VERIFICATION OF CONJUNCTIVE ARTIFACT-CENTRIC SERVICES

GIUSEPPE DE GIACOMO, RICCARDO DE MASELLIS, RICCARDO ROSATI

Dipartimento di Ingegneria informatica, Automatica e Gestionale
Sapienza Università di Roma
Via Ariosto 25, 00185 Rome, Italy

{*degiacomo*|*demasellis*|*rosati*}*@dis.uniroma1.it*

Received (Day Month Year)
Revised (Day Month Year)

An artifact-centric service is a stateful service that holistically represents both the data and the process in terms of a (dynamic) *artifact*. An artifact is constituted by a *data component*, holding all the data of interest for the service, and a *lifecycle*, which specifies the process that the service enacts. In this paper, we study artifact-centric services whose data component is a full-fledged relational database, queried through (first-order) conjunctive queries, and the lifecycle component is specified as sets of condition-action rules, where actions are tasks invocations, again based on conjunctive queries. Notably, the database can evolve in an unbounded way due to new values (unknown at verification time) inserted by tasks. The main result of the paper is that verification in this setting is decidable under a reasonable restriction on the form of tasks, called *weak acyclicity*, which we borrow from the recent literature on data exchange. In particular, we develop a sound, complete and terminating verification procedure for sophisticated temporal properties expressed in a first-order variant of μ-calculus.

Keywords: Business Artifacts; Verification; Conjunctive Queries.

1. Introduction

In the past years, the so called artifact-centric approach to modeling workflows and services has emerged, with the fundamental characteristic of considering both data and processes as first-class citizens in service design and analysis [47,39,24,18,57,3]. In such an approach, the key elements of services are *artifacts*, which are business-relevant entities evolving over time. Artifacts are constituted by *(i)* a *data component*, which is used to hold the relevant information to be manipulated by the service, and *(ii)* the *lifecycle* formed by the invokable (atomic) *tasks* and a process based on them. Executing a task has effects on the data manipulated by the service, on the service state, and on the information exchanged with the external world. The process specifies a sequencing of the task invocations, thus characterizing the dynamic behavior of the service.

The holistic view of data and processes together aims at avoiding the notorious discrepancy between data modeling and process modeling of more traditional approaches that consider these two aspects separately. Conversely, by treating both aspects as first-class citizens, the artifact-centric approach ultimately promises to lead to a greater efficiency, especially in dealing with business transformation [9,10].

From a formal point of view, artifact-centric services deeply challenge the verification

community by requiring simultaneous attention to both data and processes. Indeed, on the one hand, they deal with full-fledged processes and require analysis in terms of verification of sophisticated temporal properties [23]. On the other hand, the presence of possibly unbounded data [4] makes the usual analysis based on model checking of finite-state systems impossible in general, since, when data evolution is taken into account, the whole system becomes infinite-state.

In this paper, we study a family of artifact-centric services whose task specification is based on the notion of *conjunctive queries*. Taking the core concepts from recent proposals by Hull et al. [9,33,18], we consider an artifact as formed by a *data component*, which describes its static part, and by a *lifecycle*, which characterizes the dynamic aspects. In our framework, the data component is a full-fledged relational database, and the lifecycle is specified, in a declarative way, as a set of condition-action rules. Conditions are evaluated on the current *snapshot* of the artifact, i.e., the current state of the database. Actions are *task* invocations, that query the current snapshot and generate the next one, possibly introducing new existential values representing inputs from the outside world. Since such values are yet unknown at analysis time, they are represented as nulls. Similar to the context of semantic web services [51], and deeply rooted in the literature on Reasoning about Actions in AI [50], here the behavior of tasks is characterized using pre-conditions and effects (or post-conditions). However, the key point of our proposal is that both pre-conditions and effects are expressed as conjunctive queries.

This implies that: *(i)* we query only positive information in the current state of the artifact (negation is not allowed in conjunctive queries), and *(ii)* we do not get disjunctive information as the effect of executing a task, though we get existential values, i.e., nulls, as the result of introducing unknown input from outside. The latter, *(ii)*, assures that the state generated by the execution of a task is still a relational database, even if it contains nulls. Such an assumption can often be made, and, in particular, in all those applications in which we have (almost) complete information on the result of executing a task. Instead, the former, *(i)*, might be a restriction in practice: it limits the way we can formulate queries on the current state and hence the way we can specify tasks. It is certainly of interest to introduce negation in the query language to specify tasks. Indeed, [37] moves the first step in this direction by extending the approach presented here to full first-order queries in preconditions of tasks. However, this extension requires a much more sophisticated technical development that hides the core technique we propose here. Concentrating on conjunctive queries allows for exposing, in the cleanest and most elegant way, the very idea at the base of the approach, that is, the correspondence between tasks execution and data exchange and data integration [30,42]. Research in these fields has deeply investigated the mapping between databases expressed through correspondences between conjunctive queries, consisting of the so-called tuple-generating dependencies (tgds) in the database jargon [4]. In a nutshell, the core idea of our work is to consider the current state of data, and their state after the performance of a task, as two databases related through a set of tgds. This view allows us to leverage on conditions that guarantee *finite chase of tgds* [30,28,1,44,45], to get decidability results even for very powerful verification languages.

On top of such a framework, we introduce a powerful verification logic based on a

first-order variant of μ-calculus [43,48,29,15] to express temporal properties. μ-calculus is well known to be more expressive than virtually all temporal logics used in verification, including CTL, LTL, CTL*, PDL, and many others. For this reason, our results for μ-calculus immediately carry over to all these other logics, giving us a very robust approach.

The main result of the paper is that the resulting setting, while quite expressive and inherently infinite-state, admits decidable verification under a reasonable restriction on the form of the effects of tasks, called *weak acyclicity* [30]. The crux of the result is that conjunctive queries are unable to distinguish between homomorphic equivalent databases: this can be exploited to bound the number of distinguishable artifact states. Thus, we can reduce verification to model checking of a finite-state transition system, which acts as a faithful abstraction of the original artifact.

The rest of the paper is organized as follows. Section 2 introduces our artifacts based on conjunctive queries. Section 3 and Section 4 illustrate the execution of artifacts, and the verification formalism for such executions. Section 5 presents our main results, including the decidability of weakly acyclic artifacts. Section 6 discusses related work. Finally, Section 7 concludes the paper.

2. Conjunctive Artifacts

Conjunctive artifact-centric services are services based on the notion of artifact, which thus merges data and processes in a single unit. More precisely, an artifact is composed by the following three components:

- The *artifact data component*, which captures the information manipulated by the artifact. In our case such a component is a relational database. States of the service correspond to states of the database.
- The set of *artifact tasks*, which is the set of atomic actions that manipulate artifact data. A characteristic aspect of our study is that such tasks are specified in terms of dependencies between conjunctive queries (see below).
- The *artifact lifecycle*, which specifies the actual process of the artifact in terms of tasks that can be executed at each state. Technically, it is specified in terms of condition-action rules, where the conditions are again based on conjunctive queries.

Formally, an *artifact* is a tuple $A = \langle \mathcal{D}, \mathcal{T}, \mathcal{C} \rangle$, where:

- $\mathcal{D} = \langle \mathcal{S}, I_0 \rangle$ is an *artifact data component* formed by the *data schema* $\mathcal{S}$ and the *initial artifact data instance* I_0;
- $\mathcal{T}$ is a set of *tasks*; and
- $\mathcal{C}$ is a lifecycle, i.e., a set of *condition-action rules*.

Next, we define each artifact component in detail.

Artifact data component. An artifact data component $\mathcal{D} = \langle \mathcal{S}, I_0 \rangle$ is formed by an artifact data schema $\mathcal{S}$ and an initial artifact instance I_0 conforming to such a schema. The

artifact data schema $\mathcal{S}$ is a relational schema formed by a finite set of *relational (predicate) symbols* $R_1, \ldots, R_n$, each one with an associated arity, and a finite or countably infinite set of *constant symbols* $\vec{c} = c_1, c_2, \ldots$.

An *artifact data instance*, or simply *instance* (including the initial one I_0), over the schema $\mathcal{S}$ is a standard first-order interpretation with a *fixed interpretation domain*. More precisely, a data instance is a pair $I = \langle \Delta, \cdot^I \rangle$ where:

- Δ is a countably infinite *domain*, fixed a-priori and shared by every data instance. We partition Δ into two countable infinite disjoint sets $const(\Delta)$ and $ln(\Delta)$, and we use the first set, called *constants*, to interpret constant symbols, while the second set, called *labeled nulls*, is used to interpret existentials (see later);
- $\cdot^I$ is an interpretation function that associates:
  - to each constant symbol c, a constant $c^I \in const(\Delta)$ such that for each $c_1, c_2 \in const(\Delta)$ if $c_1 \neq c_2$ then $c_1^I \neq c_2^I$, namely we adopt the *Unique Name Assumption*. Furthermore, we require that every interpretation interprets constants in the same way, that is, given any two interpretations I and I', we have that $c_n^I = c_n^{I'}$ for each constant symbol c_n. Thus, we blur the distinction between constant symbols and constants in $const(\Delta)$;
  - to each m-ary relation symbol R_i a finite m-ary relation ${R_i}^I \subseteq \Delta^m$.

Intuitively, an artifact data instance is alike a relational database instance, since the function $\cdot^I$ lists all tuples belonging to each relation.

An expression $R_i(d_1, \ldots, d_m)$ is called (with a little abuse of terminology) a *fact*. We say that a fact belongs to an interpretation I iff $\vec{d} = \langle d_1, \ldots, d_m \rangle \in R_i^I$, so we can characterize the interpretation function $\cdot^I$ through the set of its facts (notice that such a set is finite). Following the database literature [4], we call *active domain* $\bar{\Delta}_I$ of an instance I the set of domain elements appearing in the facts of I.

Example 2.1. Consider a scenario that concerns a bank which provides services to its customers, such as loans or money transfers. Every service the institution provides has a distinct cost, that has to be paid in advance by customers that asked for it. A customer may inquire for the provision of a service: the service first has to be approved by a supervisor, then it is paid by the customer, and finally it is provisioned by the bank. Moreover, there are special "premier customers" that do not need supervisor approval. The artifact schema $\mathcal{S}$ consists of the following relation symbols:

- Customer($\underline{custSsn}$, $name$), which contains customers information;
- Service($\underline{servCode}$, $cost$), which contains information about the different types of services that the bank offers to its customers;
- ServiceClaimed($\underline{servCode}$, $custSsn$), which keeps track of information of services requested by clients;
- Examined($\underline{servCode}$, $spvName$, $outcome$), which contains the names of supervisors in charge of evaluating customers' claims;
- Payment($\underline{servCode}$, $custSsn$, $amount$), which contains information about ser-

vice payments;
- ServiceProvided($\underline{servCode}$, $custSsn$), which holds the services which have been provided;
- PremierMember($\underline{custSsn}$), which contains the customers that reach the "premier" status;
- Account($\underline{accId}$, $custSsn$, $maxWithdrawal$, $creditCard$), which holds information about bank accounts.

