Process-Centric Views of Data-Driven Business Artifacts*

Adrien Koutsos[1] and Victor Vianu[2]

1 **ENS Cachan, France**
adrien.koutsos@ens-cachan.fr
2 **UC San Diego & INRIA-Saclay**
vianu@cs.ucsd.edu

Abstract

Declarative, data-aware workflow models are becoming increasingly pervasive. While these have numerous benefits, classical process-centric specifications retain certain advantages. Workflow designers are used to development tools such as BPMN or UML diagrams, that focus on control flow. Views describing valid sequences of tasks are also useful to provide stake-holders with high-level descriptions of the workflow, stripped of the accompanying data. In this paper we study the problem of recovering process-centric views from declarative, data-aware workflow specifications in a variant of IBM's business artifact model. We focus on the simplest and most natural process-centric views, specified by finite-state transition systems, and describing regular languages. The results characterize when process-centric views of artifact systems are regular, using both linear and branching-time semantics. We also study the impact of data dependencies on regularity of the views.

1998 ACM Subject Classification H.2.3 Languages: Query languages, H.4.1 Office Automation: Workflow management, B.4.4 Performance Analysis and Design Aids: Formal models, Verification

Keywords and phrases Workflows, data-aware, process-centric, views

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.247

1 Introduction

Data-driven workflows have become ubiquitous in a variety of application domains, including business, government, science, health-care, social networks [37], crowdsourcing [7, 8], etc. Such workflows resulted from an evolution away from the traditional process-centric approach towards data-awareness. A notable exponent of this class is the *business artifact model* pioneered in [31, 25], deployed by IBM in professional services. Business artifacts (or simply "artifacts") model key business-relevant entities, which are updated by a set of services that implement business process tasks. This modeling approach has been successfully deployed in practice [4, 3, 9, 14, 39]. In particular, the Guard-Stage-Milestone (GSM) approach [12, 23] to artifact specification uses declarative services with pre- and post-conditions, parallelism, and hierarchy. The OMG standard for Case Management Model and Notation (CMMN) [5], announced last year, draws key foundational elements from GSM [28].

Declarative, high-level specification tools such as GSM allow to generate the application code from the high-level specification. This not only allows fast prototyping and improves programmer productivity but, as a side effect, provides new opportunities for automatic

* This work was partially supported by the National Science Foundation under award IIS-1422375.

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martín Ugarte; pp. 247–264

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

verification. Indeed, the high-level specification is a natural target for verification, as it addresses the most likely source of errors (the application's specification, as opposed to the less likely errors in the automatic generator's implementation). This has spawned an entire line of research seeking to trace the boundary of decidability of properties of such systems, expressed in variants of temporal logic (see [18]).

While declarative specifications of workflows have many benefits, they also come with some drawbacks. Workflow designers are used to traditional process-centric development tools, such as BPMN (Business Process Model and Notation), workflow nets, and UML activity diagrams, that focus on control flow while under-specifying or ignoring the underlying data. Such process-centric views of workflows are also useful to provide stakeholders with customized descriptions of the workflow, tailored to their role in the organization. For example, an executive may only require a high-level summary of the business process. Descriptions of the workflows as sequences of tasks stripped of data are intuitive and often sufficient for many users. Thus, recovering the sequencing of tasks from declarative specification is often desirable.

In this paper, we study views of business artifact runs consisting of the sequences of services applied in each run[1]. This comes in two flavors, depending on whether we are interested in linear runs alone, or in the more informative branching-time runs. We call these the linear-time, resp. branching-time (service) views of the artifact system. The simplest and most intuitive specification mechanism for such views consists of a finite-state transition system, which can describe both ω-regular languages (for linear-time views), and regular infinite trees of runs (for branching-time views).

▶ **Example 1.** To illustrate linear-time service views of declarative workflows, we consider a variant of the running example of [11]. In the example, the customer can choose a product, a shipment method and apply a coupon to the order. The order is filled in a sequential manner as is customary on e-commerce web-sites. After the order is filled, the system awaits for the customer to submit a payment. If the payment matches the amount owed, the system proceeds to shipping the product. At any time before submitting a valid payment, the customer may edit the order (select a different product) an unbounded number of times. The linear-time service view of this workflow consists of the sequences of service names that occur in all infinite runs of the system, and is specified by the finite-state transition system shown in Figure 1. A more informative view, that captures the services applied in branching-time runs of the system, is shown in Figure 2. Intuitively, the branching-time view captures *precisely* which services may be applied *at each point* in a run. To understand the difference with linear-time views, consider the states labeled *edit coupon* in Figures 1 and 2 (the latter highlighted in red). In the linear-time view specification, there is only one such state, in which two actions can be taken: *no coupon* and *apply coupon*. However, the two actions are not *always* applicable whenever the state *edit coupon* is reached. If no product has an applicable coupon, the only action possible is *no coupon*. If for all products and shipping method there is some applicable coupon, then both *no coupon* and *apply coupon* are applicable. Finally, if some products have applicable coupons and others do not, then both of the above cases may occur depending on the choice of product. The different cases are captured by distinct states in the branching-time specification, while the information is lost in the linear-time specification. Of course, the *sequences* of service names occurring in all runs of the system are the same in both specifications.

[1] In various formalizations of business artifacts, tasks are called *services*. As usual in program verification, we take runs to be infinite, because many business processes run forever and because infinite runs capture information lost by finite prefixes.

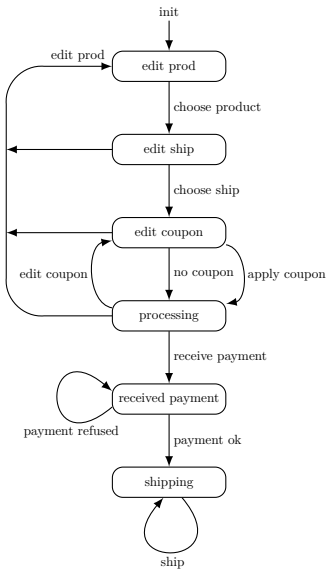

Figure 1 A linear-time process-centric view.

The main results of the paper establish under what circumstances the linear-time or branching-time service views are regular (with effective specifications). We consider the tuple artifact model used in [17, 11], and several natural restrictions derived from the GSM design methodology, some of which have been considered in the context of verification [11]. We also consider the impact of database dependencies on regularity of the views. We show that linear-time service views of tuple artifacts are ω-regular, but branching-time views are not regular. We then consider artifacts obeying a natural restriction previously used in the context of verification, called *feedback freedom* [11]. We show that branching-time views of feedback-free artifacts are still not regular, but become so under a slightly stronger restriction called *global feedback freedom*. This restriction is naturally obeyed by specifications resulting from the GSM hierarchical design methodology.

It turns out that there is a tight connection between the result on view regularity and the verification of artifact systems. Properties to be verified are specified using extensions of the classical temporal logics LTL, CTL and CTL*, in which propositions are interpreted as existential FO formulas on the tuple artifact and the underlying database. This yields the logics LTL-FO, CTL-FO and CTL*-FO [17, 11, 20]. It can be shown that regularity of the linear or branching-time service views for a class of artifact systems, with effectively constructible specifications, implies decidability of LTL-FO, resp. CTL*-FO properties for that class. The converse is false: decidability of LTL-FO or CTL*-FO properties may hold even though the corresponding service views may not be regular. Thus, regularity is a stronger result than decidability of verification. Indeed, our results imply that CTL*-FO properties are decidable for globally feedback-free artifact systems. On the other hand, we show that CTL-FO properties are undecidable for feedback-free artifact systems (also implying non-regularity of their branching-time views).

