such feature have never been combined in a unique formalism so far, and are *arithmetics*, to express conditions over and manipulate the content of numerical data values, and *universally-quantified guards*, to capture advanced forms of synchronization in concurrent and multi-agent systems and rich conditional updates in complex relational dynamic systems (see Section 2 for more details). While decidability does not hold in general for the resulting framework, we provide an algorithmic approach for verification that enjoys interesting meta-properties such (partial) soundness and completeness, and also demonstrate the benefit of injecting universal safety invariants to avoid spurious results. Computation-wise, we show that practical verification can be conducted through a careful combination of existing verification modules implemented in the MCTM model checker [Ghilardi and Ranise, 2010b]. We then demonstrate feasibility through an extensive experimental evaluation conducted on a suitably extended version of the only existing formal modeling and verification benchmark for data-aware processes [Gianola, 2022]. This extended version is of independent interest, as the research community lacks a comprehen-

sive, shared collection of realistic data-aware process models. Notably, all examples in the benchmark specify the control-flow component of the process through the BPMN language, which is the de-facto standard for the representation of business processes.

Technical contributions. To attack the verification of relational dynamic systems, we start from RASs [Calvanese *et al.*, 2020], one of the most expressive models in the literature. A RAS comes with *(i)* a read-only database with primary and foreign keys, *(ii)* a working memory consisting of read-write unbounded relations, and *(iii)* actions for constrained updates, whose preconditions query the working and read-only memory with existential queries, and whose effects consist in update formulae that can express addition, deletion, or bulk update of tuples in the read-write relations.

Our first contribution is to further push the boundaries of expressiveness of RASs, extending them with two genuinely novel essential features: *universal guards* and *arithmetics*. In the resulting framework of *relational action bases (RABs)*, unbounded relational states are evolved through actions that quantify existentially and universally over the data, and that use numerical datatypes with arithmetic predicates (including linear arithmetics for integers and reals). As surveyed later, these features match needs in different application domains.

On top of this framework, we provide a threefold technical contribution. The first concerns algorithmic techniques for verification of RABs. Specifically, we build on the last version of the MCMT model checker [Ghilardi and Ranise, 2010b], a natural choice for two reasons. On the one hand, MCMT comes with an effective backward reachability procedure that handles the automated verification of safety properties over RASs, ascertained parametrically to the content of the read-only database [Calvanese *et al.*, 2020; Calvanese *et al.*, 2021]. On the other hand, MCMT includes two separate modules to respectively handle universal quantifiers via dynamic forms of *instantiation* [Alberti *et al.*, 2012; Feldman *et al.*, 2019], and deal with the elimination of data variables via *cover computation* [Calvanese *et al.*, 2021], combined with quantifier elimination for arithmetics [Calvanese *et al.*, 2022a]. While such modules were so far employed in isolation, we here show for the first time that they can be gracefully combined, both theoretically and practically, inside the backward search of MCMT, enabling verification of (parameterized) safety in the novel setting of RABs. This is surprising since these two techniques may appear to interfere with each other, while we show that they interact harmlessly. The integration of these two techniques results in two (over-)approximation steps in the symbolic computation of unsafe states, which could cause the resulting procedure to detect unsafety spuriously. We thus investigate meta-properties of the procedure; decidable cases for which termination is guaranteed are inherited from the RAS fragments from [Calvanese *et al.*, 2020]. This shows that MCMT can handle RABs off-the-shelf, without destroying the meta-properties of the combined procedure.

As a second technical contribution, we put MCMT at work: we show that it terminates producing a correct answer with very good performance, on a set of 34 RABs that we derive from a benchmark of data-aware processes [Li *et al.*, 2017].

Last, we study the injection on invariants in the safety analysis of RABs. We obtain the key result that under the existence of universal invariants, spurious results for unsafety are never produced. This requires to prove, using model theory, that the two approximation techniques do not interfere with each other, but again combine well. Notably, our invariance result is analogous, in spirit, to that in [Karbyshev *et al.*, 2017], but substantially differs from the technical point of view: it holds for a different verification procedure and covers the whole expressiveness of RABs, out of reach so far.

The paper is organized as follows. In Section 2, we discuss the related work and the motivations for the need of new features of RABs. After the preliminaries, in Section 4 we introduce the general model of RABs and we describe an informative RAB example inspired by a process for evaluating visa applications, where we concretely present the novel modeling features. In Section 5, we introduce the SMT-based verification procedure for checking safety of RABs and we show its meta-properties. In Section 6, we prove how universal invariants can be exploited to avoid spurious unsafety outcomes, and in Section 7 we experimentally test our approach against the only existing verification benchmark for data-aware processes. We conclude in Section 8. Full proofs and details are provided in the online extended version [Ghilardi *et al.*, 2022b].

2 Motivation and Related Work

We discuss related work with two goals: *(i)* motivate the need of arithmetics and universal quantification when modeling relational dynamic systems in different application domains; *(ii)* discuss how RABs relate to the state-of-the-art. As the importance of arithmetics is well-known (see, e.g., [Gerevini *et al.*, 2008; Belardinelli, 2014; Deutsch *et al.*, 2016; Felli *et al.*, 2022]), we concentrate on universal quantification, discussing that it is essential to model the injection of fresh objects, capture conditional updates, deal with constraint checking, and capture synchronization in concurrent systems.

Injection of fresh objects. Relational dynamic systems need to create new, fresh objects during the execution. Classical examples are the creation of a new primary key within a database [Bagheri Hariri *et al.*, 2013a; Belardinelli *et al.*, 2014], or of a new identifier in a (high-level) Petri net [Polyvyanyy *et al.*, 2019; Ghilardi *et al.*, 2022a]. Without universal quantification, freshness cannot be guaranteed: it is possible to nondeterministically pick an object, but not to enforce that it is not contained in the working memory.

Conditional updates and constraint checking. Universal quantification is need to express rich conditional updates operating at once over the entire extension of a relation [Bagheri Hariri *et al.*, 2013a; Belardinelli *et al.*, 2014]. Conditional updates can also be used to enforce constraints on the working memory of the system, using the following modelling pattern. Considering the forms of actions supported by RABs, the constraint of interest can be formulated as a universally quantified sentence. The system alternates an action mode and a check mode. In the check mode, the system verifies whether the constraint holds. If so, the system goes back

to the action mode. If not (i.e., the negation of the constraint holds), the system enters into an error state. Universally quantified sentences can express, key and disjointness constraints.

Universal synchronization. Dynamic systems with multiple concurrent entities call for synchronization mechanisms. Two key scenarios are (parameterized) multiagent systems, where *all* agents synchronously perform a joint action [Kouvaros and Lomuscio, 2016; Felli *et al.*, 2020], and equality synchronization in proclets [Fahland, 2019], where a transition can be performed by a parent entity only if *all* its children are in a certain state (e.g., all same-order items have been validated). With universally quantified variables, RABs support such mechanisms, allowing for verification of proclets that has been out of reach [Fahland, 2019; Ghilardi *et al.*, 2022a].

Related work. Verification of relational dynamic systems has been addressed along two main lines. In the first, a bound is imposed on the number of objects that can be stored in a single state. This makes verification for first-order μ-calculus [Bagheri Hariri *et al.*, 2014; Calvanese *et al.*, 2018] and a fragment of first-order LTL [Calvanese *et al.*, 2022b] reducible to finite-state model checking, for systems with a fixed initial state and objects only equipped with equality comparison. This carries over relational multiagent systems verified against epistemic first-order CTL [Belardinelli *et al.*, 2014]. These techniques do not correspond to concrete verification tools, and only preliminary results exist on implementations [Yang *et al.*, 2022].