As artifact data instance we have an instantiation of the above relations. Namely, an instance I_0 can be:

- $\mathsf{Customer}^{I_0} = \{\langle 337505, JohnSmith\rangle, \langle 125232, MaryStewart\rangle\}$, and
- $\mathsf{Service}^{I_0} = \{\langle L057, 100\rangle, \langle L113, 150\rangle, \langle C002, 50\rangle\}$,

and all other relations are empty. □

In order to query data instances, we use *conjunctive queries* that are a special class of first-order formulas, widely used in databases, corresponding to relational algebra select-project-join queries. Formally, a *conjunctive query* is a formula cq of the form:

$$\exists \vec{y}.body(\vec{y}, \vec{x})$$

where $body$ is a conjunction of atoms (i.e., atomic formulas) involving constant symbols, existentially quantified variables $\vec{y}$ and free variables $\vec{x}$.

Intuitively, a conjunctive query returns, as answer, the domain elements (both constants and nulls) that, when substituted to the free variables, make the formula true in the instance. More formally, given an artifact instance $I = \langle \Delta, \cdot^I \rangle$, the *answer to a conjunctive query* $cq(\vec{x})$ with free variables $\vec{x}$, over I, denoted by $cq(\vec{x})^I$ is defined as:

$$cq(\vec{x})^I = \{\eta \mid \langle I, \eta\rangle \models cq(\vec{x})\}$$

where $\eta : \vec{x} \to \Delta$ is an assignment for the free variables. In fact, as usual in the database literature [4], we see assignments η simply as tuples of domain elements to be substituted to the free variables.

The notion of homomorphism [20] indeed plays a key role in our setting, so we remind its definition here. Given two instances $I_1 = \langle \Delta, \cdot^{I_1} \rangle$ and $I_2 = \langle \Delta, \cdot^{I_2} \rangle$ over the same schema $\mathcal{S}$, a *homomorphism* from I_1 to I_2, denoted by $h : I_1 \to I_2$, is a function from Δ to Δ such that:

(1) for every constant $c \in const(\Delta)$, we have that $h(c) = c$;
(2) for every $\langle d_1, \ldots, d_m\rangle \in R_i^{I_1}$, we have that $\langle h(d_1), \ldots, h(d_m)\rangle \in R_i^{I_2}$.

Two instances I_1 and I_2 are *homomorphically equivalent*, written $I_1 \overset{h}{=} I_2$, if there exist two homomorphisms $h_1 : I_1 \to I_2$ and $h_2 : I_2 \to I_1$.

A homomorphism $h : I_1 \to I_2$ preserves the interpretation of constants $const(\Delta)$ but not of labeled nulls $ln(\Delta)$ in I_1, which are mapped in a non-injective way, either to constants or nulls, in I_2. In other words, a homomorphism interprets nulls of I_1 as existential values.

The characterizing property of conjunctive queries from the semantical point of view is that they are invariant under homomorphic equivalence [4]. That is, if two data instances I and I' are homomorphic, then each boolean (without free variables) conjunctive query cq produces exactly the same (boolean) answer: $cq(\vec{x})^I = cq(\vec{x})^{I'}$.

The existential interpretation of labeled nulls given by homomorphisms suggests a different way of answering conjunctive queries, that essentially sees the set of facts in the interpretation as a *theory* where all nulls are treated as existential variables. To make this notion precise, given an interpretation I we define the (*infinite*) set W_I of all interpretations $I' = \langle \Delta, \cdot^{I'} \rangle$ over $\mathcal{S}$ such that there exists an homomorphism $h : I \to I'$. Then we define the *Certain Answers* of a conjunctive query cq as:

$$cert^I(cq) = \bigcap_{I' \in W_I} cq^{I'}$$

that is, the certain answers to a query are all those tuples of elements in I that are returned by the query in every interpretation I' such that there exists an homomorphism $h : I \to I'$. It is easy to see that such tuples can only be formed by constants in $const(\Delta)$ that appear in the active domain of I, since these are the only elements in the answers that are preserved by homomorphism. Intuitively, when using certain answers we consider the current instance as representative of several possible instances, and therefore, we return the tuples that make the query true in all such instances. Alternatively, it can be shown that the certain answers correspond to the tuples of constants such that, when substituted to the free variables of the query, would make the resulting query logically implied by the theory constituted by a single conjunctive query formed by the logical AND of all facts in I, considering all labeled nulls as existentially quantified variables.

In our framework, we assume that the user can pose arbitrary conjunctive queries to the current instance, but require them to be evaluated through certain answers. In this way, we become independent of the particular null values occurring in the data instance, since they are not returned as answers, though they can still be used as witnesses of existentially quantified variables. On the other hand, when we evolve the artifact by executing a task, we do consider null values in the current instance as legitimate elements to be propagated to the next state according to the task effects.

Artifact tasks. Executing a task over an instance results in a new instance, which is specified by the task effects. The formalization of an effect is borrowed from the database and data exchange literature, in particular from the notion of *tuple generating dependencies* (tgds) [4,30]. An *effect specification* ξ over a schema $\mathcal{S}$ is a formula of the form:

$$\exists \vec{y}.\phi(\vec{x}, \vec{y}, \vec{c}) \to \exists \vec{w}.\psi(\vec{x}, \vec{w}, \vec{d})$$

where ϕ and ψ are conjunctions of atoms over $\mathcal{S}$; $\vec{x}$, $\vec{y}$, $\vec{w}$ denote sets of variables and $\vec{c}, \vec{d}$ denote set of constants occurring in ϕ and ψ. We call the left-hand side of ξ the *premise*, and the right-hand side the *conclusion*. Notice that both the premise and the conclusion are *conjunctive queries*. Formally, let $I = \langle \Delta, \cdot^I \rangle$ be an artifact instance over the schema $\mathcal{S}$, and $\xi = \exists \vec{y}\ \phi(\vec{x}, \vec{y}, \vec{c}) \to \exists \vec{w}\ \psi(\vec{x}, \vec{w}, \vec{d})$ an effect specification. The result of *enacting effect specification* ξ on I, is the set of facts $\xi(I)$ defined as follows:

Let $\vec{\eta} = (\exists \vec{y}\ \phi(\vec{x}, \vec{y}, \vec{c}))^I$, be the answer to the query $\exists \vec{y}\ \phi(\vec{x}, \vec{y}, \vec{c})$ in I, then for each $\eta_i \in \vec{\eta}$ we proceed as follows: for each atoms $R_i(\vec{x}, \vec{w}, \vec{d})$ occurring in ψ, we include in $\xi(I)$ a new fact $R_i^{I'}(\vec{x}, \vec{w}, \vec{d})|^{\psi}_{\eta_i}$, obtained by substituting every variable in $\vec{x}$ with the corresponding element given by the assignment η_i, and every variable in $\vec{w}$ with a fresh (not appearing elsewhere) labeled null $\ell \in ln(\Delta)$.

Intuitively, the left-hand side of the effect, acting like a query, selects domain elements, both constants and null values, from the active domain of the current instance; while the right-hand side builds the resulting instance by inserting such domain elements in the relations of its atoms, and by possibly introducing fresh labeled nulls as witnesses of the existential variables in the query.

A *task* T for a schema $\mathcal{S}$ is specified as a set $\vec{\xi} = \{\xi_1, \ldots, \xi_n\}$ of effect specifications. The result of *executing task* T on I, denoted by $I' = do(T, I)$, is a new instance $I' = \langle \Delta, \cdot^{I'} \rangle$ on the same schema $\mathcal{S}$, obtained as the union of the enactments of each effect specification. Namely $I' = \langle \Delta, \cdot^{I'} \rangle$ where the interpretation function $\cdot^{I'}$ is characterized by the facts $\bigcup_{\xi \in \vec{\xi}} \xi(I)$.

Let us make some key observations on such tasks. First, we observe that the role of the existential quantification on the two sides of an effect specification is very different. The existential quantification on the left-hand side is the usual one used in conjunctive queries, which projects out variables used only to make joins. Instead, the existential quantification on the right-hand side is used as a witness of values that should be chosen by the user when executing the effect. In other words, the choice function used for assigning witnesses to the existential variables on the right-hand side should be in the hands of the user. Here, since we do not have such a choice at hand, we introduce a fresh null, to which we assign an existential meaning through homomorphism.

The second observation is that we do not make any persistence (or frame [50]) assumption in our formalization. In principle, at every move, we substitute the whole old data instance with a new one. On the other hand, it should be clear that we can easily write effect specifications that *copy* big chunks of the old instance into the new one. For instance, $R_i(\vec{x}) \rightarrow R_i(\vec{x})$ copies the whole extension of a relation R_i.

Example 2.2. Continuing our previous example, we now turn to the available tasks. As syntactic sugar, we include some input parameters (the symbols between parentheses after the task name). In order to execute a task, its parameters must be instantiated with constants as specified by the condition-action rules that form artifact lifecycle (see below). The tasks in our domain are the following:

- ClaimService($custSsn,\ servCode$), with effects:

$$\{\, \exists x, y.\, \mathsf{Customer}(custSsn, x) \wedge \mathsf{Service}(servCode, y)\ \rightarrow \mathsf{ServiceClaimed}(servCode, custSsn),\ \mathsf{copyFrame}\}$$

This task models the choice of the customer $custSsn$ to apply for the provision of a new service of type $servCode$. Since the resulting instance is a completely

new one consisting of tuples specified by the task effects, we need to explicitly "copy" all facts that we do not require to be dropped after the task execution. This is done by effects of the form $R(x_1, \ldots, x_n) \rightarrow R(x_1, \ldots, x_n)$ for each relation $R \in \mathcal{S}$. We denote collectively such copying effects as copyFrame. Intuitively, the result of firing task Claim_service($cust\_ssn$, $serv\_code$) on an instance I results in a new instance I' that not only contains I, but also includes the new tuple ServiceClaimed($custSsn, servCode$) provided that the premise is satisfied by I, otherwise $I' = I$.

- MakePayment($custSsn$, $servCode$, $amount$) with effects:

$$\{\mathsf{ServiceClaimed}(servCode, custSsn) \rightarrow \mathsf{Payment}(servCode, custSsn, amount), \mathsf{copyFrame}\}$$

  This task models the payment operation performed by a customer for a service that has been previously requested, i.e., the resulting instance may include the tuple Payment($custSsn, servCode, amount$).
- GrantApproval($servCode$) with effects:

$$\{\exists x.\mathsf{ServiceClaimed}(servCode, x) \rightarrow \exists z.\, \mathsf{Examined}(servCode, z, \text{“}approved\text{”}), \mathsf{copyFrame}\}$$

  This task represents the approval of a service that has been requested, by including (according to its effects) the fact Examined($servCode, \ell$, “$approved$”) where ℓ is a fresh labeled null that models a possible supervisor.
- ProvideServices() with effects:

$$\{\exists v, z.\mathsf{ServiceClaimed}(x, y) \wedge \mathsf{Examined}(x, v, \text{“}approved\text{”}) \wedge \mathsf{Payment}(x, y, z) \wedge \mathsf{Service}(x, z) \rightarrow \mathsf{ServiceProvided}(x, y), \mathsf{copyFrame}\}$$

  This task models the delivery of all services that have had explicitly approved by a supervisor and that were already paid.
- QuickService() with effects:

$$\{\exists z.\mathsf{ServiceClaimed}(x, y) \wedge \mathsf{Payment}(x, y, z) \wedge \mathsf{PremierMember}(y) \rightarrow \mathsf{ServiceProvided}(x, y), \mathsf{copyFrame}\}$$

  This task delivers all the services for which the correct amount was paid and that have been requested by a premier customer.
- AwardPremierStatus() with effects:

$$\{\exists y, t, u, w, z.\, \mathsf{Customer}(x, y) \wedge \mathsf{ServiceProvided}(z, x) \wedge \mathsf{Account}(u, x, w, t) \rightarrow \mathsf{PremierMember}(x), \mathsf{copyFrame}\}$$

This task awards the premier status to all customers holding a bank account who applied for the provision of a service that had already been accepted.