The proof techniques developed here have additional side-effects beneficial to verification. In our previous approaches to automatic verification of business artifacts [17, 11], given an artifact specification $\mathcal{A}$ and an LTL-FO property φ, the verifier either certifies satisfaction of the property or produces a counter-example run on some database $D(\mathcal{A}, \varphi)$ depending on both $\mathcal{A}$ and φ. In contrast, the techniques of the present paper allow to show that,

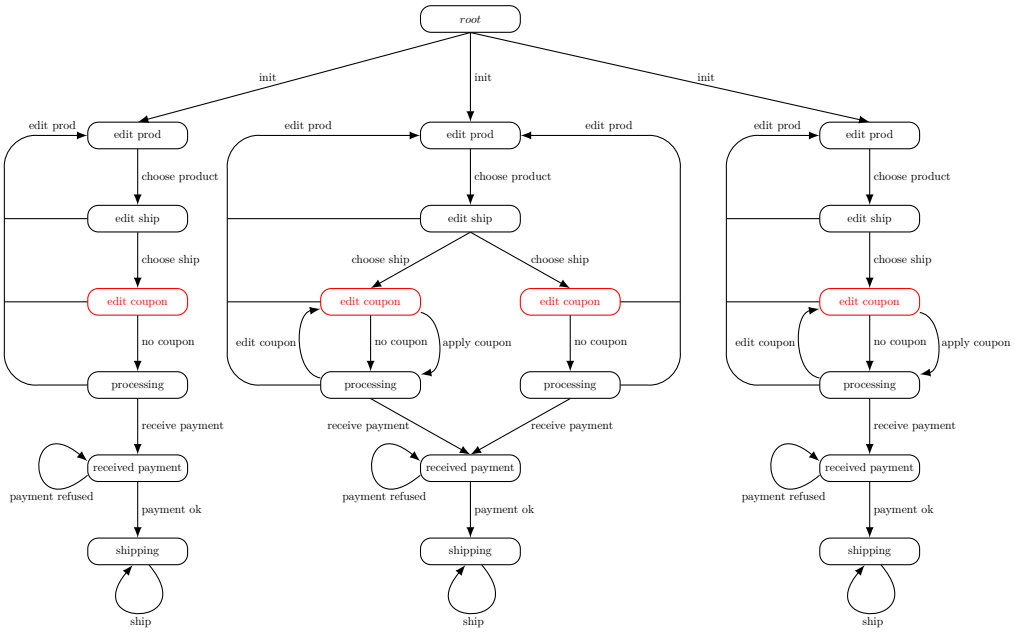

Figure 2 A branching-time process-centric view.

for specifications and properties without constants, there exists a *single* database $D(\mathcal{A})$, depending only on $\mathcal{A}$, which serves as a universal counter-example for *all* LTL-FO properties violated by $\mathcal{A}$. Pre-computing such a database may allow more efficient generation of counter-examples for specific LTL-FO properties.

Decidability results on verification of branching-time properties of infinite-state artifact systems are scarce and require significant restrictions. In [22, 13], decidability results are shown for properties expressed in an FO extension of μ-calculus, under restrictions on the artifact system orthogonal to ours. In [2], the artifact model is extended to a multi-agent setting, and decidability results are shown for properties expressed in an FO extension of CTL that can also refer to each agent's knowledge using a Kripke-style semantics. Decidability of similar FO extensions of CTL is shown in [27] for the case when the database consists of a single unary relation.

2 Background

After some basic definitions, we present the tuple artifact model, the languages LTL-FO and CTL$^{(*)}$-FO, and review previous results on verification of LTL-FO properties.

We assume an infinite data domain **dom**. A *relational schema* is a finite set of relation symbols with associated arities. An *database instance* over a relational schema $\mathcal{DB}$ is a mapping I associating to each $R \in \mathcal{DB}$ a *finite* relation over **dom** with the same arity as R (denoted $arity(R)$). We assume familiarity with first-order logic (FO) over relational schemas (e.g., see [1, 26]). FO formulas may use equality and constants from **dom**.

Artifact systems. In the tuple artifact model, an artifact consists of a finite set of variables whose values evolve during the workflow execution. Transitions are specified declaratively, using pre-and-post conditions that may query an underlying database.

▶ **Definition 2.** An *artifact schema* is a a tuple $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ where $\bar{x}$ is a finite sequence $x_1, \ldots, x_k$ of *artifact variables* and $\mathcal{DB}$ is a relational schema.

For each $\bar{x}$, we also define a set of variables $\bar{x}' = \{x' \mid x \in \bar{x}\}$ where each x' is a distinct new variable. In a transition, a primed variable represents the value of the variable in the new artifact.

▶ **Definition 3.** An *instance* of an artifact schema $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ is a tuple $A = \langle \nu, D \rangle$ where ν is a valuation of $\bar{x}$ into **dom** and D is a finite instance of $\mathcal{DB}$.

▶ **Definition 4.** A *service* over an artifact schema $\mathcal{A}$ is a pair $\sigma = \langle \pi, \psi \rangle$ where:
- $\pi(\bar{x})$, called *pre-condition*, is a quantifier-free[2] FO formula over $\mathcal{DB}$ with variables $\bar{x}$;
- $\psi(\bar{x}, \bar{x}')$, called *post-condition*, is a quantifier-free FO formula over $\mathcal{DB}$ with variables $\bar{x}, \bar{x}'$.

▶ **Definition 5.** An *artifact system* is a triple $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$, where $\mathcal{A}$ is an artifact schema, Σ is a non-empty set of services over $\mathcal{A}$, and Π is a pre-condition restricting the value of the initial artifact variables (as above, a quantifier-free FO formula over $\mathcal{DB}$, with variables $\bar{x}$).

▶ **Definition 6.** Let $\sigma = \langle \pi, \psi \rangle$ be a service over an artifact schema $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$, and let D be an instance over $\mathcal{DB}$. Let ν, ν' be valuations of $\bar{x}$. We say that ν' is a *possible successor* of ν w.r.t. σ and D (denoted $\nu \xrightarrow{\sigma} \nu'$ when D is understood) iff:
- $D \models \pi(\nu)$, and
- $D \models \psi(\nu, \nu')$.

Note that $\psi(\bar{x}, \bar{x}')$ need not bind $\bar{x}'$ to the database, so ν' may contain values not in D.

▶ **Definition 7.** Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$. A *run* of Γ on database instance D over $\mathcal{DB}$ is an infinite sequence $\rho = \{(\rho_i, \sigma_i)\}_{i \geq 0}$, where for each $i \geq 0$, ρ_i is a valuation of $\bar{x}$, $\sigma_i \in \Sigma$, $\rho_i \xrightarrow{\sigma_i} \rho_{i+1}$, and $D \models \Pi(\rho_0)$.

The assumption that runs are infinite is not a restriction, since finite runs can be artificially extended to infinite runs by adding a self-looping transition. The linear-time semantics of an artifact system Γ is provided by the set of all runs. Specifically, we denote by $\mathit{Runs}_D(\Gamma)$ the set of all runs of Γ on database instance D, and by $\mathit{Runs}(\Gamma)$ the union of $\mathit{Runs}_D(\Gamma)$ for all databases D. The more informative branching-time semantics is provided by its *tree of runs*, additionally capturing all choices of transitions available at each point in the run. More precisely, $\mathit{TRuns}_D(\Gamma)$ is a labeled tree whose nodes are all finite prefixes of runs of Γ on D, such that the children of a prefix are all prefixes extending it by one transition. Each node ρ is labeled by the value of the last artifact tuple in ρ, and each edge from ρ to $\rho.(\nu, \sigma)$ is labeled by σ. The global tree of runs $\mathit{TRuns}^*(\Gamma)$ is the tree obtained by placing all trees $\mathit{TRuns}_D(\Gamma)$ (for every database D) under a common root, connected by edges with a special label *init*. The tree of runs allows to define properties in branching-time temporal logic, such as "any client has the option of canceling a purchase at any time". Note that such a property is not captured by the linear runs in $\mathit{Runs}(\Gamma)$.

Temporal properties of artifact systems. Temporal properties are specified using extensions of LTL (linear-time temporal logic) and CTL$^{(*)}$ (branching-time temporal logics). We begin with LTL. Recall that LTL is propositional logic augmented with temporal operators

[2] ∃FO conditions can be easily simulated by additional artifact variables.