The second line of research studies relational dynamic systems operating over a working memory that can contain unboundedly many objects per state, and accessing read-only data. Verification is studied parametrically to the read-only storage, ensuring that desired properties hold irrespectively of its content. This setting, surveyed in [Vianu, 2009], is more delicate than the one of state-bounded systems: the identification of verifiable fragments calls for a very careful analysis of modelling features supported in action specifications. At the same time, perhaps surprisingly, there are practical verifiers working with unbounded data, based on explicit-state model checking and vector addition systems [Li *et al.*, 2017], or on symbolic model checking via SMT [Calvanese *et al.*, 2019] implemented in a dedicated module of the MCMT model checker [Ghilardi and Ranise, 2010b]. Other SMT-based approaches dealing with universal quantification and parameterized verification, such as [Gurfinkel *et al.*, 2016; Cimatti *et al.*, 2021], cannot instead be readily applied, as they do not natively handle *relational* dynamic systems.

3 Preliminaries

We adopt the usual first-order syntactic notions of signature, term, formula, and the like. Signatures are multi-sorted. Every sort comes with equality. For simplicity, most definitions are given for single-sorted languages - the adaptation to multi-sorted is straightforward. Notation $t(\underline{x})$ (resp., $\phi(\underline{x})$) means that term t (resp., formula ϕ) has free variables included in $\underline{x}$, where $\underline{x}$ represents a tuple $\langle x_1, \ldots, x_n \rangle$ of variables. We assume that terms and formulae are well-typed. A formula is *universal* (resp., *existential*) if it has the form $\forall \underline{x}(\phi(\underline{x}))$ (resp., $\exists \underline{x}(\phi(\underline{x}))$), where ϕ is a quantifier-free formula. A *sentence* is a formula without free variables.

For semantics, we use the standard notions of 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. $T \models \phi$ expresses that ϕ is true in all models of T for every assignment to the variables occurring free in ϕ. ϕ is T-*satisfiable* iff there exists a model $\mathcal{M}$ of T and an assignment to the free variables of ϕ that makes ϕ true in $\mathcal{M}$: if ϕ is quantifier-free, the problem of establishing for ϕ the existence of such a model and an assignment is called *SMT problem* for T. Examples of theories from the SMT literature are $\mathcal{EUF}$, the theory of equality with uninterpreted symbols, and $\mathcal{LIA}/\mathcal{LRA}$, the theory of linear integer/real arithmetics. A T-cover of a formula $\exists \underline{x} \phi(\underline{x}, \underline{y})$ is a formula $\psi(\underline{y})$ that is implied by $\exists \underline{x} \phi(\underline{x}, \underline{y})$ and implies all the other implied formulae $\phi'(\underline{z}, \underline{y})$ modulo T. A theory T has *covers* iff all formulae $\exists \underline{x} \phi(\underline{x}, \underline{y})$ have a T-cover, and an effective procedure for computing them is available. Computing T-covers is strictly related to the problem of eliminating quantifiers in suitable theory extensions of T [Calvanese *et al.*, 2021].

We introduce case-defined functions. Fix a Σ-theory T; a T-*partition* is a finite set $\kappa_1(\underline{x}), \ldots, \kappa_n(\underline{x})$ of quantifier-free formulae s. t. $T \models \forall \underline{x} \bigvee_{i=1}^{n} \kappa_i(\underline{x})$ 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 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 $F \notin \Sigma$, and $T' = T \cup \bigcup_{i=1}^{n} \{\forall \underline{x} \ (\kappa_i(\underline{x}) \rightarrow F(\underline{x}) = t_i(\underline{x}))\}$. Intuitively, F represents a case-defined function using nested if-then-else expressions as: $F(\underline{x}) := \texttt{case of } \{\kappa_1(\underline{x}) : t_1; \cdots; \kappa_n(\underline{x}) : t_n\}$. We identify T with any of its case-definable extensions T'. In fact, given a Σ'-formula ϕ', one can easily find a Σ-formula ϕ that is equivalent to ϕ' in all models of T'. We also use specific λ-abstractions and abbreviate FO formulae of the form $\forall y.\ b(y) = F(y, \underline{z})$ (where, typically, F is a symbol introduced in a case-defined extension) into $b = \lambda y.F(y, \underline{z})$.

4 Relational Action Bases

Now we introduce the formalism of *relational action bases* (RABs), following the widely adopted framework of artifact systems [Hull, 2008], in their most general form structured in three components [Deutsch *et al.*, 2016]: *(i)* a read-only relational database (DB) with primary and foreign keys, to store background, static information; *(ii)* a mutable working memory consisting of a set of evolving relations; *(iii)* a set of guarded transitions that inspects the DB and the working memory, and updates the latter. RABs actually take the RAS model of [Calvanese *et al.*, 2020] (which supports the most expressive forms of guarded transitions), with two essential extensions: full-fledged arithmetic theories, and universal quantification in the guards and effects of transitions.

4.1 Static DB Schemas

Static DB schemas define read-only relations with primary and foreign key constraints, which host pure identifiers sub-

ject to $\mathcal{EUF}$, or integer/real data attributes subject to arithmetic constraints expressed in $\mathcal{LIA}$/$\mathcal{LRA}$.

Definition 1. *A* static DB schema *(*SDB *schema for short) is a pair* $DB := \langle \Sigma^{DB} \cup \Sigma^{ar}, T^{DB} \cup T^{ar} \rangle$*, where:*

- Σ^{DB}*, called* SDB signature*, is a finite multi-sorted signature (where sorts are partitioned into* id *and* value *sorts* $\Sigma^{DB}_{srt} = \Sigma^{DB}_{ids} \uplus \Sigma^{DB}_{val}$*), whose symbols are equality, unary functions, n-ary relations and constants;*
- T^{DB}*, called* SDB theory*, is* $\mathcal{EUF}(\Sigma^{DB}) \cup \{\forall x\ (x = \mathsf{undef} \leftrightarrow f(x) = \mathsf{undef})\}$*, for every function f in* Σ^{DB}*;*
- Σ^{ar}*, called* arithmetic signature*, is the signature of $\mathcal{LRA}$ or of $\mathcal{LIA}$, and its sorts are* Σ^{ar}_{srt}*;*
- T^{ar}*, called* arithmetic theory*, is $\mathcal{LRA}$ or $\mathcal{LIA}$.*
- $\Sigma^{DB}_{val} \cup \Sigma^{ar}_{srt}$ *can only be the codomain sort of a symbol from* Σ^{DB} *other than an equality predicate;*

We respectively call $\Sigma := \Sigma^{DB} \cup \Sigma^{ar}$ *and* $T := T^{DB} \cup T^{ar}$ *the* full signature *and the* full theory *of* DB*.*

The definition employs an unusual functional approach for modeling DB schemas (needed for the technical machinery of safety checking procedures), and captures the most sophisticated read-only DB schemas considered in the literature [Deutsch *et al.*, 2019; Calvanese *et al.*, 2019]. Unary functions capture relations with primary and foreign keys. Specifically, the domain of a unary function is an *id sort*, representing object identifiers for that sort. Functions sharing the same id sort as domain model the attributes of such objects, which either point to other id sorts (implicitly representing foreign keys), or to so-called *value sorts* that denote primitive datatypes. While in previous works only uninterpreted value sorts could be employed, here we also support real/integer datatypes, subject to arithmetic theories that considerably increases the modeling power of the language.

We use a special undef constant to model NULL-like objects/values in the working memory, and an axiom $\forall x\ (x = \mathsf{undef} \leftrightarrow f(x) = \mathsf{undef})$ to indicate that application of a function to undef returns an undefined value/object and that this is the only case for which the function is undefined.