□

Artifact lifecycle. The artifact lifecyle is defined in terms of condition-action rules, that specify, for every instance, which tasks can be executed. A *(condition-action) rule* for a schema $\mathcal{S}$ is an expression ϱ of the form $\pi \mapsto T$ where π is a precondition and T is a task. The precondition is a *closed* formula over $\mathcal{S}$ defined according to the following syntax:

$$\pi ::= cq \mid \neg\pi \mid \pi_1 \wedge \pi_2$$

where cq is a boolean conjunctive query. Preconditions are arbitrary boolean combinations of boolean conjunctive queries interpreted under the certain answer semantics, namely we define the semantic relation *artifact instance I logically implies precondition π*, written $I \rhd \pi$, by induction on the structure of the precondition, as follows:

$$\begin{array}{ll} I \rhd cq & \text{iff } cert^I(cq) = true \\ I \rhd \neg\pi & \text{iff } I \not\rhd \pi \\ I \rhd \pi_1 \wedge \pi_2 & \text{iff } I \rhd \pi_1 \text{ and } I \rhd \pi_2 \end{array}$$

In order to execute a task T on an instance I,

Given a condition-action rule $\pi \mapsto T$ and an instance I, if I logically implies precondition π, then the task T is executable and, if it is executed, it generates a new instance I' according to T's effects.

Observe that, while we disallow negation in task effects so as to exploit the theory of conjunctive queries, in the condition-action rules we allow for it, but in order to do so we actually require conditions to be based on certain answers of conjunctive queries. In this way in conditions, we are only composing (using the booleans) the results of the conjunctive queries, and hence two homomorphic equivalent instances are guaranteed to satisfy the same conditions. Notice also that negation in this way becomes a sort of (stratified) "negation-as-failure" [21].

Example 2.3. The *artifact lifecycle* of our running example is specified by the following condition-action rules:

$\top(CustSsn, servCode) \mapsto \mathsf{ClaimService}(CustSsn, servCode)$
$\top(CustSsn, servCode, amount) \mapsto \mathsf{MakePayment}(CustSsn, servCode, amount)$
$\top(ServCode) \mapsto \mathsf{GrantApproval}(ServCode)$
$\exists x, y, v, w.\, \mathsf{Payment}(x, y, w) \wedge \mathsf{Service}(x, w) \wedge \mathsf{RequestExamined}(x, v, \text{“}approved\text{”}) \mapsto \mathsf{ProvideServices}()$
$\exists x, y, w.\, \mathsf{Payment}(x, y, w) \wedge \mathsf{Service}(x, w) \wedge \mathsf{PremierMember}(y) \mapsto \mathsf{QuickService}()$
$\exists x, y, u, w, t.\, \mathsf{ServiceProvided}(x, y) \wedge \mathsf{Account}(u, y, w, t) \mapsto \mathsf{AwardPremierStatus}()$

Again, we use parameters (occurring as free variables above) as syntactic sugar for a much larger set of condition-action rules obtained by instantiating the parameters to constants from a finite set. For example, such a set may contain all constants from the initial data

instance of the artifact specified below, plus some extra ones used for convenience, e.g. to represent some predetermined amounts of money to be use for the parameter $amount$. □

3. Conjunctive Artifact Execution

Let us consider an *artifact* $A = \langle \mathcal{D}, \mathcal{T}, \mathcal{C} \rangle$, with data component $\mathcal{D} = \langle \mathcal{S}, I_0 \rangle$ where $\mathcal{S}$ is the artifact data schema and I_0 is the initial artifact data instance. Moreover, let $\mathcal{I}$ be the set of all possible instances over $\mathcal{S}$.

Artifact execution tree. We can describe all possible executions of an artifact A by the so-called *execution tree* of A. The *execution tree* is a tuple $\mathfrak{T}_A = \langle \Sigma, \sigma_0, L, Tr \rangle$ where Σ is the set of states (or nodes), σ_0 is the root, $L : \Sigma \to \mathcal{I}$ is a labeling of the states with data instance, and $Tr \subseteq \mathcal{I} \times \mathcal{T} \times \mathcal{I}$ is the transition relation that determines the successor nodes to the current one. We use the notation $\sigma \stackrel{T}{\Longrightarrow} \sigma'$ for $\langle \sigma, T, \sigma' \rangle \in Tr$.

The set Σ of states, its labeling L and the set Tr of transitions are defined inductively as follows:

- the root is $\sigma_0 \in \Sigma$, with $L(\sigma_0) = I_0$;
- given a state σ for each task $T \in \mathcal{T}$ such that there exists a rule $\varrho = \pi \mapsto T$ such that $L(\sigma) \rhd \pi$, add a state $\sigma'_T \in \Sigma$ with $L(\sigma'_T) = do(T, I)$ (i.e., $L(\sigma'_T)$ is the data instance obtained by applying T to $L(\sigma)$) and a transition $\sigma \stackrel{T}{\Longrightarrow} \sigma'_T$.

Notice that, in the execution tree, each state correspond to the full history that has generated it starting from the initial state, i.e., there is a correspondence between a state and the path that connects it to the root. Also, given a state σ we have one T-successor for each task T executable in $L(\sigma)$.

Observe that, in constructing the execution tree, we have a certain freedom in labeling the states, since in generating new data instance $do(T, I)$ from the current one I, we are free to choose any fresh labeled null for the existential variables in the right-hand side of the effects specifications. However, all such instances are *equivalent modulo nulls renaming*. Given two instances $I_1 = \langle \Delta, \cdot^{I_1} \rangle$ and $I_2 = \langle \Delta, \cdot^{I_2} \rangle$ over the same schema $\mathcal{S}$, a *nulls renaming* from I_1 to I_2, denoted by $r : I_1 \to I_2$ is an injective homomorphism, i.e., a function such that:

(1) for every constant $c \in const(\Delta)$ we have that $r(c) = c$;
(2) for every couple of different labeled nulls $\ell_1, \ell_2 \in ln(\Delta)$ we have that $r(\ell_1) \neq r(\ell_2)$ and
(3) for every $\langle d_1, \ldots, d_m \rangle \in R_i^{I_1}$, we have that $\langle r(d_1), \ldots, r(d_m) \rangle \in R_i^{I_2}$

Two instances I_1 and I_2 are *equivalent modulo nulls renaming*, denoted by $I_1 \stackrel{mnr}{=} I_2$ iff they are isomorphic, i.e., iff there exists a nulls renaming $r : I_1 \to I_2$ such that its inverse r^{-1} is a null renaming from I_2 to I_1. Hence, modulo nulls renaming, there exists a single execution tree $\mathfrak{T}_A$ for an artifact A.

Artifact transition systems and bisimulation. The execution tree is a special case of a so-called transition system. A *transition system* for A is a tuple $\mathfrak{A}_A = \langle \Sigma, \sigma_0, L, Tr \rangle$ where *(i)* Σ is the (possibly infinite) set of states; *(ii)* σ_0 is the initial state; *(iii)* $L : \Sigma \to \mathcal{I}$ is a labeling function that associates to each state in Σ a data instance in $\mathcal{I}$. *(iv)* $Tr \subseteq \mathcal{I} \times \mathcal{T} \times \mathcal{I}$ is the transition relation.

Not all transition systems for an artifact A represent the same behavior as the execution tree. To capture which transition systems do, we need to formally capture equivalences between transition systems. To this aim, we make use of the notion of *bisimulation* [46]. In formally detailing such a notion, we consider that the user can only query data instances through conjunctive queries, evaluated to return certain answers.

Given two transition systems for the same artifact A, $\mathfrak{A}_1 = \langle \Sigma_1, \sigma_{0,1}, L_1, Tr_1 \rangle$ and $\mathfrak{A}_2 = \langle \Sigma_2, \sigma_{0,2}, L_2, Tr_2 \rangle$, a *bisimulation* is a relation $B \subseteq \Sigma_1 \times \Sigma_2$ such that $\langle \sigma_1, \sigma_2 \rangle \in B$ implies that:

(1) for every conjunctive query cq we have that $cert^{L_1(\sigma_1)}(cq) = cert^{L_2(\sigma_2)}(cq)$;
(2) if $\sigma_1 \stackrel{T}{\Longrightarrow} \sigma_1'$ then there exists σ_2' such that $\sigma_2 \stackrel{T}{\Longrightarrow} \sigma_2'$ and $\langle \sigma_1', \sigma_2' \rangle \in B$;
(3) if $\sigma_2 \stackrel{T}{\Longrightarrow} \sigma_2'$ then there exists σ_1' such that $\sigma_1 \stackrel{T}{\Longrightarrow} \sigma_1'$ and $\langle \sigma_1', \sigma_2' \rangle \in B$.

We say that two states σ_1 and σ_2 are *bisimilar*, denoted as $\sigma_1 \sim \sigma_2$, if there exists a bisimulation B such that $\langle \sigma_1, \sigma_2 \rangle \in B$. Two transition systems $\mathfrak{A}_1 = \langle \Sigma_1, \sigma_{0,1}, L_1, Tr_1 \rangle$ and $\mathfrak{A}_2 = \langle \Sigma_2, \sigma_{0,2}, L_2, Tr_2 \rangle$ are *bisimilar* if $\sigma_{0,1} \sim \sigma_{0,2}$.

With the notion of bisimulation at hand, we can state that any transition system that is bisimilar to the execution tree represents the behavior of the artifact. We can exploit this fact to perform verification on a transition system that is more manageable than the execution tree. We will do so later: first, we introduce a suitable verification formalism.

4. Verification Formalism

We turn to verification of conjunctive artifact-centric services. To specify dynamic properties we use μ-calculus [29,54], one of the most powerful temporal logics for which model checking has been investigated, and indeed is able to express both linear time logics, as LTL, and branching time logics such as CTL or CTL* [23]. The main characteristic of μ-calculus is the ability of expressing directly least and greatest fixpoints of (predicate-transformer) operators formed using formulas relating the current state to the next one. By using such fixpoint constructs, one can easily express sophisticated properties defined by induction or co-induction. This is the reason why virtually all logics used in verification can be considered as fragments of μ-calculus. From a technical viewpoint, μ-calculus separates local properties, i.e., properties asserted on current state or states that are immediate successors of the current one, from properties that talk about states that are arbitrarily far [15]. The latter are expressed through the use of fixpoints. Such a separation is very convenient for theoretical investigation, and indeed makes μ-calculus the language of choice for much theoretical work [29]. On the other hand, from a practitioner point of view, expressing properties using directly fixpoint can be cumbersome and, in most applications, simpler logics like CTL or LTL are preferred. For a thorough introduction to μ-calculus, we refer

the reader to Stirling's book [54] which looks at μ-calculus both from the theoretical and from the practical point of view. The choice of using μ-calculus in our investigation allows for immediately transferring the results obtained to simpler logics like LTL, CTL, CTL*, etc.