G (always), **F** (eventually), **X** (next) and **U** (until) (e.g., see [32]). Informally, **G**p says that p holds at all times in the run, **F**p says that p will eventually hold, **X**p says that p holds at the next configuration, and p**U**q says that q will hold at some point in the future and p holds up to that point. For example, **G**$(p \rightarrow \mathbf{F}q)$ says that whenever p holds, q must hold sometime in the future.

The extension of LTL used in [11], called[3] LTL-FO, is obtained from LTL by replacing propositions with quantifier-free FO statements about particular artifact records in the run. The statements use the artifact variables and the database. In addition, they may use *global* variables, shared by different statements and allowing to refer to values in different records. The global variables are universally quantified over the entire property. We illustrate LTL-FO with a simple example.

▶ **Example 8.** The following specifies a desirable business rule for an e-commerce workflow with artifact variables including `amount_paid`, `amount_refunded`, `status`, `amount_owed` (with obvious meaning).

$$(\varphi)\ \forall x \mathbf{G}((\texttt{amount\_paid} = x \wedge \texttt{amount\_paid} = \texttt{amount\_owed}) \rightarrow \mathbf{F}(\texttt{status} = \text{"shipped"} \vee \texttt{amount\_refunded} = x))$$

Property φ states that if a correct payment is submitted then at some time in the future either the product is shipped or the customer is refunded the correct amount. Note the use of universally-quantified variable x to relate the value of paid and refunded amounts across distinct steps in the run sequence.

The branching-time extensions CTL-FO and CTL*-FO are defined analogously from CTL and CTL*. Recall that CTL* augments LTL with path quantifiers **A** (for all) and **E** (exists) while CTL restricts the use of path quantifiers so that they are always followed by a temporal operator (see [21]).

We note that variants of LTL-FO have been introduced in [21, 35]. The use of globally quantified variables is also similar in spirit to the *freeze quantifier* defined in the context of LTL extensions with data by Demri and Lazić [15, 16].

Verification of artifact systems. We informally review the results of [17, 11] on verification of LTL-FO properties of artifact systems. Classical model-checking applies to finite-state transition systems. Checking that an LTL property holds is done by searching for a counterexample run of the system. The finiteness of the transition system is essential and allows to decide property satisfaction in PSPACE using an automata-theoretic approach (see e.g. [10, 29]). In contrast, artifacts are infinite-state systems because of the presence of unbounded data. The main idea for dealing with the infinite search space is to explore the space of runs of the artifact system using *symbolic* runs rather than actual runs. This yields the following result.

▶ **Theorem 9.** *[17] It is* PSPACE*-complete to check, given an artifact system $\mathcal{A}$ and an LTL-FO property φ, whether $\mathcal{A}$ satisfies φ.*

Unfortunately, Theorem 9 fails even in the presence of simple data dependencies or arithmetic. Specifically, as shown in [17, 11], verification becomes undecidable as soon as

[3] The variant of LTL-FO used here differs from some previous ones in that the FO formulas interpreting propositions are quantifier-free. By slight abuse we use here the same name.

the database is equipped with at least one key dependency, *or* if the specification of the artifact system uses simple arithmetic constraints allowing to increment and decrement by one the value of some atributes. Hence, a restriction is needed to achieve decidability for these extensions. We discuss this next.

To gain some intuition, consider the undecidability of verification for artifact systems with increments and decrements. The proof of undecidability is based on the ability of such systems to simulate *counter machines*, for which the problem of state reachability is known to be undecidable [30]. To simulate counter machines, an artifact system uses an attribute for each counter. A service performs an increment (or decrement) operations by "feeding back" the incremented (or decremented) value into the next occurrence of the corresponding attribute. To simulate counters, this must be done an unbounded number of times. To prevent such computations, a restriction is imposed in [11], called *feedback freedom*, designed to limit the data flow between occurrences of the same artifact variable at different times in runs that satisfy the desired property. The formal definition considers, for each run, a graph capturing the data flow among variables, and imposes a restriction on the graph. Intuitively, paths among different occurrences of the same variable are permitted, but only as long as each value of the variable is independent on its previous values. This is ensured by a condition that takes into account both the artifact system and the property to be verified, called *feedback freedom*. It turns out that artifact systems designed in a hierarchical fashion by successive refinement, in the style of the Guard-Stage-Milestone approach [12, 23], naturally satisfy the feedback freedom condition [19]. Indeed, there is evidence that the feedback freedom condition is permissive enough to capture a wide class of applications of practical interest. This is confirmed by numerous examples of real-life business processes modeled as artifact systems, encountered in IBM's practice [11].

Feedback freedom turns out to ensure decidability of verification in the presence of arithmetic constraints, and also under a large class of data dependencies including key and foreign key constraints on the database.

▶ **Theorem 10.** *[11] It is decidable, given an artifact system* Γ *using arithmetic (linear inequalities over* $\mathbb{Q}$*) and whose database satisfies a set of key and foreign key constraints, and an LTL-FO property* φ *such that* (Γ, φ) *is feedback free, whether every run of* $\mathcal{A}$ *on a valid database satisfies* φ*.*

Unfortunately, the complexity is non-elementary with respect to the number of artifact variables.

3 Linear-time service views

In this section, we define linear-time service views of artifact systems and establish their regularity. Throughout the paper, all results establishing regularity are effective, in the sense that specifications can be effectively constructed.

We begin by recalling some basics on languages on infinite words. Let Δ be a finite alphabet. An ω-word over Δ is an infinite sequence of symbols from Δ and an ω-language is a set of ω-words. An ω-language is ω-regular if it is accepted by a *Büchi automaton* (e.g., see [36]). A Büchi automaton is a non-deterministic finite-state automaton accepting the infinite words for which some run of the automaton goes through an accepting state infinitely often. A Büchi automaton in which every state is accepting is also referred to as a finite-state transition system (or safety automaton, see [24]). Thus, a finite-state transition system defines the ω-words for which there is some non-blocking run of the system with transitions labeled by the symbols of the word.

Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$. For each run $\rho = \{(\rho_i, \sigma_i)\}_{i \geq 0}$ of Γ, the service view of ρ, denoted $\mathcal{S}(\rho)$, consists of the ω-word $(\sigma_i)_{i \geq 0}$. The linear-time service view of Γ is $\mathcal{S}_{lin}(\Gamma) = \{\mathcal{S}(\rho) \mid \rho \in \mathit{Runs}(\Gamma)\}$. We say that $\mathcal{S}_{lin}(\Gamma)$ is *effectively* ω-regular for a class of artifact systems if a Büchi automaton defining $\mathcal{S}_{lin}(\Gamma)$ can be effectively constructed from each Γ in that class.

We will show that $\mathcal{S}_{lin}(\Gamma)$ is effectively ω-regular for artifact systems. To do so, we will use symbolic representations of the runs of Γ. Let C be the set of constants used in Γ. To each $x \in \bar{x}$ we associate an infinite set of new variables $\{x_i\}_{i \geq 0}$, and we denote $\bar{x}_i = \{x_i \mid x \in \bar{x}\}$. An *equality type* for variables $\bar{y}$ is an equivalence relation on $\bar{y} \cup C$ in which no distinct constants in C are equivalent. An isomorphism type of $\bar{y}$ is a pair (H, ϵ) where ϵ is an equality type for $\bar{y}$ and H is an instance of $\mathcal{DB}$ using elements in $\bar{y} \cup C$ that is consistent with ϵ.

▶ **Definition 11.** A symbolic run ϱ of Γ is a sequence $\{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$ such that, for each $i \geq 0$:

- **(i)** (H_i, ϵ_i) is an isomorphism type of $\bar{x}_i \cup \bar{x}_{i+1}$,
- **(ii)** the pre-condition Π of Γ holds in the restriction of (H_0, ϵ_0) to $\bar{x}_0$,
- **(iii)** (H_i, ϵ_i) and $(H_{i+1}, \epsilon_{i+1})$ agree on $\bar{x}_{i+1}$,
- **(iv)** the pre-condition of σ_i holds in the restriction of (H_i, ϵ_i) to $\bar{x}_i$, and
- **(v)** the post-condition of σ_i holds in (H_i, ϵ_i).