Definition 2. *An* SDB instance *of* $DB := (\Sigma, T)$ *with* $\Sigma := \Sigma^{DB} \cup \Sigma^{ar}$ *is a Σ-structure $\mathcal{M}$ that is a model of T and s.t. every id sort of* Σ^{DB} *is interpreted in $\mathcal{M}$ on a* finite *set.*

There is a key difference between SDB instances and arbitrary *models* of $T^{DB} \cup T^{ar}$: finiteness of id sorts and of the non-id values that can be pointed from id sorts using functions. This is customary for DBs [Abiteboul *et al.*, 1995]. Notice, T^{DB} has the finite model property for constraint satisfiability, so the SMT problem can be equivalently reframed by asking for the existence of an SDB instance instead of a generic T^{DB} model [Calvanese *et al.*, 2020].

Example 1. *Consider a visa application center, with a read-only DB storing information critical to the application process. We formalize this in a DB signature Σ_v with: (i) one id sort to identify citizens; (ii) two value sorts INT and STRING, used for evaluating applications (e.g., by giving scores).*

4.2 RAB Transitions

The working memory of a RAB consists of *individual* and *function* variables. Function variables model evolving relations, in the style of [Li *et al.*, 2017], while individual variables are used both to store global information about the control state of the process, and to load and manipulate (components of) tuples from the static relations.

Given an SDB schema $\langle \Sigma, T \rangle$, a *(working) memory extension* of Σ is a signature Σ_{ext} obtained by adding to Σ extra sort symbols $E, E_1, E_2 \ldots$, called *memory sorts*, together with corresponding equality predicates. Now, the "old" sorts from Σ are called *basic* (variables of basic sorts we call 'basic sort' variables). This is done to model the mutable working memory similarly to the static DB: implicit identifiers of working memory tuples form working memory relations, whereas data values of such tuples have basic sorts.

A *memory schema* is a pair $(\underline{x}, \underline{a})$ of individual and unary function variables. Variables in $\underline{x}$ are called *memory variables*, and the ones in $\underline{a}$ – *memory components*. The latter are required to have a memory sort as source sort and a basic sort as target sort. Given an SDB instance $\mathcal{M}$ of Σ_{ext}, an *assignment* to a memory schema $(\underline{x}, \underline{a})$ over Σ_{ext} is a map α assigning to every $x_i \in \underline{x}$ of sort S_i an element $x^\alpha \in S_i^{\mathcal{M}}$ and to every $a_j : E_j \longrightarrow U_j$ (with $a_j \in \underline{a}$) a function $a_j^\alpha : E_j^{\mathcal{M}} \longrightarrow U_j^{\mathcal{M}}$. The notion of assignment formally captures the current configuration of the working memory. An assignment to $(\underline{x}, \underline{a})$ can be seen as an SDB instance *extending* the SDB instance $\mathcal{M}$. Assuming that $\underline{a}$ contains $a_{i_1} : E \longrightarrow S_1, \cdots, a_{i_n} : E \longrightarrow S_n$, the memory relation E in the assignment $(\mathcal{M}, \alpha)$ is the set $\{\langle e, a^\alpha_{i_1}(e), \ldots, a^\alpha_{i_n}(e)\rangle \mid e \in E^{\mathcal{M}}\}$. Thus each tuple of E is formed by an implicit unique "identifier" $e \in E^{\mathcal{M}}$ (called *index*) and by "data" $\underline{a}^\alpha_i(e)$ taken from the static DB $\mathcal{M}$. When the system evolves, $E^{\mathcal{M}}$ remains fixed, while $\underline{a}^\alpha_i(e)$ may get repeatedly updated. "Removing" a tuple from E results in e being reset to undef. This clarifies the relational nature of the working memory.

Given a memory schema $(\underline{x}, \underline{a})$ over Σ_{ext} with $\underline{x} = x_1, \ldots, x_n$ and $\underline{a} = a_1, \ldots, a_m$, we list the kind of formulae that can be used in RABs: ***(1)*** an *initial formula* $\iota(\underline{x}, \underline{a}) := (\bigwedge_{i=1}^{n} x_i = c_i) \wedge (\bigwedge_{j=1}^{m} a_j = \lambda y.d_j)$, where c_i, d_j are constants from Σ (typically, c_i and d_j are initially set to undef); ***(2)*** a *state formula* $\exists \underline{e}\, \phi(\underline{e}, \underline{x}, \underline{a})$, where ϕ is quantifier-free and $\underline{e}$ are individual variables of artifact sorts (called *'index' variables*); ***(3)*** a *transition formula (t-formula)*

$$\mathrm{tr}(\underline{x}, \underline{a}, \underline{x}', \underline{a}') := \exists \underline{e}, \underline{d} \begin{pmatrix} \gamma(\underline{e}, \underline{d}, \underline{x}, \underline{a}) \wedge (\forall k\ \gamma_u(k, \underline{e}, \underline{d}, \underline{x}, \underline{a})) \\ \wedge \bigwedge_i x'_i = F_i(\underline{e}, \underline{d}, \underline{x}, \underline{a}) \\ \wedge \bigwedge_j a'_j = \lambda y.G_j(y, \underline{e}, \underline{d}, \underline{x}, \underline{a}) \end{pmatrix}$$

where the $\underline{e}$ and $\underline{d}$ are 'index' and 'basic sort' individual variables resp., k is an individual variable of artifact sort, γ (the "(plain) guard") and γ_u (the "universal guard") are quantifier-free, $\underline{x}'$ and $\underline{a}'$ are renamed copies of $\underline{x}$ and $\underline{a}$, and F_i, G_j (the "updates") are case-defined functions. The existentially quantified "data" variables $\underline{d}$ (of basic sort) are essential: they support existential queries over the SDB schema for retrieving data elements or arithmetical values (integers or reals), and (non-deterministic) external inputs. There are two key technical differences with the RAS model [Calvanese *et al.*, 2020]: universally quantified guards γ_u, and the usage of arithmetical constraints and operations in guards and updates.

Such transition formulae can model operations over tuples like *(i)* insertion (with/without duplicates) in a memory relation, *(ii)* removal from a memory relation, *(iii)* transfer from a memory relation to memory variables (and vice-versa), and *(iv)* bulk removal/update in a memory relation, based on a condition expressed on such relation. These operations can be formalized as RAB transitions: modeling patterns using this approach have been shown in [Calvanese *et al.*, 2019].

Definition 3. *A* relational action base (RAB) *is a tuple* $\mathcal{S} = \langle\langle \Sigma, T\rangle, \Sigma_{ext}, \underline{x}, \underline{a}, \iota(\underline{x}, \underline{a}), \tau(\underline{x}, \underline{a}, \underline{x}', \underline{a}')\rangle$*, where: (i)* $\langle \Sigma, T\rangle$ *is an* SDB *schema, (ii)* Σ_{ext} *is a memory extension of* Σ*, (iii)* $(\underline{x}, \underline{a})$ *is a memory schema over* Σ_{ext}*, (iv)* ι *is an initial formula, and (v)* τ *is a disjunction of t-formulae* tr.

Notice that τ symbolically represents the union of all system transitions. Such transitions are used to establish interaction between the static DB (with SDB schema DB) and the working memory (with memory schema $(\underline{x}, \underline{a})$).

Example 2. *Consider a RAB* $\mathcal{S}_{va}$ *capturing a process for evaluating visa applications and informing the applicants about the visa decisions.* $\mathcal{S}_{va}$ *works over the* SDB *schema from Example 1. The working memory of* $\mathcal{S}_{va}$ *consists of variables capturing the main process phases (*$pState$*), the citizen's visa status (*$visaStat$*) and ID used to notify about the application result (*$toNotify$*), and a multi-instance artifact for managing visa applications. The latter is formalized by adding to DB signature* Σ_v *a memory sort* appIndex *(to "internally" identify the applications), and by adding a memory schema with the applicant's ID* $applicant$: appIndex $\longrightarrow$ Int*, evaluation score* $appScore$: appIndex $\longrightarrow$ Int*, and application results* $appResult$: appIndex $\longrightarrow$ String.