Specifically, we introduce a variant of μ-calculus, called $\mu\mathcal{L}$, that conforms to the basic assumption of our formalism: the use of conjunctive queries and certain answers to talk about data instances. This intuitive requirement can be made formal as follows: $\mu\mathcal{L}$ must be invariant with respect to the notion of bisimulation introduced above.

Given an artifact $A = \langle \mathcal{S}, \mathcal{T}, \mathcal{C} \rangle$, the *verification formulas* of $\mu\mathcal{L}$ for A have the following form:

$$\Phi ::= cq \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid [T]\Phi \mid \langle T \rangle \Phi \mid \mu Z.\Phi \mid \nu Z.\Phi \mid Z$$

where cq is a boolean conjunctive query (interpreted through certain answers) over the artifact schema and Z is a predicate variable symbol.

The symbols μ and ν can be considered as quantifiers, and we make use of notions of scope, bound and free occurrences of variables, closed formulas, etc., referring to them. For formulas of the form $\mu Z.\Phi$ and $\nu Z.\Phi$, we require the *syntactic monotonicity* of Φ w.r.t. Z: Every occurrence of the predicate variable Z in Φ must be within the scope of an even number of negation signs. In μ-calculus, given the requirement of syntactic monotonicity, the least fixpoint $\mu Z.\Phi$ and the greatest fixpoint $\nu Z.\Phi$ always exist. In order to define the meaning of such formulas, we resort to interpretations that are transition systems. Let $\mathfrak{A} = \langle \Sigma, \sigma_0, L, Tr \rangle$ be a transition system for A with initial data instance I_0, and let $\mathcal{V}$ be a predicate valuation on $\mathfrak{A}$, i.e., a mapping from the predicate variables to subsets of the states in $\mathfrak{A}$. Then, we assign meaning to μ-calculus formulas by associating to $\mathfrak{A}$ and $\mathcal{V}$ an *extension function* $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$, which maps μ-calculus formulas to subsets of $\mathcal{I}$. The extension function $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$ is defined inductively as follows:

$$\begin{array}{ll}
(cq)^{\mathfrak{A}}_{\mathcal{V}} & = \{\sigma \in \Sigma \mid cert^{L(\sigma)}(cq)\} \\
(Z)^{\mathfrak{A}}_{\mathcal{V}} & = \mathcal{V}(Z) \subseteq \Sigma \\
(\neg\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \Sigma - (\Phi)^{\mathfrak{A}}_{\mathcal{V}} \\
(\Phi_1 \wedge \Phi_2)^{\mathfrak{A}}_{\mathcal{V}} & = (\Phi_1)^{\mathfrak{A}}_{\mathcal{V}} \cap (\Phi_2)^{\mathfrak{A}}_{\mathcal{V}} \\
(\langle T \rangle \Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \{\sigma \in \Sigma \mid \exists \sigma'.\ \sigma \stackrel{T}{\Longrightarrow} \sigma' \text{ and } \sigma' \in (\Phi)^{\mathfrak{A}}_{\mathcal{V}}\} \\
([T]\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \{\sigma \in \Sigma \mid \forall \sigma'.\ \sigma \stackrel{T}{\Longrightarrow} \sigma' \text{ implies } \sigma' \in (\Phi)^{\mathfrak{A}}_{\mathcal{V}}\} \\
(\mu Z.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcap\{\mathcal{E} \subseteq \Sigma \mid \quad (\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z \leftarrow \mathcal{E}]} \subseteq \mathcal{E}\ \} \\
(\nu Z.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcup\{\mathcal{E} \subseteq \Sigma \mid \quad \mathcal{E} \subseteq \ (\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z \leftarrow \mathcal{E}]}\}
\end{array}$$

Intuitively, $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$ assigns to the various constructs of μ-calculus the following meaning:

- The boolean connectives have the expected meaning.
- The extension of $\langle T \rangle \Phi$ includes the states σ such that starting from σ, there is an execution of task T that leads to a successor state σ' included in the extension of Φ.
- The extension of $[T]\Phi$ includes the states σ such that starting from σ, each execution of task T leads to some successor state σ' included in the extension of Φ.

- The extension of $\mu Z.\Phi$ is the *smallest subset* $\mathcal{E}_\mu$ of Σ such that, assigning the extension $\mathcal{E}_\mu$ to Z, the resulting extension of Φ is contained in $\mathcal{E}_\mu$. That is, the extension of $\mu X.\Phi$ is the *least fixpoint* of the operator $\lambda\mathcal{E}.(\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z\leftarrow\mathcal{E}]}$ (here $\mathcal{V}[Z \leftarrow \mathcal{E}]$ denotes the predicate valuation obtained from $\mathcal{V}$ by forcing the valuation of Z to be $\mathcal{E}$).
- Similarly, the extension of $\nu Z.\Phi$ is the *greatest subset* $\mathcal{E}_\nu$ of Σ such that, assigning the extension $\mathcal{E}_\nu$ to Z, the resulting extension of Φ contains $\mathcal{E}_\nu$. That is, the extension of $\nu Z.\Phi$ is the *greatest fixpoint* of the operator $\lambda\mathcal{E}.(\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z\leftarrow\mathcal{E}]}$.

In expressing temporal properties using $\mu\mathcal{L}$ below, we use the following abbreviations: $\langle-\rangle\Phi \doteq \bigvee_{T\in\mathcal{T}}\langle T\rangle\Phi$ and $[-]\Phi \doteq \bigwedge_{T\in\mathcal{T}}[T]\Phi$, where $\mathcal{T}$ is the set of all tasks of the artifact. In this way, we can talk about *all next states* (resulting from every possible task execution) or about *some next states* resulting from certain task executions.

With this abbreviations at hand, it is easy to express natural temporal properties such as "eventually a local property ϕ holds in all runs" (a *liveness* property):

$$\mu Z.\phi \vee [-]Z$$

or "always a local property ϕ holds (in all runs)"(a *safety* property) :

$$\nu Z.\phi \wedge [-]Z$$

By the way, notice that the negation of $\nu Z.\phi \wedge [-]Z$ is not $\mu Z.\phi \vee [-]Z$ but instead $\mu Z.\phi \vee \langle-\rangle Z$ which expresses that eventually ϕ holds along some (but not necessarily all) runs.

Coming back to the first formula, $\mu Z.\phi \vee [-]Z$ can be seen as the "smallest solution" of the equation: $Z = \phi \vee [-]Z$ that is the smallest predicate that substituted to the variable Z makes the equation true. More formally $\mu Z.\phi \vee [-]Z$ denotes in every interpretation (i.e., transition system) $\mathfrak{A}$, the least fixpoint of the operator $\lambda\mathcal{E}.(\phi \vee [-]Z)^{\mathfrak{A}}_{\mathcal{V}[Z\leftarrow\mathcal{E}]}$, i.e., the smallest set $\mathcal{E}_\mu$ of states of $\mathfrak{A}$ that makes the equation $\mathcal{E}_\mu = (\phi \vee [-]Z)^{\mathfrak{A}}_{\mathcal{V}[Z\leftarrow\mathcal{E}_\mu]}$ true. Similarly $\nu Z.\phi \wedge [-]Z$ can be seen as the "greatest solution" of the equation $Z = \phi \wedge [-]Z$, or more precisely, in every interpretation $\mathfrak{A}$ the greatest fixpoint of the operator $\lambda\mathcal{E}.(\phi \wedge [-]Z)^{\mathfrak{A}}_{\mathcal{V}[Z\leftarrow\mathcal{E}]}$, i.e., the greatest set $\mathcal{E}_\nu$ of states of $\mathfrak{A}$ that makes the equation $\mathcal{E}_\nu = (\phi \wedge [-]Z)^{\mathfrak{A}}_{\mathcal{V}[Z\leftarrow\mathcal{E}_\nu]}$ true.

The reasoning problem we are interested in is *model checking*: verify whether a $\mu\mathcal{L}$ *closed formula* Φ *holds in an artifact* A *with initial data instance* I_0. Formally, such a problem is defined as checking whether $L(\sigma_0) \in I \in (\Phi)^{\mathfrak{T}^{I_0}_A}_{\mathcal{V}}$ (where $\mathcal{V}$ is any valuation, since Φ is closed), that is, whether Φ is true in the root of the A execution tree.

On the other hand, we know that there are several transition systems that are bisimilar to the execution tree $\mathfrak{T}^{I_0}_A$. The following theorem states that the formula evaluation in $\mu\mathcal{L}$ is indeed invariant w.r.t. bisimilarity, so we can equivalently check any such transition system.

Theorem 4.1. Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be two bisimilar transition systems. Then, for every pair of states σ_1 and σ_2 such that $\sigma_1 \sim \sigma_2$ (including the initial ones), for all formulas Φ of $\mu\mathcal{L}$, we have that $\sigma_1 \in (\Phi)^{\mathfrak{A}_1}_{\mathcal{V}}$ iff $\sigma_2 \in (\Phi)^{\mathfrak{A}_2}_{\mathcal{V}}$.

Proof. The proof is analogous to the standard proof of bisimulation invariance of μ-calculus [15], though taking into account our specific definition of bisimulation, which makes

use of conjunctive queries and certain answers as their evaluation.

□

In particular, if for some reason we can get a transition system that is bisimilar to the execution tree and is *finite*, then we can apply the following theorem.

Theorem 4.2. Checking a $\mu\mathcal{L}$ formula Φ over a finite transition system $\mathfrak{A}_A = \langle \Sigma, \sigma_0, L, Tr \rangle$ can be done in time

$$O((|\mathfrak{A}| \cdot |\Phi|)^k)$$

where $|\mathfrak{A}| = |\Sigma| + |Tr|$, i.e., the number of states plus the number of transitions of $\mathfrak{A}$, $|\Phi|$ is the size of formula Φ (in fact, considering conjunctive queries as atomic), and k is the number of nested fixpoints, i.e., fixpoints whose variables are one within the scope of the other.

Proof. It suffices to use the standard μ-calculus model checking algorithms [29], with the proviso that for atomic formulas we use the computation of certain answers of conjunctive queries. □

Example 4.1. Let us consider again our running example with initial artifact data instance I_0 of Example 2.1 where $\mathsf{Customer}^{I_0} = \{\langle 337505, JohnSmith \rangle, \langle 125232, MaryStewart \rangle\}$, $\mathsf{Service}^{I_0} = \{\langle L057, 100 \rangle, \langle L113, 150 \rangle, \langle C002, 50 \rangle\}$, and all other relations are empty. The following liveness property asks if it is possible to obtain the provision of any service at all, i.e., if by executing tasks we can eventually get to a state where some service has been provided:

$$\mu Z.\ (\exists x, y.\mathsf{ServiceProvided}(x, y)\ \vee \bigvee_{T \in \mathcal{T}} \langle - \rangle Z)$$

The formula is actually true, for example a state where $\mathsf{ServiceProvided}(L057, 337505)$ holds can be reached from the initial state through the following sequence of tasks: $\mathsf{ClaimService}(337505, L057)$, $\mathsf{MakePayment}(337505, L057, 100)$, $\mathsf{GrantApproval}(L057)$ and finally $\mathsf{ProvideServices}()$.