We denote by $\mathit{SRuns}(\Gamma)$ the set of symbolic runs of Γ.

It is easy to see that each run of Γ has a corresponding symbolic run, representing the consecutive isomorphism types of the artifact tuples in the run. We make this more precise. Let $\approx$ be the transitive closure of $\cup_{i \geq 0} \epsilon_i$. Clearly, $\approx$ is an equivalence relation on $\cup_{i \geq 0} \bar{x}_i \cup C$. It is easily seen that ϵ_i is the restriction of $\approx$ to $\bar{x}_i \cup \bar{x}_{i+1} \cup C$. Let $[x_i]_{\epsilon_i}$ and $[x_i]_{\approx}$ denote the equivalence class of x_i in ϵ_i, resp. $\approx$.

▶ **Definition 12.** Let $\{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$ be a symbolic run of Γ. An *enactment* of ϱ is a triple (D, ρ, θ) where D is a database over $\mathcal{DB}$, $\rho = \{(\rho_i, \sigma_i)\}_{i \geq 0}$ is a run of Γ over D, and θ is a mapping from $(\cup_{i \geq 0} \bar{x}_i)$ to **dom** such that, for each $i \geq 0$:

- $\theta(x_i) = \rho_i(x)$ for every $x \in \bar{x}$,
- if $y, z \in \bar{x}_i \cup \bar{x}_{i+1}$ and $(y, z) \in \epsilon_i$ then $\theta(y) = \theta(z)$
- the function $\hat{\theta}$ defined by $\hat{\theta}([y]_{\epsilon_i}) = \rho(y)$ for $y \in \bar{x}_i \cup \bar{x}_{i+1}$ is an isomorphism from H_i/ϵ_i to $D|(\rho_i \cup \rho_{i+1})$.

▶ **Lemma 13.** *For every database D over $\mathcal{DB}$ and run ρ of Γ over D there exists a symbolic run ϱ of Γ and a mapping h from $(\cup_{i \geq 0} \bar{x}_i)$ to **dom** such that (D, ρ, h) is an enactment of ϱ.*

Proof. The run ϱ can be easily constructed by induction from ρ. ◀

Consider the converse of Lemma 13: does every symbolic run have an enactment on some database? This is much less obvious. It is easy to construct, for each symbolic run $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$, a triple (D_ϱ, ρ, h) satisfying the definition of enactment except for the finiteness of D_ϱ. This can be done as follows. The (infinite) domain of D_ϱ simply consists of all equivalence classes of $\approx$, h maps each x_i to $[x_i]_{\approx}$, the relations are interpreted as $(\cup_{i \geq 0} H_i)/_{\approx}$, and ρ is the image of ϱ under h. However, it is far less clear that a *finite* database D_ϱ exists with the same properties. Intuitively, different classes of $\approx$ must be merged in order to obtain a finite domain, and this must be done consistently with the H_i's. We are able to show the following key result.

▶ **Theorem 14.** *Every symbolic run* $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i\geq 0}$ *of* Γ *has an enactment* $(D_\varrho, \rho, \theta)$ *where the size of* D_ϱ *is exponential in* $|\bar{x}|$.

Proof. The roadmap of the proof is the following. We first define a normal form for artifact systems, called *linear propagation* form, requiring that the only equalities in pre-and-post conditions of services be of the form $x = x'$ where $x \in \bar{x}$ (excepting equalities with constants). We show that for every artifact system Γ we can construct an artifact system $\bar{\Gamma}$ in linear-propagation form, and a mapping h from the symbolic runs of Γ to symbolic runs of $\bar{\Gamma}$, such that from every enactment of $h(\varrho)$ one can construct an enactment of ϱ. Finally, we show that every symbolic run of an artifact system in linear-propagation form has an enactment. This is done as follows. Consider an artifact system $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ with $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ in linear propagation form. Recall that for a symbolic run $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i\geq 0}$ of Γ, we denote by $\approx$ the transitive closure of $(\cup_{i\geq 0}\epsilon_i)$ and by $[x_i]_\approx$ the class of x_i wrt $\approx$. Note that, because Γ is in linear propagation form, we can assume that $[x_i]_\approx$ contains only variables x_j. We define $span([x_i]_\approx) = \{j \mid x_j \approx x_i\}$. Clearly, each span is an interval. Next, for $x \in \bar{x}$ we define $lane(x) = \{[x_i]_\approx \mid i \geq 0\}$ (totally ordered by $[x_i]_\approx \leq [x_j]_\approx$ iff $i \leq j$). The remainder of the proof consists of defining certain finite characteristics of equivalence classes of $\approx$, so that different classes in a given lane can be collapsed if they share the same characteristics. This yields an enactment of the symbolic run over the finite database whose elements are the collapsed classes. We omit the rather involved technical details. ◀

We can now show the regularity of the service view of Γ. From Lemma 13 and Theorem 14 it follows that $\mathcal{S}_{lin}(\Gamma) = \{\mathcal{S}(\varrho) \mid \varrho \in SRuns(\Gamma)\}$. We can construct a finite-state transition system $\mathcal{F}(\Gamma)$ accepting the latter as follows:

- States: the isomorphism types (H, ϵ) of $\bar{x} \cup \bar{x}'$,
- Initial states: the states whose restrictions to $\bar{x}$ satisfy Π
- Transitions: $(H, \epsilon) \xrightarrow{\sigma} (\bar{H}, \bar{\epsilon})$ if (H, ϵ) satisfies items $(iv) - (v)$ of Definition 11 for service σ and $(H, \epsilon)|\bar{x}'$ and $(\bar{H}, \bar{\epsilon})|\bar{x}$ are identical modulo renaming $\bar{x}'$ to $\bar{x}$.

The ω-language accepted by $\mathcal{F}(\Gamma)$ consists of the sequences of transition labels in all infinite runs of the system starting from some initial state. By construction, this is precisely $\{\mathcal{S}(\varrho) \mid \varrho \in SRuns(\Gamma)\}$. Thus, we have the following.

▶ **Theorem 15.** $\mathcal{S}_{\text{lin}}(\Gamma)$ *is effectively* ω*-regular for artifact systems* Γ.

Verification vs. ω-regularity. We note that the effective ω-regularity of $\mathcal{S}_{lin}(\Gamma)$ implies decidability of LTL-FO properties of artifact systems, but is strictly stronger. Decidability of verification follows from ω-regularity because for each Γ and LTL-FO property $\xi = \forall \bar{y} \varphi_f$, and each choice of isomorphism type for $\bar{y}$, one can construct an artifact system Γ_{φ_f} and an LTL formula $\bar{\varphi}$ such that $\Gamma \models \varphi_f(\bar{y})$ iff $\mathcal{S}_{lin}(\Gamma_{\varphi_f}) \models \bar{\varphi}$. Essentially, Γ_{φ_f} is obtained, for a fixed choice of $\bar{y}$, by augmenting the artifact variables and pre-and-post conditions of each service of Γ in order to record the truth values of the FO-components of φ_f in each transition. Since a finite-state transition system specifying $\mathcal{S}_{lin}(\Gamma_{\varphi_f})$ can be effectively constructed, this reduces verification to classical finite-state LTL model-checking, and yields decidability. The converse is falsified by results of [34] which imply that artifact systems equipped with a total order do not have ω-regular service views, yet verification of LTL-FO properties is decidable.

Universal test databases. The above results, notably Theorem 14, have some potentially significant benefits for verification. We can show the following rather surprising result.