We now showcase a few t-formulae for managing visa applications. We assume that if a memory variable/component is not mentioned in a t-formula, then its values remain unchanged. To insert an application into the system, the process has to be `enabled`*. The corresponding update simultaneously (i) selects the applicant's ID and inserts it into the memory component* $applicant$*, (ii) evaluates the visa application and inserts a non-negative score into the memory component* $appScore$*. Since memory tuples must have implicit identifiers, the above insertion requires a"free" index (i.e., an index pointing to an undefined applicant) to be selected:*

$$\exists i{:}\mathsf{appIndex}, \exists a{:}\mathsf{Int}, s{:}\mathsf{Int}\ \begin{pmatrix} pState = \texttt{enabled} \wedge a \neq \mathsf{undef} \wedge s \geq 0 \\ \wedge\ pState' = \texttt{enabled} \\ \wedge\ applicant' = \lambda j.\,(\mathsf{if}\ j = i\ \mathsf{then}\ a\ \mathsf{else}\ applicant[j]) \\ \wedge\ appScore' = \lambda j.\,(\mathsf{if}\ j = i\ \mathsf{then}\ s\ \mathsf{else}\ appScore[j]) \end{pmatrix}$$

Applications can be nondeterministically evaluated (by assigning `evaluation` *to* $pState$*), resulting in highly evaluated ones being* `approved` *and others* `rejected`*:*

$$pState = \texttt{evaluation} \wedge pState' = \texttt{evaluated} \\ \wedge appResult' = \lambda j.\begin{pmatrix} \mathsf{if}\ appScore[j] > 80\ \mathsf{then}\ \texttt{approved} \\ \mathsf{else}\ \texttt{rejected} \end{pmatrix}$$

If there is at least one approved application, a nondeterministically selected applicant is getting notified:

$$\exists i{:}\mathsf{appIndex}\ \begin{pmatrix} pState = \texttt{evaluated} \wedge pState' = \texttt{notified} \\ \wedge\ applicant[i] \neq \mathsf{undef} \wedge appResult[i] = \texttt{approved} \\ \wedge\ toNotify' = applicant[i] \wedge visaStat' = appResult[i] \end{pmatrix}$$

Algorithm 1: $\mathsf{BReach}_{\mathsf{RAB}}$

input: RAB $\langle\langle \Sigma, T\rangle, \Sigma_{ext}, \underline{x}, \underline{a}, \iota(\underline{x}, \underline{a}), \tau(\underline{x}, \underline{a}, \underline{x}', \underline{a}')\rangle$
input: (Unsafe) state formula $\upsilon(\underline{x}, \underline{a})$

```
P ⟵ υ; B̃ ⟵ ⊥;
while P ∧ ¬B̃ is T-satisfiable do
    if ι ∧ P is T-satisfiable then
        return (UNSAFE, unsafe trace witness)
    B̃ ⟵ P ∨ B̃;
    P ⟵ InstPre(τ, P);
    P ⟵ Covers(T, P);
return (SAFE, B̃);
```

Finally, we demonstrate a transition with a universal guard *checking whether no application has been approved and, if so, changes the process state to* `no-visa`*:* $pState = \texttt{evaluated} \wedge \forall k\ (appResult[k] \neq \texttt{approved}) \wedge pState' = \texttt{no-visa}$

5 Parameterized Safety Verification

We turn to safety verification of a RAB $\mathcal{S}$, formalizing it by analogy with RASs [Calvanese *et al.*, 2020], in tradition of artifact systems [Deutsch *et al.*, 2018]. As we will see, inspite of similar formulation, the corresponding algorithmic techniques for RABs are different, and so is proving their (meta-) properties. The main idea is to analyse whether $\mathcal{S}$ is safe (i.e., $\mathcal{S}$ never reaches an undesired state satisfying *unsafe formula* $\upsilon(\underline{x}, \underline{a})$) *independently from* the specific configuration of read-only data, i.e., for every instance of the SDB schema of $\mathcal{S}$. Technically, $\mathcal{S}$ is *safe w.r.t.* υ if there is no SDB instance $\mathcal{M}$ of $\langle \Sigma_{ext}, T\rangle$, no $k \geq 0$, and no assignment in $\mathcal{M}$ to the variables $\underline{x}^0, \underline{a}^0 \ldots, \underline{x}^k, \underline{a}^k$ s.t. formula

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

is true in $\mathcal{M}$ ($\underline{x}^i$, $\underline{a}^i$ are renamed copies of $\underline{x}$, $\underline{a}$). Formula can be seen as a *symbolic unsafe trace*. The safety problem $(\mathcal{S}, \upsilon)$ consists of establishing whether $\mathcal{S}$ is safe w.r.t υ.

Example 3. *The following formula describes an unsafety property for the RAB from Example 2, checking whether the evaluation notification goes to an applicant who was rejected:* $\exists i{:}\mathsf{appIndex}(applicant[i] \neq \mathsf{undef} \wedge toNotify = applicant[i] \wedge visaStat = \texttt{rejected} \wedge pState = \texttt{notified})$.

Safety verification procedure. Algorithm 1 introduces the $\mathsf{BReach}_{\mathsf{RAB}}$ procedure for safety verification. It shows how the backward reachability procedure for SMT-based safety verification, first studied in [Ghilardi and Ranise, 2010a] for array-based systems and refined in [Calvanese *et al.*, 2020] to deal with RASs, is extended to handle the advanced features of RABs. $\mathsf{BReach}_{\mathsf{RAB}}$ takes as input a RAB $\mathcal{S}$ and an unsafe formula υ. The main loop starts from the undesired states of the system (represented by υ), and explores *backwards* the state space by iteratively computing, symbolically, the set of states that can reach the undesired ones. Every iteration *regresses* the current undesired states by considering the transitions τ of $\mathcal{S}$ in a reversed fashion. As we describe next,

due to the presence of universal guards, τ can be reversed only in an approximated way. The computation of symbolic, approximated preimages is handled in Lines 5 and 6.

Let $\psi(\underline{x}, \underline{a})$ be a state formula describing the state of variables $\underline{x}, \underline{a}$. The *exact preimage* of the set of states captured by $\psi(\underline{x}, \underline{a})$ is the set of states captured by $\mathit{Pre}(\tau, \psi)$ (note that, when $\tau = \bigvee_i \mathrm{tr}_i$, then $\mathit{Pre}(\tau, \psi) = \bigvee_i \mathit{Pre}(\mathrm{tr}_i, \psi)$). This is the exact set of states that, by executing τ once, reaches the set of states described by ψ. The main issue we get in doing so is that a state formula is an existentially quantified Σ-formula over indexes only. While ψ is a state formula, *Pre* is not, making it impossible to reiterate the preimage computation. This is due to universally quantified 'index' variable k and existentially quantified 'data' variables $\underline{d}$ in τ. To attack this problem, we introduce in Lines 5 and 6 of $\mathsf{BReach}_{\mathrm{RAB}}$ two *over-approximations*, guaranteeing that the preimage is a proper state formula and that we use preimage computation to *regress* undesired states arbitrarily many times.