Next, consider the safety property asking whether every possible reachable instance will always contain the information that the service $L113$ has been paid and provided:

$$\nu Z.\ (\exists x, y, z.\mathsf{Payment}(L113, x, y) \wedge\ \mathsf{ServiceProvided}(L113, z)\ \wedge \bigwedge_{T \in \mathcal{T}} [-](Z))$$

This is trivially false, since in the initial instance I_0 there is no payment for any service.

As a last example, we look at a fairness property, expressing that it is always true that eventually a service is provided:

$$\nu Z_1.(\mu Z_2.((\exists x_1, x_2, x_3.\mathsf{Service}(x_1, x_2) \wedge \mathsf{ServiceProvided}(x_1, x_3)) \vee \langle - \rangle Z_2) \wedge [-] Z_1)$$

This is not the case, because there is an (infinite) path in the execution tree, e.g. the one obtained by repeating forever action $\mathsf{GrantApproval}(L113)$, that passes through states in which $\exists x_1, x_2, x_3.\mathsf{Service}(x_1, x_2, x_3) \wedge \mathsf{ServiceProvided}(x_1, x_3)$ will never hold.

More sophisticated temporal properties, such as strong forms of fairness, are also easily expressible in $\mu\mathcal{L}$. □

5. Decidability of Weakly Acyclic Conjunctive Artifacts

In this section we study decidability of verification in conjunctive artifacts. First, observe that, so far, we do not have a concrete technique for the verification problem, since the model checking results in Theorem 4.2 only apply to finite structures. In fact, as a consequence of the undecidability of the implication problem for tgds (see e.g. [4]), it is obvious that, without any restrictions on effect specifications, model checking in our setting is undecidable. Addressing sufficient conditions for decidability is the purpose of this section. We start by introducing the notion of execution transition system and showing its relationship with the notion of execution tree of an artifact.

Execution transition system. Given an artifact $A = \langle \mathcal{D}, \mathcal{T}, \mathcal{C} \rangle$ with initial artifact data instance I_0, we define the *execution transition system* $\mathfrak{S}_A = \langle \Sigma_s, \sigma_{0,s}, L_s, Tr_s \rangle$ inductively as follows:

- $\sigma_{0,s} \in \Sigma$ and such that $L_s(\sigma_{0,s}) = I_0$;
- for all instances $\sigma \in \Sigma_s$ and for each task $T \in \mathcal{T}$ such that there exists a rule $\varrho = \pi \mapsto T$ such that $L_s(\sigma) \rhd \pi$, let $I' = do(T, L_s(\sigma))$ be data instance resulting from the execution of task T in $L_s(\sigma)$ then:
  - if there exists an instance $\sigma' \in \Sigma_s$ such that $L_s(\sigma') \overset{h}{=} I'$ then add the transition $\sigma \overset{T}{\Longrightarrow} \sigma'$ to Tr_s;
  - if such a state does not exists, then add the a new state $\sigma_{I'}$ to Σ with $L_s(\sigma_{I'}) = I'$ and add the transition edge $\sigma \overset{T}{\Longrightarrow} \sigma_{I'}$ to Tr_s.

Theorem 5.1. Let $A = \langle \mathcal{D}, \mathcal{T}, \mathcal{C} \rangle$ be an artifact with initial data instance I_0. Then, the execution tree $\mathfrak{T}_{A,I_o} = \langle \Sigma_t, \sigma_{0,t}, L_t, Tr_t \rangle$ is bisimilar to the execution transition system $\mathfrak{S}_{A,I_0} = \langle \Sigma_s, \sigma_{0,s}, L_s, Tr_s \rangle$.

Proof. Let us consider the bisimulation relation $B_{ts} = \{\langle \sigma_t, \sigma_s \rangle \mid \sigma_t \in \Sigma_t \wedge \sigma_s \in \Sigma_s \wedge L_s(\sigma_s) \overset{h}{=} L_t(\sigma_t)\}$. This is the relation formed by couples of states of the two transition systems such that their labeling data instances are homomorphically equivalent. We show that B_{ts} is a bisimulation (according to our definition). Indeed consider $\langle \sigma_s, \sigma_t \rangle \in B_{ts}$. Then:

(1) For each cq, since $L_s(\sigma_s) \overset{h}{=} L_t(\sigma_t)$ we have that $cert^{L_s(\sigma_s)}(cq) = cert^{L_t(\sigma_t)}(cq)$ from the definition of certain answers and homomorphical equivalence.
(2) If $\sigma_t \overset{T}{\Longrightarrow} \sigma'_t$ then there is a rule $\varrho = \pi \mapsto T$ and $L_t(\sigma_t) \rhd \pi$. Since $L_s(\sigma_s) \overset{h}{=} L_t(\sigma_t)$ then *(i)* $L_s(\sigma_s) \rhd \pi$ as well, so $\sigma_s \overset{T}{\Longrightarrow} \sigma'_s$ moreover it is easy to see that $L_t(\sigma'_t) \overset{h}{=} L_s(\sigma'_s)$ by considering definition of executing a task.
(3) Symmetric to the previous case.

Finally observe that since $L_t(\sigma_{0,t}) = L_s(\sigma_{0,s}) = I_0$ we trivially get that $\langle \sigma_{0,s}, \sigma_{0,t} \rangle \in B_{ts}$. □

This theorem basically allow us to make use of the execution transition system rather than an execution tree for our verification tasks, taking advantage of Theorem 4.1. In other words, the certain answer semantics give us the freedom of using *equivalence classes* of homomorphically equivalent instances for the purpose of verification. Notice, however, that this theorem is not sufficient to achieve a decidability result, since the number of states in the execution transition system is bounded only by the number of homomorphically non-equivalent data instances, which is infinite in general. In the following, we concentrate on conditions that guarantee its finiteness.

Inflationary approximate. Artifacts can both increase and decrease the size of the data stored in the data component as tasks are executed. For the development below, it is convenient to disregard the possibility of erasing data, so as to have a sort of abstraction of the original artifact in which the information monotonically increases only. To do so we introduce what we call here the *inflationary approximate* of an artifact, which is indeed a variant of the original one in which, essentially, information only increases. Notice that we are not interested in any way to the actual behavior, i.e., transition system of the inflationary approximate. We are interested only in the fact that the inflationary approximate gives us an upper bound on the data instances constituting the state of the transition system of the original artifact. In particular, if such a bound is finite, we get that also the states of the original transition system are finite, and hence finite state model checking techniques can be applied for the verification of the original artifact.

Given an artifact $A = \langle \mathcal{D}, \mathcal{T}, \mathcal{C} \rangle$ let us introduce $A^+ = \langle \mathcal{D}, \mathcal{T}^+, \top \rangle$, the *inflationary approximate* of the artifact A, that differs from A in the fact that every effect $T^+ \in \mathcal{T}^+$ copies all the fact and because the rules are of the form $\varrho = \top \mapsto T^+$ for every task T^+, namely it is always possible to execute a task. Let $\mathcal{I}$ be the set of possible interpretation over $\mathcal{S}$, in the following we will make use of two different functions: the first one, $\mathsf{f} : \mathcal{T} \times \mathcal{I} \to \mathcal{I}$, is defined as $\mathsf{f}(T, I) = do(T, I)$, so it computes the usual result of executing a task on I; while the second one, $\mathsf{g} : \mathcal{T}^+ \times \mathcal{I} \to \mathcal{I}$, is the inflationary approximate of the first one: $\mathsf{g}(T^+, I) = do(T^+, I)$, that is, it generates the result of executing the inflationary task T^+ on I. Notice that no contradiction can arise since effects of tasks, being based on conjunctive queries, are only positive. In this setting, the need of comparing instances that disagree on names of labeled nulls raises. To do so we make use again of the notion of *nulls renaming* introduced in Section 3.

Given two instances I_1 and I_2, we say that I_1 *is contained in* I_2 *modulo nulls renaming*, written as $I_1 \overset{mnr}{\subseteq} I_2$, if there exists a nulls renaming $r : I_1 \to I_2$.

Let $A = \langle \mathcal{D}, \mathcal{T}, \mathcal{C} \rangle$ be an artifact, and $A^+ = \langle \mathcal{D}, \mathcal{T}^+, \top \rangle$ be its inflationary approximate, we introduce, for every task T^+, $I_{good} = \{I_0 \cup \mathsf{g}(T^+, I) \mid I \subseteq I_{good}\}$. Now we define the instance $I_{I_0}^{max} = \bigcap \{I_{good} \mid I_{good} = \{I_0 \cup \mathsf{g}(T^+, I) \mid I \subseteq I_{good}\}\}$.

Notice also that, as an immediate consequence of its definition, we get that $\mathsf{g}(T, I_{I_0}^{max}) = I_{I_0}^{max}$, since $I_{I_0}^{max}$ is a *fixpoint*, indeed, the *least fixpoint* [55].

Lemma 5.1. *Let $A = \langle \mathcal{D}, \mathcal{T}, \mathcal{C} \rangle$ be an artifact with initial data instance I_0, and let $I_{I_0}^{max}$ be as above. Then for every task T^+ and for every sequence of instances $I_0, \ldots, I_n$ such that $I_{i+1} = \mathsf{g}(T^+, I_i)$, we have that $I_i \overset{mnr}{\subseteq} I_{I_0}^{max}$, for $i = 0, \ldots, n$.*

Proof. By induction on the number i of application of the g function.
Base case: trivial, by definition of $I_{I_0}^{max}$ we have that $I_0 \overset{mnr}{\subseteq} I_{I_0}^{max}$.
Inductive case: we show that for every task T, if $I_{i+1} = \mathsf{g}(T, I_i)$, then $I_{i+1} \overset{mnr}{\subseteq} I_{I_0}^{max}$. Recalling that g is inflationary, we have that it is also *monotonically increasing*, namely, for every task T and instance I, we have that $I \subseteq \mathsf{g}(T, I)$. Since by inductive hypothesis $I_i \overset{mnr}{\subseteq} I_{I_0}^{max}$, we get that for every task T^+, $I_{i+1} = \mathsf{g}(T^+, I_i) \overset{mnr}{\subseteq} I_{I_0}^{max}$ holds.