▶ **Theorem 16.** *Let Γ be a constant-free artifact system with k artifact variables. One can construct a database D^* of size double exponential in $|\bar{x}|$ such that for every constant-free LTL-FO formula ξ over Γ, $\Gamma \models \xi$ iff $Runs_{D^*}(\Gamma) \models \xi$.*

Proof. Consider an LTL-FO formula ξ over Γ. As shown in [11] (Lemma 3.3), global variables can be easily eliminated, so one can assume that $\xi = \varphi_f$. Let $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i\geq 0}$ be a symbolic run of Γ. Satisfaction of ξ by ϱ is defined similarly to actual runs, by evaluating each FO component of φ_f on the consecutive H_i/ϵ_i. Consider an enactment (D, ρ, θ) of ϱ, where $\rho = \{(\rho_i, \sigma_i)\}_{i\geq 0}$. Because H_i/ϵ_i and $D|(\rho_i \cup \rho_{i+1})$ are isomorphic, it is clear that $\varrho \models \xi$ iff $\rho \models \xi$. This in conjunction with Lemma 13 and Theorem 14 shows that $\Gamma \models \xi$ iff every symbolic run of Γ satisfies ξ. Because each symbolic run has an enactment on some database of size exponential in k, $Runs^k(\Gamma) = \cup\{Runs_D(\Gamma) \mid |D| \leq exp(k)\}$ are enactments of all symbolic runs of Γ. Thus, $\Gamma \models \xi$ iff all runs in $Runs^k(\Gamma)$ satisfy ξ. Suppose Γ and ξ are constant free. There are finitely many non-isomorphic databases of size bounded by $exp(k)$, and let D^* be their disjoint union. Clearly, $\Gamma \models \xi$ iff all runs over D^* satisfy ξ. The size of D^* is double exponential in k. ◀

Thus, D^* acts as a universal test database (akin to an Armstrong relation) for satisfaction of constant-free LTL-FO properties of Γ. In particular, a fixed D^* can be pre-computed and used to generate a counter-example run for *every* constant-free LTL-FO property violated by Γ. In contrast, the counter-example databases constructed by the algorithms in [17, 11] depend on both the specification and property. Note that, if Γ and ξ use some set C of constants, then constructing a single universal test database is no longer possible: one needs a separate database for each isomorphism type over C. Constructing the most concise test databases possible, and evaluating the practical benefits, are important issues yet to be explored.

4 Branching-Time Service Views

In this section we consider branching time service views of artifact systems. Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$. Recall the branching-time semantics of Γ, given by $TRuns^*(\Gamma)$. The *branching-time service view* of Γ, denoted $\mathcal{TS}^*(\Gamma)$, is the tree obtained from $TRuns^*(\Gamma)$ by ignoring the content of the nodes and retaining only the service labels of the edges. We use the following definition of regularity for infinite trees: $\mathcal{TS}^*(\Gamma)$ is regular if it is isomorphic to the unfolding of a finite-state transition system with edge labels (equivalently, $\mathcal{TS}^*(\Gamma)$ has finitely many non-isomorphic subtrees). Analogously to the linear case, we say that $\mathcal{TS}^*(\Gamma)$ is *effectively regular* for a class of artifact systems if such a finite-state transition system can be effectively constructed from each Γ in that class.

As we shall see, it turns out that branching-time service views of artifact systems are generally *not* regular. One might hope that regularity holds for artifacts obeying the natural feedback-free property that has proven so effective in overcoming undecidability of LTL-FO properties for specifications with dependencies and arithmetic [11]. Unfortunately, this is not the case. Indeed, we show that even very simple CTL-FO properties are not decidable for feedback-free artifacts, thus implying that their branching time service views are not effectively regular.

▶ **Theorem 17.** *It is undecidable, given an artifact system Γ and a CTL-FO formula ξ such that (Γ, ξ) is feedback-free, whether $\Gamma \models \xi$.*

Proof. The proof is by a reduction from the Post Correspondence Problem (PCP) [33]. We build upon a result of [17] (Theorem 4.2) showing that checking LTL-FO properties is

undecidable for databases satisfying a functional dependency (FD). This uses a reduction from the PCP. Next, we note that satisfaction of FDs by the database can be expressed as a CTL-FO property. Using this, we wish to mimick the reduction from the PCP that works for databases with FDs. However, there is a glitch: LTL-FO model checking is *decidable* for feedback-free specifications and properties even in the presence of FDs. Thus, a direct reduction from the linear-time case is not possible. Instead, we show how feedback freedom can be circumvented collectively by different branches of the tree of runs while being obeyed by each individual branch, and use this to adapt the PCP reduction to the branching-time framework. ◀

Similarly to the linear-time case, it can be shown that effective regularity of $\mathcal{TS}^*(\Gamma)$ implies decidability of CTL$^{(*)}$-FO properties. We therefore have the following.

▶ **Corollary 18.** *$\mathcal{TS}^*(\Gamma)$ is not effectively regular for feedback-free*[4] *artifact systems Γ.*

Note that Corollary 18 does not exclude the possibility that $\mathcal{TS}^*(\Gamma)$ might be regular. However, it says that even if this holds, a transition system defining $\mathcal{TS}^*(\Gamma)$ *cannot* be effectively constructed from each Γ.

Intuitively, feedback-freedom is ineffective in the branching-time setting because the restriction can be circumvented collectively by different branches of the tree of runs while being obeyed by each individual branch. Fortunately, specifications resulting from hierarchical design methodologies such as the Guard-Stage-Milestone discussed earlier, satisfy a stronger restriction that holds in the global run, called *global feedback freedom*. In a nutshell, global feedback freedom extends feedback freedom by having the computation graph take into account connections among variables in the entire tree of runs rather than just individual branches. We omit the technical details. We will show the following.

▶ **Theorem 19.** *$\mathcal{TS}^*(\Gamma)$ is effectively regular for globally feedback-free artifact systems Γ.*

The proof requires some technical development, which we sketch in the remainder of the section. The basic idea is to show that there are only finitely many subtrees $TRuns_D(\Gamma)$ of $TRuns^*(\Gamma)$ up to bisimulation. Moreover, each is realized by a database of bounded size, depending only on $|\bar{x}|$. Since bisimilar trees have the same branching-time service views, this establishes the theorem.

We recall the standard notion of bisimulation. Two infinite trees $\mathcal{T}, \mathcal{T}'$ with labeled edges are bisimilar if there exists a relation $\sim$ from the nodes of $\mathcal{T}$ to those of $\mathcal{T}'$ such that: (i) $root(\mathcal{T}) \sim root(\mathcal{T}')$, (ii) if $\alpha \sim \alpha'$ and $\alpha \xrightarrow{\sigma} \beta$ then there exists β' such that $\alpha' \xrightarrow{\sigma} \beta'$ and $\beta \sim \beta'$, and (iii) if $\alpha \sim \alpha'$ and $\alpha' \xrightarrow{\sigma} \beta'$ then there exists β such that $\alpha \xrightarrow{\sigma} \beta$ and $\beta \sim \beta'$.

We now present the main steps in the proof. Let $\Gamma = \langle\langle \bar{x}, \mathcal{DB}\rangle, \Sigma, \Pi\rangle$ be an artifact system, with $|\bar{x}| = k$. As in the global feedback freedom definition, we assume that service pre-and-post conditions are in $CQ^\neg$ form (conjunctions of literals). For a service $\sigma = (\pi, \psi)$ we denote by $f_\sigma(\bar{x}, \bar{y})$ the formula $\pi(\bar{x}) \wedge \psi(\bar{x}, \bar{y})$.

▶ **Definition 20.** The set $\mathcal{T}_n$ of *n-types* of $\bar{x}$ is defined inductively as follows.

- $\mathcal{T}_0 = \{\ true\ \}$
- For $n \geq 0$, $\mathcal{T}_{n+1}$ consists of all formulas of the form

$$\bigwedge_{\sigma \in \Sigma_0} \bigwedge_{\tau \in \mathcal{T}_\sigma} \exists \bar{y}(f_\sigma(\bar{x}, \bar{y}) \wedge \tau(\bar{y}))$$

where $\emptyset \neq \Sigma_0 \subseteq \Sigma$ and $\emptyset \neq \mathcal{T}_\sigma \subseteq \mathcal{T}_n$.