The first approximation (Line 5) compiles away the universal quantifier ranging over the index k through *instantiation*, by invoking $\mathit{InstPre}(\tau, \psi)$. Given $\tau := \bigvee_{r=1}^{p} \mathrm{tr}_r$, let $\forall k\ \gamma_u^r(k, \underline{e}, \underline{d}, \underline{x}, \underline{a})$ be the universal guard of tr_r for all $r = 1, \ldots, p$. $\mathit{InstPre}(\tau, \psi)$ approximates Pre by instantiating the universally quantified 'index' variable k with the existential 'index' variables appearing in $\mathit{Pre}(\mathrm{tr}_r, \psi)$, for all $r = 1, \ldots, p$. Formally, given $\psi := \exists \underline{e}_1 \varphi_1(\underline{e}_1, \underline{x}, \underline{a})$, $\mathit{InstPre}(\mathrm{tr}_r, \psi)$ is the formula obtained from

$$\exists\, \underline{x}', \underline{a}', \underline{e}, \underline{e}_1, \underline{d} \left(\begin{pmatrix} \gamma(\underline{e}, \underline{d}, \underline{x}, \underline{a}) \\ \wedge \bigwedge_{k \in \underline{e} \cup \underline{e}_1} \gamma_u^r(k, \underline{e}, \underline{d}, \underline{x}, \underline{a}) \\ \wedge \bigwedge_i x_i' = F_i(\underline{e}, \underline{d}, \underline{x}, \underline{a}) \\ \wedge \bigwedge_j a_j' = \lambda y. G_j(y, \underline{e}, \underline{d}, \underline{x}, \underline{a}) \end{pmatrix} \wedge \varphi_1(\underline{e}_1, \underline{x}', \underline{a}') \right)$$

by making the appropriate substitutions (followed by beta-reduction) in order to eliminate the existentially quantified variables $\underline{x}'$ and $\underline{a}'$ (see, e.g., [Calvanese *et al.*, 2020] for details). The second approximation (Line 6) takes the so-computed result $\mathit{InstPre}(\tau, \psi) \equiv \exists \underline{e}_2, \underline{d}\, \varphi_2$, and compiles away the existentially quantified data variables $\underline{d}$. This is done by invoking $\mathsf{Covers}(T, \exists \underline{e}_2, \underline{d}\, \varphi_2)$: it 'eliminates' the $\underline{d}$ variables via the cover computation algorithm for the theory combination $T^{DB} \cup T^{ar}$, which admits covers [Calvanese *et al.*, 2022a]. Both approximation operators are T-implied by the exact preimage operator.

Together, Lines 5 and 6 produce proper state formula ψ' that can be fed into another approximate preimage computation step. In fact, $\mathsf{BReach}_{\mathrm{RAB}}$ iteratively computes such preimages starting from the unsafe formula υ, until one of two possible termination conditions holds: these conditions are tested by SMT-solvers. The first condition occurs when the *non-inclusion* test in Line 2 fails, detecting a *fixpoint*: the set of current unsafe states is included in the set of states reached so far by the search. In this case, $\mathsf{BReach}_{\mathrm{RAB}}$ stops returning that $\mathcal{S}$ is *safe* w.r.t. υ. The second termination condition, occurring when the *non-disjointness* test in Line 3 succeeds, detects that set of current unsafe states intersects the initial states (i.e., satisfies ι). In this case, $\mathsf{BReach}_{\mathrm{RAB}}$ stops returning that $\mathcal{S}$ is *unsafe* w.r.t. υ, together with a symbolic unsafe trace of the form (1). Such a trace provides a sequence of transitions tr_i that, starting from the initial configurations, witness how $\mathcal{S}$ can evolve from an initial state in ι, under some instance of its SDB, to a state satisfying υ.

Being preimages computed in an over-approximated way, unsafety may be spuriously returned, together with an unsafe spurious trace that cannot be produced by a given RAB. We next study this and other meta-properties of $\mathsf{BReach}_{\mathrm{RAB}}$.

Meta-properties. Consider a procedure for verifying safety of RABs. Given a RAB $\mathcal{S}$ and an unsafe formula υ, a `SAFE` (resp. `UNSAFE`) output is *correct* iff $\mathcal{S}$ is safe (resp. unsafe) w.r.t. υ. We use this to define some key meta-properties.

Definition 4. *Given a RAB $\mathcal{S}$ and an unsafe formula υ, a procedure for verifying unsafety of $\mathcal{S}$ w.r.t. υ is: (i)* sound *if, when terminating, it returns a correct result; (ii)* partially sound *if a `SAFE` result is always correct; (iii)* complete *if, whenever `UNSAFE` is the correct result, then it returns so.*

By using that reachable states via approximated preimages include those obtained via exact preimages, we get:

Proposition 1. *If BReach_{RAB} returns `SAFE` when applied to the safety problem $(\mathcal{S}, \upsilon)$, then $\mathcal{S}$ is safe w.r.t. υ.*

It can be also proved (see the extended version [Ghilardi *et al.*, 2022b]) that if S is *unsafe* w.r.t. υ, the procedure terminates with `UNSAFE`. We show that $\mathsf{BReach}_{\mathrm{RAB}}$ can only be used partially to verify unsafety of RABs. Next, *effectiveness* means that all subprocedures in $\mathsf{BReach}_{\mathrm{RAB}}$ can be effectively computed (from [Calvanese *et al.*, 2020], the occurring T-satisfiability tests are decidable).

Theorem 1. BReach_{RAB} *is* effective*,* partially sound *and* complete *when verifying unsafety of RABs.*

RABs that do not employ universal guards collapse to RASs equipped with numerical values and arithmetics. By carefully combining results on soundness and completeness of backward reachability of RASs [Calvanese *et al.*, 2020], and on the existence of combined covers for $\mathcal{EUF}$ and $\mathcal{LIA}$/$\mathcal{LRA}$ [Calvanese *et al.*, 2022a], we obtain for these RABs that no spurious results are produced:

Theorem 2. BReach_{RAB} *is* effective*,* sound *and* complete *(and hence a semi-decision procedure) when verifying safety of RABs without universal guards.*

Proof. From Theorem 1, the only thing that remains to prove is that Algorithm 1 is fully sound if the RAB does not contain universal guards. In this case, $\mathit{InstPre} \equiv \mathit{Pre}$, hence Algorithm 1 coincides with the SMT-based backward reachability procedure for RASs introduced in [Calvanese *et al.*, 2020]. Notice, however, that RABs without universal guards still strictly extend plain RASs from [Calvanese *et al.*, 2020], since SDB are strictly more expressive than DB schemas of RASs: SDB schema contain arithmetic operations and are constrained by theory combinations where one component is an arithmetical theory. Nevertheless, using the results and the procedures presented in [Calvanese *et al.*, 2021; Calvanese *et al.*, 2022a] for computing covers, and given the equivalence between computing covers and eliminating quantifiers in model completions from [Calvanese *et al.*, 2020], in order to conclude one can use the analogous arguments based on model completions as the ones used in [Calvanese *et al.*, 2020] for proving full soundness of RASs. □

6 Safety Universal Invariants

Injecting and exploiting invariants in the verification of dynamic systems is a well-known approach. Also, instantiation is a widely studied in invariant checking [Feldman *et al.*, 2019]. In our setting, invariants can dramatically prune the search space of backward reachability procedures, in general also increasing the chances that the procedure terminates [Ghilardi and Ranise, 2010a]. We contribute to this line by introducing a suitable notion of (universal) invariant for RABs, and by studying its impact to safety verification.

Definition 5. $\phi(\underline{x}, \underline{a}) := \forall \underline{i}\, \psi(\underline{i}, \underline{x}, \underline{a})$ *(with* $\underline{i}$ *variables of artifact sort) is an* (inductive) universal invariant *for a RAB* $\mathcal{S}$ *iff (a)* $T \models \iota(\underline{x}, \underline{a}) \Rightarrow \phi(\underline{x}, \underline{a})$*, and (b)* $T \models \phi(\underline{x}, \underline{a}) \wedge \tau(\underline{x}, \underline{a}, \underline{x}', \underline{a}') \Rightarrow \phi(\underline{x}', \underline{a}')$*. If, in addition to (a) and (b), we also have (c)* $\phi(\underline{x}, \underline{a}) \wedge \upsilon(\underline{x}, \underline{a})$ *is* T*-unsatisfiable, then* $\phi(\underline{x}, \underline{a})$ *is a* safety universal invariant *for the safety problem* $(\mathcal{S}, \upsilon)$.