□

Lemma 5.2. *Let $A = \langle \mathcal{D}, \mathcal{T}, \mathcal{C} \rangle$ be an artifact with initial data instanceI_0, and let $I_{I_0}^{max}$ be as above. Then for every task $T \in \mathcal{T}$ and for every sequence of instances $I_0, \ldots, I_n$, such that $I_{i+1} = \mathsf{f}(T, I_i)$, we have that $I_i \overset{mnr}{\subseteq} I_{I_0}^{max}$, for $i = 0, \ldots, n$.*

Proof. By induction on the length n of application of the f function.
Base case: trivial, by definition of $I_{I_0}^{max}$ we have that $I_0 \overset{mnr}{\subseteq} I_{I_0}^{max}$.
Inductive case: for the sake of readability, we split the proof in two parts: we first show that (1) for every task T and couple of instances I_f, I_g such that $I_\mathsf{f} \overset{mnr}{\subseteq} I_\mathsf{g}$, we have that $\mathsf{f}(T, I_\mathsf{f}) \overset{mnr}{\subseteq} \mathsf{g}(T, I_\mathsf{g})$ and then that (2) for every task T, if $I' = \mathsf{f}(T, I)$, then $I' \overset{mnr}{\subseteq} I_{I_0}^{max}$. Starting from (1), for every task $T \in \mathcal{T}$, let us use $I'_\mathsf{f} = \mathsf{f}(T, I_\mathsf{f})$ and $I'_\mathsf{g} = \mathsf{g}(T^+, I_\mathsf{g})$, we show how to construct a nulls renaming $r : I'_\mathsf{f} \to I'_\mathsf{g}$. Notice that $I'_\mathsf{f} = I'_{\mathsf{f},old} \cup I'_{\mathsf{f},new}$, namely, I'_f is made up by some facts that may be copied from I_f, and some new facts. On the other hand, $I'_\mathsf{g} = I_\mathsf{g} \cup I'_{\mathsf{g},new}$. Since, $I'_{\mathsf{f},old} \overset{mnr}{\subseteq} I_\mathsf{g}$ we have, by definition, that $r : I'_{\mathsf{f},old} \to I_\mathsf{g}$. We now extend r in order to cover $I'_{\mathsf{f},new}$. Any new fact $R_i^{I'_\mathsf{f}}(\vec{x}, \vec{w}, \vec{d})|^{\psi}_{\eta_i} \in I'_{\mathsf{f},new}$ comes from an effect specification $\exists \vec{y}.\phi(\vec{x}, \vec{y}, \vec{c}) \to \exists \vec{w}.\psi(\vec{x}, \vec{w}, \vec{d})$, and since the set $\vec{\eta_\mathsf{f}} = (\exists \vec{y}.\psi(\vec{x}, \vec{y}, \vec{c}))^{I_\mathsf{f}}$ are computed over the instance $I_\mathsf{f} \overset{mnr}{\subseteq} I_\mathsf{g}$, we have that (with abuse of notation) $\vec{\eta_\mathsf{f}} \overset{mnr}{\subseteq} \vec{\eta_\mathsf{g}}$ with $\vec{\eta_\mathsf{g}} = (\exists \vec{y}.\psi(\vec{x}, \vec{y}, \vec{c}))^{I_\mathsf{g}}$. Hence, for every value $\ell_{i,\mathsf{f}}$ introduced in $I'_{\mathsf{f},new}$ in a certain position of the schema, we have a correspondent value $\ell_{i,\mathsf{g}}$ in $I'_{\mathsf{g},new}$ in the same position. We then extend the identity r with $r(\ell_{i,\mathsf{f}}) = \ell_{i,\mathsf{g}}$ for each i and $r(d_j) = d_j$ for every new constant $d_j \in \vec{d}$ introduced by effects. It is easy to verify that $r : I'_\mathsf{f} \to I'_\mathsf{g}$ is indeed a nulls renaming by construction. We now prove (2): by inductive hypothesis we have $I \overset{mnr}{\subseteq} I_{I_0}^{max}$, from Lemma 5.1 we have that, for evert task $T \in \mathcal{T}$, $\mathsf{g}(T^+, I) \overset{mnr}{\subseteq} I_{I_0}^{max}$, from (1) that $\mathsf{f}(T, I) \overset{mnr}{\subseteq} \mathsf{g}(T, I)$, and therefore, by transitivity, we get the claim.

□

We have thus showed that the data instances of inflationary approximate bounds the data instances of the original artifact. The next step is to find conditions that guarantee

finiteness of the data instances themselves. To do so, we resort to the literature on boundedness of data exchange and the condition of weak acyclicity defined there. Before continuing, we briefly summarize such notions in the paragraph below.

Boundedness of data exchange and weak acyclicicy. The *data exchange* problem address the issue of translating and restructuring data from one logical schema, called *source schema* to a new one, the *target schema*. Technically, the source and the target schema are related through a set of dependencies, called source-to-target dependencies, that, intuitively, formalize how to restructure data in the "new" schema, while the so-called set of target-to-target dependencies is used to represent constraints on the target schema. Both these dependencies have the form of containment (or implication) between conjunctive queries. Dependencies of this form are called *tuple-generating dependencies*, or *tgds*. The problem of data exchange is then the following: given an instance of the source schema, materialize an instance over the target schema by *chasing*, i.e., recursively applying all tgds as many times as possible. However, in principle there is no guarantee that chasing will ever finish. In fact, roughly speaking, tgds generate databases that include new "unknown" values (i.e., labelled nulls). E.g., a dependency may express the constraint on the new database that "Every employee is involved in *a* project" without telling us which project. Clearly, problems arise when such labelled null values are used for generating new ones, therefore creating a sort of loop that makes the resulting instance infinite. In order to avoid this obstacle, restriction on the form of tgds allowed have been proposed, so as to enforce the so-called *weak acyclicity*, meaning that, intuitively, dependencies should not generate values in a cyclic way, hence guaranteeing the termination of the chase and a finite resulting instance [30]. Notice that weak acyclicity is only a sufficient condition to obtain such a result, and lately several generalization of the condition have been proposed [1,28,44,45]. In this paper we stick to the original definition of weak acyclicity for simplicity, but we stress that all the results that we are presenting hold also for more general conditions that guarantee the termination of the chase and the finiteness of the resulting instance.

Weakly acyclic artifacts. After this intermezzo, we are now ready to define sufficient conditions on artifact, corresponding to the above notion of weak acyclicity, that guarantee that the instances of inflationary approximate, and thus the original artifact, are indeed finite.

Roughly speaking, the above lemmas guarantee that every possible instance that can be produced from I_0 by applying in every possible way f and g functions is bounded by the least fixpoint $I_{I_0}^{max}$. Notice however that $I_{I_0}^{max}$ is infinite in general, so, in order to get decidability, we need a finite bound on $I_{I_0}^{max}$. To get such condition we exploit results from [30] on *weakly-acyclic* tgds. Weak-acyclicity is a syntactic notion that involves the so-called *dependency graph* of the set of tgds TG. Informally, a set TG of tgds is weakly-acyclic if there are no cycles in the dependency graph of TG involving "existential" relation positions. The key property of *weakly-acyclic* tgds is that chasing a data instance with them (i.e., applying them in all possible way) generates a set of facts (a database) that is finite. Formally, given an artifact $A = \langle \mathcal{D}, \mathcal{T}, \mathcal{C} \rangle$, the dependency graph (that is a directed

graph) is constructed as follows: (1) for every relation symbol $R_i \in \mathcal{S}$ there is a node (called position) for every pair (R_i, att) where att is an attribute in R_i and (2) add edges as follows: for every action $\xi = \exists y.\phi(\vec{x}, \vec{y}, \vec{c}) \rightarrow \exists w.\psi(\vec{x}, \vec{w}, \vec{d})$ and for every $x \in \vec{x}$ that occurs in ψ: for every occurrence of x in ϕ in position p:

- for every occurrence of x in ψ in position p' add an edge $p \rightarrow p'$ (if it does not already exist);
- in addition, for existential variable $w_i \in \vec{w}$ and for every occurrence of w_i in ψ in position p'' add a *special edge* $p \xrightarrow{*} p''$ (if it does not already exists).

We say that A is weakly acyclic if the dependency graph of the effect specifications in $\mathcal{T}$ contains no cycles going trough special edges. Notice that if A is weakly acyclic, its inflationary approximate A^+ is weakly acyclic as well.

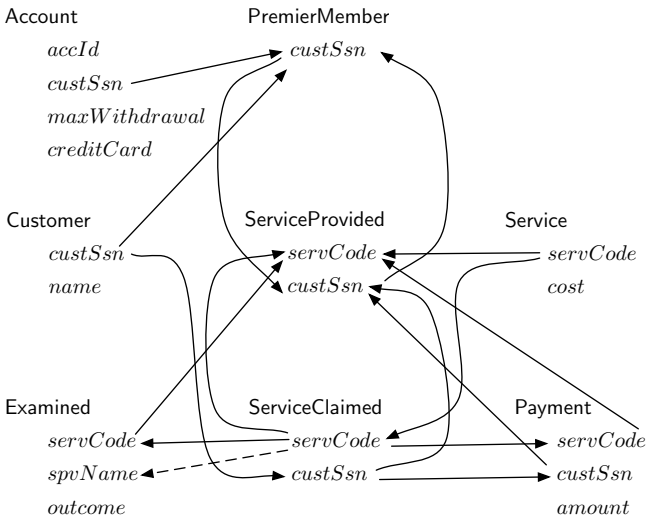

Figure 1. Dependency graph of the running example.

Example 5.1. It is easy to see that the artifact in our running example is weakly acyclic. To verify this, we build the dependency graph associated to it, shown in Figure 5, and we check that there are no cycles going through special edges. In our case there is a single special edge, which is denoted by a dashed arrow, and indeed such an edge is not involved in any cycle. □

We would like to exploit the result of [30]. In order to do so, we show that inflationary executing a task is equivalent to a sequence of chase steps. Here we give a brief definition of chase step (more details in [30]). We first define the notion of homomorphism from a conjunctive formula $\exists y.\phi(\vec{x}, \vec{y}, \vec{c})$ to an instance I as a mapping h from the variables $\vec{x} \cup \vec{y}$ to $const(\Delta) \cup ln(\Delta)$ such that for every atom $R_i(x_1, \ldots, x_n)$ of ϕ, the fact

$R_i(h(x_1), \ldots, h(x_n))$ is in I. Now we are ready to define a *chase step*: let I be an instance, $\xi = \exists y.\phi(\vec{x}, \vec{y}, \vec{c}) \rightarrow \exists w.\psi(\vec{x}, \vec{w}, \vec{d})$ an effect specification, i.e., a tgd, and h an homomorphism from $\exists y.\phi(\vec{x}, \vec{y}, \vec{c})$ to I such that there is no extension of h to an homomorphism h' from $\exists y.\phi(\vec{x}, \vec{y}, \vec{c}) \wedge \exists w.\psi(\vec{x}, \vec{w}, \vec{d})$ to I. We say that ξ *can be applied to* I *with homomorphism* h. Let I' be the union of I with the set of facts obtained by: (a) extending h to h' such that each variable in y is assigned a fresh labeled null, followed by (b) taking the image of the atoms of ψ under h'. We say that the result of applying ξ to I with h is I', and we write $I \xrightarrow{\xi,h} I'$. To conclude, a *chase sequence* of I_0 with a set $\vec{\xi}$ of effect specifications, namely, tgds, is a sequence (finite or infinite) of chase step $I_i \xrightarrow{\xi_i,h_i} Ii+1$ with $i = 0, 1, \ldots$ and $\xi_i \in \vec{\xi}$.

We are now ready to define a correspondence between the enactment of a task and a chase sequence.