[4] An artifact system Γ is feedback free if $(\Gamma, true)$ is feedback free.

Let D be a database over $\mathcal{DB}$ and ν be a valuation of $\bar{x}$. It is easy to check that, for every ν that labels some node in $TRuns_D(\Gamma)$, and each $n \geq 0$, there exists a unique strongest[5] $\tau_n \in \mathcal{T}_n$ such that $D, \nu \models \tau_n$. We denote the latter by $\tau_n(D, \nu)$. It is clear that $\tau_{n+1}(D, \nu) \rightarrow \tau_n(D, \nu)$ for every $n \geq 0$.

Note that all subtrees of $TRuns_D(\Gamma)$ rooted at node labeled ν are isomorphic. Let $TRuns^{\nu}_D(\Gamma)$ be any such subtree. We will show that the sequence of types $\{\tau_n(D, \nu) \mid n \geq 0\}$ provides sufficient information to determine $TRuns^{\nu}_D(\Gamma)$ up to bisimilarity (Lemma 22). Before however, we need the following key lemma.

▶ **Lemma 21.** *Let Γ be a globally feedback-free artifact system. There exists $b > 0$, depending only on $|\bar{x}|$, such that for every database D, tuple ν labeling a node in $TRuns_D(\Gamma)$ and $n \geq 0$, $\tau_n(D, \nu)$ is equivalent to an FO sentence of quantifier rank $\leq b$.*

Using the lemma, we can prove the following.

▶ **Lemma 22.** *Let ν_1, ν_2 be valuations of $\bar{x}$ labeling nodes in $TRuns_D(\Gamma)$. If $\tau_n(D, \nu_1) = \tau_n(D, \nu_2)$ for every $n \geq 0$ then $TRuns^{\nu_1}_D(\Gamma)$ and $TRuns^{\nu_2}_D(\Gamma)$ are bisimilar.*

From Lemma 21 it follows that for every D and reachable ν, there exists $N > 0$ such that $\tau_n(D, \nu) \equiv \tau_N(D, \nu)$ for every $n \geq N$. We denote $\tau_N(D, \nu)$ by $\tau^*(D, \nu)$ and call it the *type* of ν in D. Thus, $\tau^*(D, \nu)$ is equivalent to $\{\tau_n(D, \nu) \mid n \geq 0\}$. Observe that, by Lemma 21, there are finitely many tuple types. The set of all tuple types is denoted by $\mathcal{T}$.

Finally, we define database types as follows.

▶ **Definition 23.** The type of a database D is $\tau(D) = \{\tau^*(D, \nu) \mid D \models \Pi(\nu)\}$.

We have the following.

▶ **Lemma 24.** *Let D_1 and D_2 be databases over $\mathcal{DB}$ such that $\tau(D_1) = \tau(D_2)$. Then $TRuns_{D_1}(\Gamma)$ and $TRuns_{D_2}(\Gamma)$ are bisimilar.*

Note that, since there are finitely many tuple types, there are also finitely many database types. Since a database type can be written as the conjunction of finitely many tuple types, Lemma 21 also applies to database types, and each can be written as an $\exists^*$FO sentence. Let d be the maximum number of existential quantifiers in these sentences. Thus, there are finitely many equivalence classes of trees of database runs under bisimulation, and each has a representative $TRuns_D(\Gamma)$ for some database D whose domain is of size $\leq d$. Since trees of runs equivalent under bisimulation have the same branching-time service views, it follows that $\mathcal{TS}^*(\Gamma)$ is ω-regular, and a finite-state transition system defining it can be effectively constructed from Γ. This concludes the proof of Theorem 19.

▶ Remark. Theorem 19 continues to hold for artifact systems extended with arithmetic (e.g., linear constraints over $\mathbb{Q}$). To see this, augment $\mathcal{DB}$ with a finite set $\mathcal{C}$ of relation symbols with fixed interpretations as linear constraints over $\mathbb{Q}$, and let the data domain be $\mathbb{Q}$. The definition of global freedom applies, by treating the relation symbols in $\mathcal{C}$ as arbitrary relations, and Lemma 21 carries through. Also, satisfiability of a type involving mixed data and arithmetic relations can be effectively tested: the only interaction between the two is via equality types.

[5] With respect to implication.

As noted earlier, effective regularity of the branching-time service views of a class of systems generally implies decidability of its CTL*-FO properties. In order for this to hold, we must however extend the global feedback freedom restriction to pairs (Γ, φ) where Γ is an artifact system and φ a CTL*-FO property. Taking into account the property is done similarly to feedback-freedom. We can then show the following.

▶ **Theorem 25.** *It is decidable, given an artifact system Γ and a CTL*-FO formula φ such that (Γ, φ) is globally feedback free, whether $\Gamma \models \varphi$.*

Proof. From a globally feedback-free (Γ, φ) one can construct a globally feedback-free artifact system $\bar{\Gamma}$ and a CTL* formula $\bar{\varphi}$ such that $\Gamma \models \varphi$ iff $\mathcal{TS}^*(\bar{\Gamma}) \models \bar{\varphi}$. Since $\mathcal{TS}^*(\bar{\Gamma})$ is specified by a finite-state transition system effectively constructed from Γ and φ, the result follows. ◀

5 The impact of data dependencies

In this section we consider the impact of data dependencies on the regularity of service views. We consider tuple and equality generating dependencies. We briefly recall (see [1] for details) that an equality-generating dependency (EGD) is an FO sentence of the form $\forall\bar{x}(\varphi(\bar{x}) \rightarrow y = z)$, where φ is a conjunction of relational atoms and $y, z \in \bar{x}$. A tuple-generating dependency (TGD) is a sentence of the form $\forall\bar{x}(\varphi(\bar{x}) \rightarrow \exists\bar{z}\psi(\bar{x}, \bar{z}))$, where φ and ψ are conjunctions of relational atoms. If $\bar{z}$ is empty, the dependency is called *full*; if every atom in $\psi(\bar{x}, \bar{z})$ contains an existentially quantified variable in $\bar{z}$, it is called *embedded* (note that every TGD can be written as a conjunction of full and embedded TGDs). A set of TGDs is *acyclic* if the following graph is acyclic: the nodes are database relations and there is an edge from P to Q if P occurs in the body of a TGD whose head contains Q. Inclusion dependencies are instances of TGDs and functional dependencies (FDs) are EGDs.

Linear-time service views. We first consider the impact of EGDs. Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$. For a set Δ of dependencies, $\mathcal{S}^{\Delta}_{lin}(\Gamma) = \{\mathcal{S}(\rho) \mid \rho \in \mathit{Runs}_D(\Gamma), D \models \Delta\}$. We say that $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is effectively regular for a class $\mathbf{A}$ of artifact systems and $\mathbf{D}$ of dependencies, if a Büchi automaton defining $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ can be effectively constructed from each Γ in $\mathbf{A}$ and set Δ of dependencies in $\mathbf{D}$.

We can show the following.

▶ **Theorem 26.** *$\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is not effectively ω-regular for artifact systems Γ and sets Δ of EGDs. Moreover, this holds even if Δ consists of a single FD.*

Proof. It can be shown that it is undecidable, given an artifact system Γ, a set Δ of EGDs, and a service σ, whether there exists a run of Γ on a database satisfying Δ in which service σ is used. This holds even if Δ consists of a single FD. The result follows. ◀

Note that, similarly to Corollary 18, Theorem 26 leaves open the possibility that $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ might be ω-regular.

▶ Remark. One might wonder if $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ can be characterized by some natural extension of ω-regular languages. It turns out that Theorem 26 can be extended to any family $\mathcal{L}$ of ω-languages with the following properties: (i) $\mathcal{L}$ is closed under intersection with ω-regular languages, and (ii) emptiness of languages in $\mathcal{L}$ is decidable. This assumes a finite specification mechanism for languages in $\mathcal{L}$, and that (i) is effective, i.e. the specification of the intersection of a language in $\mathcal{L}$ with the ω-language defined by a Büchi automaton must be computable. One example of such $\mathcal{L}$ is the family of ω-context-free languages, defined by infinitary extensions of pushdown automata and context-free grammars (see [6, 36]).