Following the arguments from [Calvanese *et al.*, 2020], the T-satisfiability tests required in (a), (b), and (c) are decidable. Using invariants for safety verification is, to some extent, "more general" than using (variants of) backward reachability. Unfortunately, the so-called *invariant method* is more challenging because finding safety invariants cannot be mechanized (see, e.g., [Ghilardi and Ranise, 2010a]). Yet, if a universal invariant has been found in some way (e.g., with heuristics) or supplied by a trusted knowledge source, then it can be effectively employed in the fixpoint test of backward reachability, e.g., by replacing Line 2 of Algorithm 1 with:

$$2'\ \textbf{while}\ (P \wedge \mathit{Inv} \wedge \neg\tilde{B}\ \text{is}\ T\text{-sat.})\ \textbf{do}$$

where Inv is a conjunction of universal invariants. This further constrains the formula $P \wedge \neg\tilde{B}$ with Inv, possibly increasing the chances to detect unsatisfiability.

The following result is useful: it shows that Algorithm 1 can be used to find safety universal invariants.

Proposition 2. *If Algorithm 1 returns* $(\texttt{SAFE}, \tilde{B})$*, then* $\neg\tilde{B}$ *is a safety universal invariant.*

Sketch. By construction, $\tilde{B}$ is an existential formula over artifact sorts only, hence $\neg\tilde{B}$ is universal. We show in the extended version [Ghilardi *et al.*, 2022b] that conditions (a),(b),(c) from Definition 5 hold. □

Let P_n the value of the variable P at the n-th iteration of the main loop $\mathsf{BReach}_{\mathsf{RAB}}$. We prove the following key, nontrivial result on how safety universal invariants relate to the (approximated) preimages computed by $\mathsf{BReach}_{\mathsf{RAB}}$.

Theorem 3. *If a safety universal invariant* ϕ *exists for a RAB* $\mathcal{S}$ *w.r.t.* υ*, then for every* $n \in \mathbb{N}$*, we have* $P_n \rightarrow \neg\phi$.

Sketch. The argument is quite delicate: it is always possible to extend a structure w.r.t. its Σ-reduct and at the same time to restrict the given structure w.r.t. its $\Sigma_{ext} \setminus \Sigma$-reduct, and these 'opposite' constructions does not interfere with each other. We prove the statement by induction on the number n of iterations of the main loop of $\mathsf{BReach}_{\mathsf{RAB}}$.

Base case. In case $n = 0$, the statement follows from $P_0 \equiv \upsilon$ and condition (b) of Definition 5.

Inductive case. By inductive hypothesis, we have that $T \models P_n \rightarrow \neg\phi$. We show that $T \models P_{n+1} \rightarrow \neg\phi$ holds as well. First, we notice that, by condition (c) of Definition 5, $T \models \tau(\underline{x}, \underline{a}, \underline{x}', \underline{a}') \wedge \neg\phi(\underline{x}', \underline{a}') \rightarrow \neg\phi(\underline{x}, \underline{a})$ holds. The previous implication can be rewritten as

$$T \models \mathit{Pre}(\tau, \neg\phi) \rightarrow \neg\phi \tag{2}$$

since, by definition of preimage, we have $T \models \exists\underline{x}'\exists\underline{a}'(\tau(\underline{x}, \underline{a}, \underline{x}', \underline{a}') \wedge \neg\phi(\underline{x}', \underline{a}')) \equiv \mathit{Pre}(\tau, \neg\phi)$.

Preimage is monotonic, i.e., given two formulae α and β, $\alpha \rightarrow \beta$ implies $\mathit{Pre}(\tau, \alpha) \rightarrow \mathit{Pre}(\tau, \beta)$. Hence, from the inductive hypothesis $T \models P_n \rightarrow \neg\phi$, we get

$$T \models \mathit{Pre}(\tau, P_n) \rightarrow \mathit{Pre}(\tau, \neg\phi). \tag{3}$$

From Formulae (3) and (2) it follows that

$$T \models \mathit{Pre}(\tau, P_n) \rightarrow \neg\phi. \tag{4}$$

By construction and induction, $\mathit{InstPre}(\tau, P_n)$ is an existential formula over artifact and basic sorts. By applying Covers, we eliminate the existentially quantified variables over basic sorts, and get an existential formula over artifact sorts only. Hence, since $P_{n+1} \equiv \mathsf{Covers}(T, \mathit{InstPre}(\tau, P_n))$, we can assume that P_{n+1} has the form $\exists\underline{\hat{e}}\, \psi(\underline{\hat{e}})$, where ψ is quantifier-free. Thus, we need to prove that:

$$T \models \exists\underline{\hat{e}}\, \psi(\underline{\hat{e}}) \rightarrow \neg\phi. \tag{5}$$

By reduction to absurdum, suppose that there is a SDB instance $\mathcal{A}$ s.t. $\mathcal{A} \not\models \exists\underline{\hat{e}}\, \psi(\underline{\hat{e}}) \rightarrow \neg\phi$. By model-theoretic properties of covers, from $\mathcal{A}$ it is possible to build a SDB instance $\mathcal{B}_1$ s.t. $\mathcal{B}_1 \models \mathit{InstPre}(\tau, P_n) \wedge \phi$. Moreover, from $\mathcal{B}_1$ we can find a SDB instance $\mathcal{B}_2$ where the instantiations of the universal quantifiers cover all the elements of its domain: hence, $\mathcal{B}_2 \models \mathit{Pre}(\tau, P_n) \wedge \phi$, which contradicts entailment (4), as wanted. All the details are in [Ghilardi *et al.*, 2022b]. □

By combining Definition 5 and Theorem 3 we get:

Corollary 1. *If there exists a safety universal invariant* ϕ *a RAB* $\mathcal{S}$*, then* $\mathsf{BReach}_{\mathit{RAB}}$ *cannot return* `UNSAFE`.

Corollary 1 implies that if there is a safety universal invariant for $\mathcal{S}$, then Algorithm 1 cannot find any spurious unsafe traces. We know from Proposition 1 that in case Algorithm 1 returns $(\texttt{SAFE}, \tilde{B})$, then the system is safe: notice also that in this case, $\neg\tilde{B}$ is a safety universal invariant (see [Ghilardi *et al.*, 2022b] for details). However, if it returns an unsafe outcome, the answer could be wrong due to the presence of spurious unsafe traces.

The importance of Corollary 1 lies in the fact that, even if we do not know any safety universal invariant, we are assured that in case one exists, the procedure behaves well/properly/in a fair way, as no unsafe outcome can be returned and therefore no spurious trace can be detected. Corollary 1 is conceptually analogous to the invariant result in [Karbyshev *et al.*, 2017], with two subtantial technical differences: *(i)* they deal with formulae that fall into the decidable Barneys-Schönfinkel FO fragment (the '$\exists^*\forall^*$' fragment) with only relational symbols, which does not capture functional key dependencies nor standard arithmetical operations over real/integers as we do here; *(ii)* the two results hold for different verification procedures: ours for backward reachability, theirs for IC3.

Example 4. *The following is a safety universal invariant for* $(\mathcal{S}_{va}, \upsilon)$*:* $(\mathit{visaStat} = \texttt{rejected} \rightarrow \mathit{toNotify} = \mathsf{undef})$.