Lemma 5.3. *Let A be an artifact, $A^+ = \langle \mathcal{D}, \mathcal{T}^+, \top \rangle$ its inflationary approximate, and I, J two instances such that $I \overset{mnr}{=} J$. For every task $T^+ \in \mathcal{T}^+$ if $I \overset{T^+}{\Longrightarrow} I'$, then there exists a chase sequence (possibly empty) $J \xrightarrow{\xi_i,h_i}, \ldots, \xrightarrow{\xi_j,h_j} J'$ with $\xi_i, \ldots, \xi_j \in \mathcal{T}^+$ such that $I' \overset{mnr}{=} J'$.*

Proof. Recall that enacting a task in $T^+ \in \mathcal{T}^+$ at state σ such that $L_s(\sigma) = I$, that is, execute the $\mathsf{g}(T^+, I)$ function, is inflationary, and so is the chase step: so, the resulting instance, say I', is $I' = I \cup I'_{new}$ where I'_{new} is the set of "new" facts just added by T^+. Nevertheless, I'_{new} may be the empty set. In this case, the enactment of a task can still be performed (resulting in $\mathsf{g}(T^+, I) = I$) while the chase step cannot be performed (no chase step involving tgds in T^+ can be performed, since every homomorphism from ϕ to J can be extended from $\phi \wedge \psi$). This is a simple consequence of the definition of chase step, g function and execution transition system. But if $I'_{new} = \emptyset$, no new facts are added, and so no chase step are needed. Stepping back to the general and more interesting case, by definition of execution of a task, I'_{new} is made up by facts of the form $\psi_i^{I'}(\vec{x}, \vec{w}, \vec{d})|_{\eta_{i,j}}$ for each effect specification $\xi_i = \exists \vec{y}.\phi_i(\vec{x}, \vec{y}, \vec{c}) \rightarrow \exists \vec{w}.\psi_i(\vec{x}, \vec{w}, \vec{d})$ in T^+ and each assignment $\eta_{i,j} \in \vec{\eta_i}$ where $\vec{\eta_i} = \exists \vec{y}.\phi_i(\vec{x}, \vec{y}, \vec{c})^I$. Since $I \overset{mnr}{=} J$, there exists $r : I \rightarrow J$. For each $\eta_{i,j}$ let us consider the function $h_{i,j}$ built in this way: *(i)* for each $x \in \vec{x}$ (resp. $y \in \vec{y}$) such that $\eta_{i,j}(x) = c$ (resp. $\eta_{i,j}(y) = c$) with $c \in const(\Delta)$, $h_{i,j}(x) = \eta_{i,j}(x)$ (resp. $h_{i,j}(y) = \eta_{i,j}(y)$); *(ii)* for each $x \in \vec{x}$ (resp. $y \in \vec{y}$) such that $\eta_{i,j}(x) = \ell$ (resp. $\eta_{i,j}(y) = \ell$) with $\ell \in ln(\Delta)$, $h_{i,j}(x) = r(\eta_{i,j}(x))$ (resp. $h_{i,j}(y) = r(\eta_{i,j}(y))$). We have that $h_{i,j}$ is actually an homomorphism from the set of variables $\vec{x}$, $\vec{y}$ in ϕ_i to the instance J. Since the set of new facts coming from effect ξ_i can be identified by the couple $\langle \xi_i, \eta_{i,j} \rangle$, let us informally write $I'_{new} = \{\langle \xi_1, \eta_{1,1} \rangle, \ldots, \langle \xi_n, \eta_{n,m} \rangle\}$.

Now we show that $J \xrightarrow{\xi_1,h_{1,1}}, \ldots, \xrightarrow{\xi_n,h_{n,m}} J'$ is the chase step sequence we need in order to get the claim (with each $h_{i,j}$ obtained from $\eta_{i,j}$ and r as before). Let us label such tuples (and resulting instances) with consecutive numbers. We get $J \xrightarrow{0} J_1 \xrightarrow{1}, \ldots, \xrightarrow{p} J'$. We prove by induction that, in any instance J_i with $0 \leq i < p$ it is possible to perform the $i+1$-th chase step, and then that $I' \overset{mnr}{=} J'$.

Base case: Since $\langle \xi_1, \eta_{1,1} \rangle$ is in I_{new} this means that $h_{1,1}$ is an homomorphism from variables in ψ_1 to J and that it cannot be extended to an homomorphism $\phi_1 \wedge \psi_1$ to J (otherwise $\langle \xi_1, \eta_{1,1} \rangle$ would not result in a new fact in I'). So the first chase step can be executed.

Inductive case: By inductive hypothesis every (and only) chase steps labeled with numbers less than i have been executed ($I_0 \xrightarrow{0} J_1 \xrightarrow{1} J_2 \xrightarrow{2} \ldots \xrightarrow{i} J_{i+1}$). Now we prove that it is possible to perform $J_{i+1} \xrightarrow{i+1} J_{i+2}$. If the couple labeled with $i+1$, say, $\langle \xi_{i+1}, \eta_{i+1,j} \rangle$, is in I'_{new} this means that $h_{i+1,j}$ is an homomorphism from variables in ψ_{i+1} to J and that it cannot be extended to an homomorphism $\phi_{i+1} \wedge \psi_{i+1}$ to J. Since the chase is inflationary, we have that $h_{i+1,j}$ is also an homomorphism from ψ_{i+1} to J_{i+1}, and moreover, by inductive hypothesis, the $i+1$-th couple has not been used in a previous chase step, so $h_{i+1,j}$ cannot be extended from $\phi_{i+1} \wedge \psi_{i+1}$ to J_{i+1}, so the chase step $J_{i+1} \xrightarrow{\xi_{i+1}, \eta_{i+1,j}} J_{i+2}$ can be performed. Since $J' = J \cup J'_{new}$ and, by construction, $I'_{new} \overset{mnr}{=} J'_{new}$, we get that $I' \overset{mnr}{=} J'$.

□

Lemma 5.4. *Let $A = \langle \mathcal{D}, \mathcal{T}, \mathcal{C} \rangle$ be a weakly acyclic artifact with initial data instance I_0, and $A^+ = \{\mathcal{D}, \mathcal{T}, \mathcal{C}\}$ its inflationary approximate. Then the fixpoint $I^{max}_{I_0}$ has finite cardinality.*

Proof. Let $\mathfrak{S}_{A^+} = \langle \Sigma_s, \sigma_{0,s}, L_s, Tr_s \rangle$ the execution transition system of A^+. Roughly speaking, $\mathfrak{S}_{A^+}$ is the transition system obtained by applying the g function in every possible way, and generating instances that are homomorphically non-equivalent only. We show that for every sequence $\sigma_{0,s} \overset{T_i}{\Longrightarrow} \ldots \overset{T_j}{\Longrightarrow} \sigma_{n,s}$, there exists a sequence of chase steps $I_0 \xrightarrow{\xi_i, h_i} \ldots, \xrightarrow{\xi_j, h_j} I_m$, where $I_0 = L_s(\sigma_{0,s})$ and $\xi_i \ldots \xi_j$ are tgds in $\mathcal{T}^+$, such that $L_s(\sigma_{i,s}) \overset{mnr}{=} I_j$. By induction on the number of enactment of tasks. *Base case*: trivial, since the equality modulo nulls renaming is reflexive. *Inductive case*: by inductive hypothesis, there exists a $k \leq n$ and a $p \leq m$ such that $L_s(\sigma_{k,s}) \overset{mnr}{=} I_p$. We show that if $\sigma_{k,s} \overset{T}{\Longrightarrow} \sigma_{k+1,s}$ then there exists a sequence of chase step $I_p, \xrightarrow{\xi_i, h_i} \ldots \xrightarrow{\xi_j, h_j} I_{p+\ell}$ such that $L_s(\sigma_{k+1,s}) \overset{mnr}{=} I_{p+\ell}$. This is guaranteed from Lemma 5.3.

We proved that every instance generated by the execution transition system is equal modulo nulls renaming to an instance generated by a sequence of chase step. By results in [30] we have that if effect specifications in $\mathcal{T}^+$ are weakly acyclic, then there exists a polynomial in the size of the initial instance I_0 that bounds the length, and so the size, of every chase sequence of I_0 with $\mathcal{T}^+$. Since two instances that are equals modulo nulls renaming have also the same size, results in [30] also apply to our (inflationary) execution transition system, so $I^{max}_{I_0}$ has finite cardinality.

□

Theorem 5.2. Let $A = \langle \mathcal{D}, \mathcal{T}, \mathcal{C} \rangle$ be a weakly acyclic artifact with initial data instance I_0. Then, for every formula Φ of $\mu\mathcal{L}$, verifying that Φ holds in A with initial data instance I_0 is decidable.

Proof. By Theorem 5.1 and Theorem 4.1, we can perform model checking of Φ on the execution transition system for A. Now, by Lemma 5.2, we have that all data instances that can be assigned to the states of the execution transition system for A must be subsets of $I_{I_0}^{max}$. By Lemma 5.4, we get that $I_{I_0}^{max}$ has a finite cardinality. This implies that execution transition system is finite and Theorem 4.2 can be applied.

□

As mentioned above, all these results can be readily extended to generalization of the weak acyclicity condition as those proposed in [28,1,44,45].

Finally, we briefly comment on the significance of weak acyclic conditions (the original one as well as its extensions mentioned above). We argue that the restriction is not too severe, and that in most real cases artifacts are indeed weakly acyclic or can be transformed into weakly acyclic ones at cost of redesign. Our argumentation is grounded on the following observation: if an artifact is not weakly acyclic, then it will repeatedly generate new values from the old ones. Such values will depend on a chain of previous values of *unbounded* length. But this means that current values depend on old values that are arbitrarily far in the past, and moreover on an unbounded number of such old values. Notice that, if such a number can be bounded, then, in principle, the artifact can be rewritten into a weakly acyclic one. While such unbounded system exists in theory, e.g., Turing machines, where the artifact data component is the tape, most services, which are naturally more abstract than Turing machines, will not require such an unboundedness in practice. On the other hand, while we believe that most services can be rewritten into weakly acyclic ones, how to systematically to this transformation is an issue that requires further studies.

6. Related Work

A common pattern in computer science is the constantly increasing complexity of systems, therefore a main challenge is to provide formalisms, techniques, and tools that will enable the efficient design of correct and well-functioning systems despite their complexity. Hence verification of programs, processes, protocols and hardware has been recognized as one of the most important branches of computer science, and therefore largely analyzed [11,49,38,31,23]. With the growth, in the last fifteen years or so, of formal grounds for business workflows and service management, verification of services business processes, and Petri nets based process models, attracted the attention of the scientific community, because in the aforementioned systems dynamic properties, such as no deadlocks or existence of a proper completion, are of extreme interest from a practical point of view, and very challenging from a theoretical one [59,19,56].

Model checking has been a major breakthrough in verification [7,23]. Most model checking techniques require the dynamic system to verify to be finite-state, since they verify properties by systematically exhaustive exploration of the mathematical model that describes the system. Typically symbolic techniques are used to reduce the cost of the state space exploration [16]. Often, even if finite, the systems' state spaces are in practice too large, and require the use of smart abstraction techniques, such as symmetry abstraction, abstract

interpretation, symbolic simulation and abstraction refinement, to make such analysis very effective in practice [22]. Notice that most abstraction techniques give rise to so-called false-negative, i.e., the verification fails on the abstracted system but the property is true in the real one. We can see the technique presented here as a *faithful* form of abstraction (i.e., with no false negatives), where we suitably abstract several null values into one through the notion of homomorphic equivalence, though mantaining soundness and completeness of verification.