We now consider TGDs. Rather surprisingly, the easy case is that of embedded TGDs.

▶ **Theorem 27.** *$\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is effectively ω-regular for artifact systems Γ and sets Δ of embedded TGDs.*

Proof. It is enough to show that every symbolic run ϱ of Γ has an enactment on a database satisfying Δ. Indeed, this implies that $\mathcal{S}^{\Delta}_{lin}(\Gamma) = \mathcal{S}_{lin}(\Gamma)$, thus establishing effective ω-regularity. Let ϱ be a symbolic run of Γ. By Theorem 14, ϱ has an enactment (D, ρ, θ). Let d be some domain value not occurring in D or the constants of Γ. Observe that an extension $\bar{D}$ of D satisfying Δ can be obtained by chasing D with the TGDs in Δ so that d is used as a witness to every existentially quantified variable in the head of a TGD. Since $\bar{D}$ is an extension of D, $(\bar{D}, \rho, \theta)$ is also an enactment of ϱ. ◀

For full TGDs we have the following.

▶ **Theorem 28.** *There exists an artifact system Γ and set Δ of full TGDs such that $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is not ω-regular.*

Proof. Let the database schema of Γ consist of a binary relation R and Δ be the full TGD $\forall x \forall y \forall z\ (R(x, y) \wedge R(y, z) \rightarrow R(x, z))$, guaranteeing that R is transitive. Γ has one attribute variable x and two services *init* and *next*. The global precondition is $\neg R(0, 0) \wedge x = 0$ where 0 is a constant. The pre-condition of *init* is $x \neq 0$ and its post-condition is $x' = 0$. The pre-condition of *next* is *true* and its post-condition is $R(x, x') \wedge \neg R(x', x')$. Runs of Γ consist of stepping through R using *next*, starting from 0, using only elements which do not belong to a cycle, until *init* reinitializes x to 0 and the process is restarted. Since R is finite, $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ consists of all ω-words of the form $(next)^{n_1} \cdot init \cdot (next)^{n_2} \cdot init \cdots$ such that for each word there exists $N > 0$ for which $n_i \leq N$ for all $i \geq 1$. It is easy to see that this language, and therefore $\mathcal{S}^{\Delta}_{lin}(\Gamma)$, is not ω-regular. ◀

It turns out that effective ω-regularity is recovered for *acyclic* full TGDs.

▶ **Theorem 29.** *$\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is effectively ω-regular for artifact systems Γ and acyclic sets of full TGDs Δ.*

Proof. Recall the finite-state transition system $\mathcal{F}(\Gamma)$ used earlier to define $\mathcal{S}_{lin}(\Gamma)$. Its states consist of the isomorphism types of Γ, and edges are labeled by services. The same transition system can be viewed as defining the language $SRuns(\Gamma)$, by taking into account the isomorphism type of each state in addition to the edge labels.

Consider Δ. A *partial unfolding* of a TGD is obtained by replacing one relational atom $R(\bar{z})$ in its body by the body of any TGD in Δ with R in its head (if such exists), with appropriate renaming of variables. Let Δ^* by the closure of Δ under partial unfoldings. Obviously, Δ^* and Δ are equivalent. Because Δ is acyclic, Δ^* is finite.

The idea of the proof is to define a Büchi automaton $\mathcal{B}$ that accepts the runs of $\mathcal{F}(\Gamma)$ that are *inconsistent* with some TGD in Δ^*. Using Δ^* instead of Δ facilitates this task by allowing to ignore compositions of TGDs. Let $\xi = \forall \bar{y}(\exists \bar{z} \varphi(\bar{y}, \bar{z}) \rightarrow R(\bar{y}))$ in Δ^*. An inconsistency with ξ occurs in a symbolic run $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$ if for some $j \geq 0$ and $\bar{y} \subseteq \bar{x}_j \cup \bar{x}_{j+1}$, $\neg R(\bar{y})$ is in H_j and there exist $\bar{z} \subseteq \cup_{i \geq 0} \bar{x}_i$ such that $\varphi(\bar{y}, \bar{z})$ is satisfied by $\cup_{i \geq 0} H_i$. It can be seen, using the construction in the proof of Theorem 14, that a symbolic run is consistent with Δ^* iff it has an enactment on a database satisfying Δ.

The Büchi automaton non-deterministically guesses an inconsistency. The first component of the inconsistency, $\neg R(\bar{y})$, can be guessed by $\mathcal{B}$ whenever $\neg R(\bar{y})$ is in H_j for the current j. To enable checking the second component of an inconsistency, the states of $\mathcal{B}$ also contain

variables $\bar{z}$. The values of the variables $\bar{z}$ are non-deterministically guessed throughout the run, and the connections between them, as specified by the isomorphism types, are recorded. A run is accepted whenever $\neg R(\bar{y})$ and $\varphi(\bar{y}, \bar{z})$ hold for some TGD and guessed $\bar{y}$ and $\bar{z}$. The set of symbolic runs consistent with Δ^* is then $SRuns(\Gamma) \cap \mathcal{B}^c$, where $\mathcal{B}^c$ is the complement of $\mathcal{B}$. Finally, $\mathcal{S}^{\Delta^*}_{lin}(\Gamma) = h(SRuns(\Gamma) \cap \mathcal{B}^c)$, where h is the homomorphism removing the isomorphism types and retaining just the service names. Since ω-regular languages are closed under complement, intersection, and homomorphism (with effective constructions), $\mathcal{S}^{\Delta^*}_{lin}(\Gamma)$ is effectively ω-regular. ◀

We finally consider feedback-free artifact systems. Recall that these are particularly well-behaved with respect to verification. In particular, while model-checking is undecidable for artifact systems in the presence of FDs, it becomes decidable for feedback-free systems [11]. One might hope that feedback-free systems are similarly well-behaved with respect to linear service views. Indeed, in contrast to Theorems 26 and 28, we have the following.

▶ **Theorem 30.** *$\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is effectively ω-regular for feedback-free artifact systems Γ and sets Δ of EGDs and full TGDs.*

Proof. The approach is similar to that of [11] for showing decidability of model-checking. Consider a symbolic run $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$ of Γ. For each $i \geq 0$, let $\nu_i(\bar{x}_i)$ be the formula $\exists \bar{x}_0 \ldots \exists \bar{x}_{i-1}(\Pi(\bar{x}_0) \wedge \bigwedge_{0 \leq j < i} \sigma_j(\bar{x}_j, \bar{x}_{j+1}))$. Intuitively, $\nu_i(\bar{x}_i)$ completely specifies the constraints placed on $\bar{x}_i$ by the first i transitions. Let $\Phi = \{\exists \bar{x}_i \nu_i(\bar{x}_i) \mid i \geq 0\}$. It can be shown that there exists an enactment of ϱ on a database D satisfying Δ iff $D \models \Phi \cup \Delta$ (this uses the finiteness of D and a pigeonhole argument). As shown in [11], because Γ is feedback-free, each formula in Φ can be rewritten as a formula of quantifier rank bounded by $|\bar{x}|^2$. Since there are finitely many non-equivalent formulas of bounded quantifier rank [26], Φ is equivalent to a single $\exists$FO formula φ. Moreover, because all formulas in Δ are universally quantified, if ϱ has an enactment on a database satisfying Δ, it also has an enactment on such a database whose domain is bounded by the number of variables (say v) in φ. Thus, $\mathcal{S}^{\Delta}_{lin}(\Gamma) = \cup\{\mathcal{S}_{lin}(Runs_D(\Gamma)) \mid D \models \Delta, |dom(D)| \leq v\}$. Since each $\mathcal{S}_{lin}(Runs_D(\Gamma))$ is ω-regular, $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ is effectively ω-regular. ◀

Branching-time service views. We now consider briefly the impact of data dependencies on branching-time service views. Recall that these views are not regular, even for feedback-free systems. However, by Theorem 19, the views are regular for globally feedback-free systems.

Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$. For a set Δ of dependencies over $\mathcal{DB}$, $TRuns^*_{\Delta}(\Gamma)$ is the tree obtained by placing all $TRuns_D(\Gamma)$ under a common root, where $D \models \Delta$. The branching-time service view, denoted $\mathcal{TS}^*_{\Delta}(\Gamma)$, is obtained as before from $TRuns^*_{\Delta}(\Gamma)$ by ignoring the content of the nodes and retaining only the service labels of the edges.

In the presence of data-dependencies, we have the following.

▶ **Theorem 31.** *$\mathcal{TS}^*_{\Delta}(\Gamma)$ is effectively regular for globally feedback-free artifact systems Γ and sets Δ of EGDs and full TGDs.*

Proof. Recall the proof of Theorem 19 and the formulas $\exists^*$FO defining database types, whose number of existential quantifiers is bounded by some b depending only on Γ. Note that the EGDs and full TGDs in Δ can be expressed by a sentence in $\forall^*$FO. Suppose there is a database D of type τ satisfying Δ. Then there exists $D_0 \subseteq D$, whose domain consists of b witnesses to the existentially quantified variables of τ, that also has type τ and satisfies Δ. Thus, every database type that includes an instance satisfying Δ, also has a

representative satisfying Δ whose domain is bounded by b. It follows that $\mathcal{TS}^*_\Delta(\Gamma)$ is regular, and a specification can be effectively constructed from Γ and Δ. ◀

▶ Remark. Theorem 31 alternatively holds for sets Δ of EGDs and arbitrary TGDs (full and embedded), as long as the set of TGDs is acyclic.

6 Conclusions

We considered the problem of extracting process-centric views from highly declarative, data-driven workflows. Classical process-centric workflow specification frameworks provide a variety of means for describing the valid sequences (or trees) of events in the workflow, with finite-state transition diagrams at their core. We considered views consisting of the sequences of services applied during linear or branching-time runs of an artifact system. The results establish when such views are regular and can be specified effectively by finite-state transition systems. Thus, we showed that linear-time service views are regular, while branching-time views are regular only under certain restrictions (satisfied naturally by systems produced by hierarchical design methodologies in the spirit of GSM). We also considered the impact of data dependencies (tuple and equality generating dependencies) on regularity of views. We showed that linear-time views are no longer regular in presence of FDs or cyclical full TGDs, but remain regular with acyclic or embedded TGDs. Regularity of branching-time service views is preserved in the presence of EGDs and full TGDs.

Our results also have some interesting connections to verification. For instance, the techniques developed to show regularity of linear-time views yield potentially more efficient ways to generate counterexample databases witnessing violation of LTL-FO properties. As a side-effect of results on branching-time service views, we showed that CTL-FO properties are undecidable for artifact systems, but model-checking CTL*-FO becomes decidable under the same restrictions guaranteeing regularity of branching-time views.

Several interesting questions remain to be investigated. If a class of declarative workflows does not have regular service views, two courses of action are plausible. First, one might seek an extension of regular languages powerful enough to describe the views while remaining palatable to users. Alternatively, one might opt for a regular approximation of the view, resulting from relaxations that users are likely to find reasonable. In all cases, the views could be made more expressive and informative by augmenting the purely process-centric specifications with light-weight annotations on transitions with conditions on the data, in the spirit of BPEL and YAWL [38]. Besides the technical problems *per se*, this brings into play interesting HCI and usability issues.

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

2 F. Belardinelli, A. Lomuscio, and F. Patrizi. An abstraction technique for the verification of artifact-centric systems. In *Proc. Intl. Conf. on Knowledge Representation*, 2012.

3 K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F. Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Sys. Journal*, 46(4), 2007.

4 K. Bhattacharya et al. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Systems Journal*, 44(1), 2005.

5 BizAgi and Cordys and IBM and Oracle and SAP AG and Singularity (OMG Submitters) and Agile Enterprise Design and Stiftelsen SINTEF and TIBCO and Trisotech (Co-Authors). Case Management Model and Notation (CMMN), FTF Beta 1, Jan. 2013. OMG Document Number dtc/2013-01-01, Object Management Group.

6 L. Boasson and M. Nivat. Adherences of languages. *J. Comput. System Sci.*, 20(3), 1980.

7 A. Bozzon, M. Brambilla, S. Ceri, and A. Mauri. Reactive crowdsourcing. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 153–164, 2013.

8 A. Bozzon, M. Brambilla, S. Ceri, A. Mauri, and R. Volonterio. Pattern-based specification of crowdsourcing applications. In *Web Engineering, 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings*, pages 218–235, 2014.

9 T. Chao et al. Artifact-based transformation of IBM Global Financing: A case study. In *BPM*, 2009.

10 E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 2000.

11 E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. *ACM Transactions on Database Systems*, 37(3), 2012. Preliminary version in *ICDT 2011*.

12 E. Damaggio, R. Hull, and R. Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. *Information Systems*, 38:561–584, 2013.

13 G. De Giacomo, R. De Masellis, and R. Rosati. Verification of conjunctive artifact-centric services. *Int. J. Cooperative Inf. Syst.*, 21(2):111–140, 2012.

14 H. de Man. Case management: Cordys approach. BP Trends (`www.bptrends.com`), 2009.

15 S. Demri and R. Lazić. LTL with the Freeze Quantifier and Register Automata. In *LICS*, 2006.

16 S. Demri, R. Lazić, and A. Sangnier. Model checking freeze LTL over one-counter automata. In *FoSSaCS*, 2008.

17 A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, 2009.

18 A. Deutsch, R. Hull, and V. Vianu. Automatic verification of data-driven systems. *Sigmod Record*, 2014.

19 A. Deutsch, Y. Li, and V. Vianu. Hierarchical artifact systems. In preparation.

20 A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web applications. *JCSS*, 73(3):442–474, 2007.

21 E. Allen Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 995–1072. North-Holland Pub. Co./MIT Press, 1990.

22 B. Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of relational data-centric dynamic systems with external services. In *PODS*, 2013.

23 R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F. Heath III, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In *ACM DEBS*, 2011.

24 Dimitri Isaak and Christof Löding. Efficient inclusion testing for simple classes of unambiguous -automata. *Inf. Process. Lett.*, 112(14-15), 2012.

25 S. Kumaran, P. Nandi, T. Heath, K. Bhaskaran, and R. Das. ADoc-oriented programming. In *Symp. on Applications and the Internet (SAINT)*, 2003.

26 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

27 A. Lomuscio and J. Michaliszyn. Model checking unbounded artifact-centric systems. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*, 2014.

28 M. Marin, R. Hull, and R. Vaculín. Data centric BPM and the emerging case management standard: A short survey. In *BPM Workshops*, 2012.

29 S. Merz. Model checking: a tutorial overview. In *Modeling and verification of parallel processes*. Springer-Verlag New York, 2001.

30 Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, 1967.

31 A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3), 2003.

32 Amir Pnueli. The temporal logic of programs. In *FOCS*, 1977.

33 E. L. Post. Recursive unsolvability of a problem of Thue. *J. of Symbolic Logic*, 12:1–11, 1947.

34 L. Segoufin and S. Torunczyk. Automata based verification over linearly ordered data domains. In *STACS*, 2011.

35 M. Spielmann. Verification of relational transducers for electronic commerce. *JCSS.*, 66(1):40–65, 2003.

36 Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*. Elsevier, 1990.

37 W. van der Aalst and M. Song. Mining social networks: Uncovering interaction patterns in business processes. In *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer Berlin Heidelberg, 2004.

38 W. van der Aalst and A. ter Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4), 2005.

39 W.-D. Zhu et al. Advanced Case Management with IBM Case Manager. Available at `http://www.redbooks.ibm.com/abstracts/sg247929.html?Open`.