Ex	Ar	UG	#T	#Q	#U	#S	μT	MaxT	#(calls)
E01	n	y	28	14	9	3	0.82	1.28 (U)	3398.2
E02	y	n	48	3	6	6	0.31	0.37 (S)	4011.4
E03	n	y	38	17	6	6	0.74	2.49 (S)	4301.8
E04	n	n	22	9	5	7	0.32	0.64 (U)	1899.7
E05	n	n	20	9	5	7	0.23	0.51 (U)	1656.8
E06	y	n	16	6	5	7	0.17	0.34 (U)	1496.4
E07	n	y	14	10	10	2	0.18	0.62 (U)	1126.1
E08	n	n	34	4	6	6	0.65	2.50 (S)	3036.4
E09	n	n	23	3	9	3	19.76	171.63 (S)	20302.4
E10	n	y	21	11	9	3	0.22	0.46 (S)	1550.3
E11	y	y	21	8	7	5	0.15	0.38 (S)	1339.6
E12	y	y	21	8	7	5	0.15	0.38 (S)	1326.7
E13	n	y	20	10	8	4	0.65	2.39 (U)	2764.8
E14	n	y	20	10	8	4	0.62	2.24 (U)	2724.0
E15	y	n	22	13	7	5	0.37	0.76 (U)	2283.4
E16	n	n	47	19	8	4	0.96	5.47 (U)	5151.8
E17	y	n	13	8	7	5	0.08	0.13 (U)	768.5
E18	n	n	21	9	6	6	0.11	0.19 (U)	1212.6
E19	y	n	31	10	6	6	0.37	0.74 (U)	2416.2
E20	n	n	24	7	6	6	0.11	0.17 (U)	1026.5
E21	y	y	27	7	7	5	0.40	1.38 (S)	2509.6
E22	y	y	20	2	9	3	8.15	53.92 (U)	9736.3
E23	y	n	17	2	8	4	0.99	3.73 (S)	2995.9
E24	y	y	15	7	7	5	0.11	0.13 (U)	1051.5
E25	n	y	34	14	6	6	4.52	24.74 (S)	7486.7
E26	n	y	34	15	6	6	4.97	17.79 (S)	4811.1
E27	n	y	34	15	7	5	4.59	20.81 (S)	4633.7
E28	y	y	32	10	7	5	0.23	0.46 (U)	2740.0
E29	n	y	15	3	6	6	0.08	0.42 (S)	648.8
E30	y	n	26	9	6	6	0.31	0.79 (U)	2316.2
E31	y	y	37	12	8	4	0.38	0.71 (U)	3367.0
E32	y	n	51	9	9	3	2.48	8.49 (U)	9231.3
E33	y	n	43	8	9	3	1.27	4.24 (S)	6637.8
E+	y	y	15	7	22	11	7.39	98.27 (S)	5612.5

Table 1: Experimental results for safety properties

7 Practical Verification of RABs

We now put our verification machinery in practice, exploiting that, as shown in our technical development, we can employ the different modules of MCMT to handle parameterized safety checking of RABs. We test the novel features of RABs, namely universal quantification and arithmetics on relational dynamic systems, for which no other existing tool can be employed. Specifically, we conduct an extensive experimental evaluation on RABs representing 33 (32 plus one variant) concrete data-aware processes derived from the existing benchmark in [Li *et al.*, 2017]. We encode *all* these 33 models into RABs, using the concrete syntax of the database-driven mode of MCMT (available since Version 2.8). We set up our benchmark by taking 26 out of the 33 process models in [Li *et al.*, 2017], naturally extending them with arithmetic and/or universal guards (features not supported by VERIFAS [Li *et al.*, 2017]). The remaining 7 examples do not lend themselves to such an extension, and are kept unaltered. We nevertheless include them for completeness. We also add an extra example E+ that incorporates all modeling features of RABs (e.g., *bulk updates* that go beyond the capabilities of VERIFAS [Li *et al.*, 2017]), with the intention of stress-testing MCMT. Each example is checked against 12 conditions, with at least one safe and one unsafe; E+ is checked against 33 conditions. Overall, we ran MCMT over 429 specification files. The benchmark is on https://github.com/AlessandroGianola/RAB-verification.

Experiments were performed on a 2.3 GHz Intel Core i5 machine with macOS High Sierra and 8 GB RAM. Results are shown in Table 1 (more detailed information is in [Ghilardi *et al.*, 2022b]), which reports, for each example: *(i)* if it contains arithmetics (**Ar**) and universal guards (**UG**); *(ii)* the number of transitions (**#T**), also counting those that contain quantified data variables (**#Q**). The table also includes several measures: *(i)* number of UNSAFE (**#U**) and SAFE (**#S**) outcomes; *(ii)* seconds for the average (μ**T**) and maximum (**MaxT**) MCMT execution time; *(iii)* indications of number of calls to the external SMT solver (**#(calls)**).

The means of execution times are relatively small. For the 26 plus 1 models supporting the specific features of RABs, MCMT terminates in less than one second for 85.2% of the tested files, and in less than one minute for 343 out of 345. The two files over one minute take 90.36 and 98.27 seconds. For all 34 models, MCMT terminates in less than one second for 366 out of 429, the highest timing being 171.63 seconds. Our preliminary experiments indicate that universal guards and arithmetic constraints do not sensibly impact the timing.

If the outcome is unsafe, MCMT returns a witness that reconstructs the unsafe trace as a formula. With this formula one can test whether the trace is spurious or not. This feature is currently not directly supported by MCMT, but can be tested via SMT solvers off-the-shelf. In particular, trace formulae belong to the $\exists^*\forall^*$ (EPR) fragment of FOL, for which the Z3 solver [de Moura and Bjørner, 2008] has a dedicated decision procedure. The complexity is the one of satisfiability checking in the EPR fragment, which is NEXPTIME-complete.

As far as we know, no other tool or benchmark exists for further experimental evaluation. The existing model checkers (also infinite-state ones) rarely can handle specifications expressed with quantifiers: however, as also argued in [Calvanese *et al.*, 2020], quantifiers are needed when dealing with relational database-driven verification. For handling them, we need non-trivial approximation modules, such as instantiation for universal variables and covers computation for data variables. MCMT is the only model checker specifically oriented to employ these sophisticated techniques.

8 Conclusions

We have presented the RAB framework for modeling rich relational dynamic systems with arithmetics and universal guards, developing techniques for safety verification and invariant-based analysis. Our results do not only have a foundational importance, but also witness that separately implemented verification modules in the MCMT model checker can be indeed gracefully combined to obtain a safety verification procedure for RABs. The experimental evaluation here reported shows that this is an effective, practical approach.

In the paper, we have surveyed concrete scenarios that require the advanced features of RABs, in the context of multiagent systems [Felli *et al.*, 2021] and extensions of Petri nets [Fahland, 2019; Polyvyanyy *et al.*, 2019; Ghilardi *et al.*, 2020]. The next step is to encode them into RABs. Considering that encoding concrete models typically produce update formulae with a specific shape, we then want to extract relevant universal invariants as a by-product of the encoding. Another interesting direction is to verify properties that go beyond safety. This is an open research problem for SMT-based verification in general, as symbolic reachability techniques like those used in this paper are tailored to safety.

Acknowledgments

This research has been partially supported by the Italian Ministry of University and Research (MUR) under PRIN project PINPOINT Prot. 2020FNEB27, and by the Free University of Bozen-Bolzano with the ADAPTERS project.

References

[Abiteboul *et al.*, 1995] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[Alberti *et al.*, 2012] Francesco Alberti, Silvio Ghilardi, Elena Pagani, Silvio Ranise, and Gian Paolo Rossi. 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, 2012.

[Bagheri Hariri *et al.*, 2013a] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. Verification of relational data-centric dynamic systems with external services. In *Proceedings of PODS 2013*, pages 163–174, 2013.

[Bagheri Hariri *et al.*, 2013b] Babak Bagheri Hariri, Diego Calvanese, Marco Montali, Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli. Description logic knowledge and action bases. *J. Artif. Intell. Res.*, 46:651–686, 2013.