Some verification techniques, including model checking, deal also with infinite-state systems. In fact, infinite-state systems often occur in the practice, and their verification is certainly of great practical interest. For instance, any recursive program is potentially infinite-state, due to the possibly unbounded grow of the stack [17,14]. A number of solutions have been proposed to deal with state infiniteness. Many of them are based on identifying interesting classes of transition systems, definable by suitable formalisms, and some respective classes of decidable properties. For example, decidability results for general infinite-state mathematical structures are shown in [2,32]. Such structures are called *well-structured transition systems* and consist in a finite control part operating on an infinite data domain satisfying well-founded preorder. Decidability is guaranteed for a restricted class of properties such as *control state reachability*, *eventuality* formulas, and *simulation* between a finite automaton and a well-structured transition system. Interestingly, special kinds of Petri nets, rewrite systems, communicating state machines, pushdown automata and other systems fall in the general category of well-structured transition systems, hence the above mentioned properties are decidable. Other results obtain decidability by suitable manipulation of basic transition systems, such as transductions [25], tree-iteration [58], and unfoldings, that can be intuitively thought of as operations that build transition systems out of transition systems, by preserving some regularities that can be exploited by the verification algorithms. Another approach is regular model checking [13,41,12], a uniform paradigm for algorithmic verification of several classes of parameterized and infinite-state systems. In this account system states are captured by strings of arbitrary length over a finite alphabet, and the transition relation is given by a regular, length-preserving relation on strings, usually represented by a finite-state transducer. The fundamental problem of computing the set of reachable states from a given initial configuration, or reachability analysis, is tackled by using two complementary techniques: an automata-theoretic construction, and a fixpoint computation.

Artifact-based systems are in general infinite-state, due to the presence of unbounded data. The distinctive feature of artifacts is the presence, in each state, of data with an explicit possibly rich structure, such as relational databases considered here. This induces the necessity of a corresponding rich query language such as first-order logic, SQL or conjunctive queries, for querying the state. Mixing such a rich query capability on the current state with the evolution given by the lifecycle, makes artifacts infinite-state systems of a different nature with respect to the ones mentioned above. The proposals in this setting have been sparse, since they require knowledge of both dynamic systems and databases, but the issue is increasingly attracting interest lately. In [6] decidability results for verifying temporal properties over artifact systems are shown, and they are obtained by abstraction

and by bounding the size of the so-called *deployed* instances. The work reported in [27,26] share the general setting with our approach but differs in the conditions required to obtain decidability. Such conditions are not based on conjunctive queries, but on some decidability results of certain formulas of a first-order variant of linear time temporal logic [53,52]. Another relevant work is that on SPOCUS relational transducers [5], where decidability is obtained through results on inflationary Datalog. The work on service composition according to the COLOMBO model [8] is also related to the present approach. There, decidability is obtained through symbolic abstraction on data and the requirement that processes are input bounded (i.e., take only a bounded number of new values, similar to our Skolem functions, taken from input). Work on formal analysis of forms of artifact-centric processes has also been reported in [9,18,34,35]. In [9] the authors check whether an artifact modified by services *successful completes* or if there are *dead-end paths*, and decidability is obtained posing restrictions on services, such as trivial, i.e., true, or not negative preconditions. In [18] the problem of checking whether one artifact-centric workflow may emulate the possible behaviors of another one is shown, and decidability is guaranteed if the infinite domain of artifacts' attributes is ordered. Verification of more general properties, expressed in a CTL-like language are analyzed in [34,35]. Unfortunately, decidability for the full language is obtained by bounding the domain, and for unbounded (but yet ordered) domain only a fragment of the original language is decidable. Finally, among the various Petri net based business process models, *colored Petri nets* (see [40]) is the one that takes into account data: every token has a value from a (possibly infinite) domain. Such a powerful formalism is used for verification purposes, e.g., protocol verification, but again, data are abstracted or trivially bounded, making all markings, i.e., net configurations, finite. The results presented here are not subsumed by (nor subsume) any of the above results.

7. Conclusions

In this paper we have introduced conjunctive artifact-based services, a class of services which pose balanced attention to both data (here a full-fledged relational database) and processes (acting on the database) and guarantees decidability through a suitable use of conjunctive queries in specifying task pre-conditions and post-conditions. The approach presented here actually opens a new lode for research in the area, based on the connection with the theory of dependencies in databases that has been so fruitful in data exchange and data integration in recent years [30,42].

We are currently looking at extending this approach in several directions. First, we are interested in including negation in the preconditions of tasks effects and in the condition-action rules that form the lifecycle, as well as task parameters that are more than syntactic sugar, as in the case presented here. First results on this are presented in [37]. Interestingly, introducing negation rises the issue of a suitable treatment of assertions determining equalities, such as key assertions on relations forming the data component. This brings about several subtleties that we intend to explore. Looking at several interacting artifacts together is also of interest. The result presented here can be extended easily to such a case if the artifacts in the system are known initially and remain the same along the whole execution

of the entire system. If, instead, new artifacts can be created and old ones can be destroyed along the execution of the system, then being able to bound the total number of artifact simultaneously active becomes a crucial issue that require further studies. Finally, we are also interested in exploring the case in which the data model of the artifact is not simply a database but an ontology with both explicit and implicit information extracted by logical inference. This would give rise to a sort of semantic artifacts, which, akin to semantic services, abstract from the details of how the information is stored and manipulated. The first steps in this direction are shown in [36].

Acknowledgments We would like to thank our friend Piero Cangialosi that contributed to earlier versions of the paper, and Diego Calvanese and Yves Lesperance for insightful discussions. This work has been supported by the EU Project FP7-ICT ACSI (257593).

Bibliography

1. M. M. 0002, M. Schmidt, and G. Lausen. On chase termination beyond stratification. *PVLDB*, 2(1):970–981, 2009.
2. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313–321, 1996.
3. S. Abiteboul, P. Bourhis, A. Galland, and B. Marinoiu. The axml artifact model. In *TIME*, pages 11–17, 2009.
4. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
5. S. Abiteboul, V. Vianu, B. S. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *J. Comput. Syst. Sci.*, 61(2):236–269, 2000.
6. F. Belardinelli, A. Lomuscio, and F. Patrizi. Verification of deployed artifact systems via data abstraction. In *ICSOC (to appear)*, 2011.
7. B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer Publishing Company, Incorporated, 1st edition, 2010.
8. D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *VLDB*, pages 613–624, 2005.
9. K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *BPM*, pages 288–304, 2007.
10. K. Bhattacharya, R. Guttman, K. Lyman, F. F. H. III, S. Kumaran, P. Nandi, F. Y. Wu, P. Athma, C. Freiberg, L. Johannsen, and A. Staudt. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Systems Journal*, 44(1):145–162, 2005.
11. B. W. Boehm and V. R. Basili. Software defect reduction top 10 list. *IEEE Computer*, 34(1):135–137, 2001.
12. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *CAV*, pages 372–386, 2004.
13. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV*, pages 403–418, 2000.
14. P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2), 2003.
15. J. Bradfield and C. Stirling. Modal mu-calculi. In *Handbook of Modal Logic*, volume 3, pages 721–756. Elsevier, 2007.
16. R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.
17. O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification of infinite structures. In *Handbook*

of Process Algebra. Elsevier Science, 2001.
18. D. Calvanese, G. D. Giacomo, R. Hull, and J. Su. Artifact-centric workflow dominance. In *ICSOC/ServiceWave*, pages 130–143, 2009.
19. M. Castellanos, F. Casati, M. Sayal, and U. Dayal. Challenges in business process analysis and optimization. In *TES*, pages 1–10, 2005.
20. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.
21. K. L. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322, 1977.
22. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.
23. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. The MIT Press, Cambridge, MA, USA, 1999.
24. D. Cohn and R. Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.
25. B. Courcelle. Monadic second-order definable graph transductions: A survey. *Theor. Comput. Sci.*, 126(1):53–75, 1994.
26. E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. In *ICDT*, pages 66–77, 2011.
27. A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.
28. A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.
29. E. A. Emerson. Model checking and the mu-calculus. In *Descriptive Complexity and Finite Models*, pages 185–214, 1996.
30. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
31. M. Felleisen, R. B. Findler, and M. Flatt. *Semantics Engineering with PLT Redex*. MIT Press, 2009.
32. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256:63–92, April 2001.
33. C. Fritz, R. Hull, and J. Su. Automatic construction of simple artifact-based business processes. In *ICDT*, pages 225–238, 2009.
34. C. E. Gerede, K. Bhattacharya, and J. Su. Static analysis of business artifact-centric operational models. In *SOCA*, pages 133–140, 2007.
35. C. E. Gerede and J. Su. Specification and verification of artifact behaviors in business process models. In *ICSOC*, pages 181–192, 2007.
36. B. B. Hariri, D. Calvanese, G. D. Giacomo, and R. D. Masellis. Verification of conjunctive-query based semantic artifacts. In *Description Logics*, 2011.
37. B. B. Hariri, D. Calvanese, G. D. Giacomo, R. D. Masellis, and P. Felli. Foundations of relational artifacts verification. In *BPM*, pages 379–395, 2011.
38. D. V. Horn and M. Might. Abstracting abstract machines. In *ICFP*, pages 51–62, 2010.
39. R. Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *OTM Conferences (2)*, pages 1152–1163, 2008.
40. K. Jensen and L. M. Kristensen. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer, 2009.
41. B. Jonsson and M. Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *TACAS*, pages 220–234, 2000.
42. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.
43. D. C. Luckham, D. M. R. Park, and M. Paterson. On formalised computer programs. *J. Comput. Syst. Sci.*, 4(3):220–249, 1970.
44. B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages

13–22, 2009.
45. B. Marnette and F. Geerts. Static analysis of schema-mappings ensuring oblivious termination. In *ICDT*, pages 183–195, 2010.
46. R. Milner. An algebraic definition of simulation between programs. In *IJCAI*, pages 481–489, 1971.
47. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.
48. D. M. R. Park. Finiteness is mu-ineffable. *Theor. Comput. Sci.*, 3(2):173–181, 1976.
49. D. A. Peled, D. Gries, and F. B. Schneider, editors. *Software reliability methods*. Springer-Verlag New York, Inc., 2001.
50. R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.
51. S. Sohrabi, N. Prokoshyna, and S. A. McIlraith. Web service composition via generic procedures and customizing user preferences. In *International Semantic Web Conference*, pages 597–611, 2006.
52. M. Spielmann. Automatic verification of abstract state machines. In *CAV*, pages 431–442, 1999.
53. M. Spielmann. Verification of relational transducers for electronic commerce. *J. Comput. Syst. Sci.*, 66(1):40–65, 2003.
54. C. Stirling. *Modal and temporal properties of processes*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.
55. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.
56. W. M. P. van der Aalst. Challenges in business process management: Verification of business processing using petri nets. *Bulletin of the EATCS*, 80:174–199, 2003.
57. W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis, and J. Wainer. Proclets: A framework for lightweight interacting workflow processes. *Int. J. Cooperative Inf. Syst.*, 10(4):443–481, 2001.
58. I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theor. Comput. Sci.*, 275(1-2):311–346, 2002.
59. M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.

De Giacomo, G., De Masellis, R., and Rosati, R. Verification of conjunctive artifact-centric services. *Int. J. of Cooperative Information Systems*, **21** (2012), 111. doi:10.1142/S0218843012500025.

@article{DeDR12,
Author = {De Giacomo, Giuseppe and De Masellis, Riccardo and Riccardo Rosati},
Journal = Int.\ J.\ of Cooperative Information Systems,
Number = 2,
Pages = {111--139},
Title = {Verification of Conjunctive Artifact-Centric Services},
Volume = 21,
Year = 2012,
ee = {http://dx.doi.org/10.1142/S0218843012500025}}