[Bagheri Hariri *et al.*, 2014] Babak Bagheri Hariri, Diego Calvanese, Marco Montali, and Alin Deutsch. State-boundedness in data-aware dynamic systems. In *Proceedings of KR 2014*. AAAI Press, 2014.

[Baral and De Giacomo, 2015] Chitta Baral and Giuseppe De Giacomo. Knowledge representation and reasoning: What's hot. In *Proceedings of AAAI 2015*. AAAI Press, 2015.

[Belardinelli *et al.*, 2014] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. Verification of agent-based artifact systems. *Journal of Artificial Intelligence Research*, 51:333–376, 2014.

[Belardinelli, 2014] Francesco Belardinelli. Model checking auctions as artifact systems: Decidability via finite abstraction. In *Proceedings of ECAI 2014*, pages 81–86, 2014.

[Calvanese *et al.*, 2013] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. Foundations of data-aware process analysis: A database theory perspective. In *Proceedings of PODS 2013*, pages 1–12, 2013.

[Calvanese *et al.*, 2018] Diego Calvanese, Giuseppe De Giacomo, Marco Montali, and Fabio Patrizi. First-order μ-calculus over generic transition systems and applications to the situation calculus. *Information and Computation*, 259(3):328–347, 2018.

[Calvanese *et al.*, 2019] Diego Calvanese, Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin. Formal modeling and SMT-based parameterized verification of data-aware BPMN. In *Proceeding of BPM 2019*, volume 11675 of *LNCS*, pages 157–175. Springer, 2019.

[Calvanese *et al.*, 2020] Diego Calvanese, Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin. SMT-based verification of data-aware processes: a model-theoretic approach. *Mathematical Structures in Computer Science*, 30(3):271–313, 2020.

[Calvanese *et al.*, 2021] Diego Calvanese, Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin. Model completeness, uniform interpolants and superposition calculus. *Journal of Automated Reasoning*, 65(7):941–969, 2021.

[Calvanese *et al.*, 2022a] Diego Calvanese, Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin. Combination of uniform interpolants via Beth definability. *Journal of Automated Reasoning*, 66(3):409–435, 2022.

[Calvanese *et al.*, 2022b] Diego Calvanese, Giuseppe De Giacomo, Marco Montali, and Fabio Patrizi. Verification and monitoring for first-order LTL with persistence-preserving quantification over finite and infinite traces. In *Proceedings of IJCAI 2022*, pages 2553–2560, 2022.

[Cimatti *et al.*, 2021] Alessandro Cimatti, Alberto Griggio, and Gianluca Redondi. Universal invariant checking of parametric systems with quantifier-free SMT reasoning. In *Proceedings of CADE 28*, volume 12699 of *LNCS*, pages 131–147. Springer, 2021.

[De Giacomo *et al.*, 2016] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. Bounded situation calculus action theories. *Artificial Intelligence*, 237, 2016.

[de Moura and Bjørner, 2008] Leonardo de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proceedings of TACAS 2008*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[Deutsch *et al.*, 2016] Alin Deutsch, Yuliang Li, and Victor Vianu. Verification of hierarchical artifact systems. In *Proceedings of PODS 2016*, pages 179–194, 2016.

[Deutsch *et al.*, 2018] Alin Deutsch, Richard Hull, Yuliang Li, and Victor Vianu. Automatic verification of database-centric systems. *ACM SIGLOG News*, 5(2):37–56, 2018.

[Deutsch *et al.*, 2019] Alin Deutsch, Yuliang Li, and Victor Vianu. Verification of hierarchical artifact systems. *ACM Transactions on Database Systems*, 44(3):12:1–12:68, 2019.

[Fahland, 2019] Dirk Fahland. Describing behavior of processes with many-to-many interactions. In *Proceedings of PETRI NETS 2019*, volume 11522 of *LNCS*, pages 3–24. Springer, 2019.

[Feldman *et al.*, 2019] Yotam M. Y. Feldman, Oded Padon, Neil Immerman, Mooly Sagiv, and Sharon Shoham. Bounded quantifier instantiation for checking inductive invariants. *Logical Methods in Computer Science*, 15(3), 2019.

[Felli *et al.*, 2020] Paolo Felli, Alessandro Gianola, and Marco Montali. A SMT-based implementation for safety

checking of parameterized multi-agent systems. In *Proceedings of PRIMA 2020*, volume 12568 of *LNCS*, pages 259–280. Springer, 2020.

[Felli *et al.*, 2021] Paolo Felli, Alessandro Gianola, and Marco Montali. SMT-based safety checking of parameterized multi-agent systems. In *Proceedings of AAAI 2021*. AAAI Press, 2021.

[Felli *et al.*, 2022] Paolo Felli, Marco Montali, and Sarah Winkler. Linear-time verification of data-aware dynamic systems with arithmetic. In *Proceedings of AAAI 2022*, pages 5642–5650. AAAI Press, 2022.

[Gerevini *et al.*, 2008] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artif. Intell.*, 172(8-9):899–944, 2008.

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

[Ghilardi and Ranise, 2010b] Silvio Ghilardi and Silvio Ranise. MCMT: A model checker modulo theories. In *Proceedings of IJCAR 2010*, volume 6173 of *LNCS (LNAI)*, pages 22–29. Springer, 2010.

[Ghilardi *et al.*, 2020] Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin. Petri nets with parameterised data - modelling and verification. In *Proceedings of BPM 2020*, volume 12168 of *LNCS*, pages 55–74. Springer, 2020.

[Ghilardi *et al.*, 2022a] Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin. Petri net-based object-centric processes with read-only data. *Information Systems*, 107, 2022.

[Ghilardi *et al.*, 2022b] Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin. Relational action bases: Formalization, effective safety verification, and invariants (extended version). *CoRR*, abs/2208.06377, 2022.

[Gianola, 2022] Alessandro Gianola. *SMT-based Safety Verification of Data-Aware Processes: Foundations and Applications*. PhD thesis, Free University of Bozen-Bolzano, 2022.

[Gurfinkel *et al.*, 2016] Arie Gurfinkel, Sharon Shoham, and Yuri Meshman. SMT-based verification of parameterized systems. In *Proceedings of FSE 2016*, pages 338–348. ACM, 2016.

[Hull, 2008] Richard Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *Proceedings of OTM 2008*, volume 5332 of *LNCS*, pages 1152–1163. Springer, 2008.

[Karbyshev *et al.*, 2017] Aleksandr Karbyshev, Nikolaj S. Bjørner, Shachar Itzhaky, Noam Rinetzky, and Sharon Shoham. Property-directed inference of universal invariants or proving their absence. *Journal of the ACM*, 64(1):7:1–7:33, 2017.

[Kouvaros and Lomuscio, 2016] Panagiotis Kouvaros and Alessio Lomuscio. Parameterised verification for multi-agent systems. *Artificial Intelligence*, 234:152–189, 2016.

[Li *et al.*, 2017] Yuliang Li, Alin Deutsch, and Victor Vianu. VERIFAS: A practical verifier for artifact systems. *Proceedings of the VLDB Endowment*, 11(3):283–296, 2017.

[Polyvyanyy *et al.*, 2019] Artem Polyvyanyy, Jan Martijn E. M. van der Werf, Sietse Overbeek, and Rick Brouwers. Information systems modeling: Language, verification, and tool support. In *Proceedings of CAiSE 2019*, volume 11483 of *LNCS*, pages 194–212. Springer, 2019.

[Vianu, 2009] Victor Vianu. Automatic verification of database-driven systems: a new frontier. In *Proceedings of ICDT 2009*, pages 1–13, 2009.

[Yang *et al.*, 2022] Wen-Chi Yang, Jean-François Raskin, and Luc De Raedt. Lifted model checking for relational mdps. *Machine Learning*, 2022.