Process-Centric Views of Data-Driven Business Artifacts[☆]

Adrien Koutsos

ENS Cachan, France

Victor Vianu

UC San Diego & INRIA-Saclay

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 stakeholders 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. As a side effect, we obtain several new results on verification of business artifacts, including a decidability result for branching-time properties.

Keywords: Workflows, data-aware, process-centric, views

[☆]A preliminary version of this article appeared in the *Intl. Conf. on Database Theory* (ICDT) 2015 [28].

Email addresses: adrien.koutsos@ens-cachan.fr (Adrien Koutsos), vianu@cs.ucsd.edu (Victor Vianu)

1. Introduction

Data-driven workflows have become ubiquitous in a variety of application domains, including business, government, science, health-care, social networks [42], crowdsourcing [6, 7], 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 [36, 29], 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, 15, 44]. In particular, the Guard-Stage-Milestone (GSM) approach [13, 26] to artifact specification uses declarative services with pre- and post-conditions, concurrency, and hierarchy. The OMG standard for Case Management Model and Notation (CMMN) [11], announced last year, draws key foundational elements from GSM[32].

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 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 [19]).

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 such specifications for business artifacts in an intuitive, familiar process-centric language is often desirable. However, although they differ in the specific contructs, all common process-centric languages are finite-state, and can only specify regular languages. We are therefore interested in understanding

when business artifacts can be faithfully represented by regular languages of sequences of tasks.

Specifically, in this paper we study views of business artifact runs consisting of the sequences of services applied in each run[1]. The views come 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.

Example 1. *To illustrate linear-time service views of declarative workflows, we consider a variant of the running example of [12], specified in Section 2.1. 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 (we discuss the use of infinite runs in Section 2). 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. 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*.

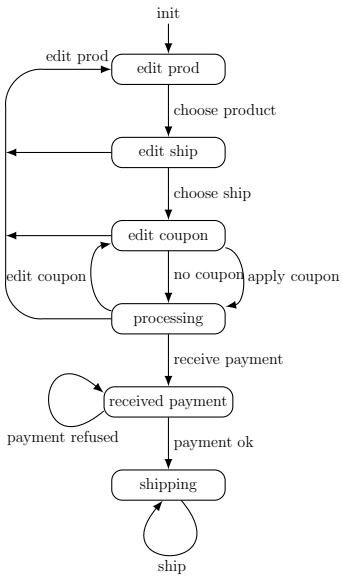


Figure 1: A linear-time process-centric view.

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 [18, 12], and several natural restrictions derived from the GSM design methodology, some of which have been considered in the context of verification [12]. 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 restriction previously used in the context of verification, called *feedback freedom* [12]. 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 obeyed by natural examples encountered in our collaboration with IBM, such as those discussed in [12]. It is also satisfied by a model abstracting core elements of GSM, studied in [20] (see Remark 20).

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 quantifier-free FO formulas on the tuple artifact and the underlying database. This yields the logics LTL(QFO),

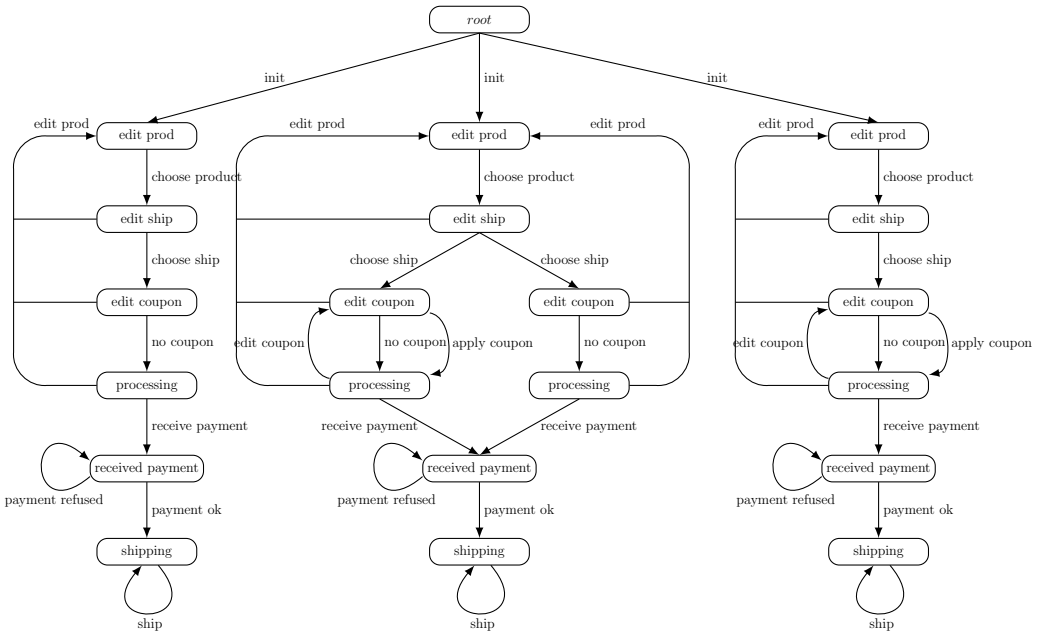


Figure 2: A branching-time process-centric view.

CTL(QFO) and CTL*(QFO) (denoted by LTL-FO, CTL-FO and CTL*-FO in [18, 12, 21]). 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(QFO), resp. CTL*(QFO) properties for that class. The converse is false: decidability of LTL(QFO) or CTL*(QFO) 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*(QFO) properties are decidable for globally feedback-free artifact systems. On the other hand, we show that CTL(QFO) 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 [18, 12], given an artifact specification $\mathcal{A}$ and an LTL(QFO) 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, 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(QFO) properties violated by $\mathcal{A}$. Pre-computing such a database may allow more efficient generation of counter-examples for specific LTL(QFO) properties.

Decidability results on verification of branching-time properties of infinite-state artifact systems are scarce and require significant restrictions. In [23, 14], 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 [31] for the case when the database consists of a single unary relation.

The present article is the journal version of the preliminary conference paper [28]. It contains the full technical development, including all proofs and an extended example.

2. Background

After some basic definitions, we review the tuple artifact model, introduce the temporal logics LTL(QFO) and CTL$^{(*)}$(QFO), and recall some previous results on verification of temporal properties.

We assume an infinite data domain **dom**. A *relational schema* is a finite set of relation symbols with associated arities. A *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, 30]). FO formulas may use equality and constants from **dom**.

2.1. Artifact systems

We use the *tuple artifact model* introduced in [12], capturing the core elements of the artifact model. 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 1. An *artifact schema* is 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 2. 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 3. 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 $\bar{x}, \bar{x}'$.

[2] $\exists$FO conditions can be easily simulated by additional artifact variables.

Definition 4. 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 5. 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 6. 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 $D \models \Pi(\rho_0)$.

As in previous work, we take runs to be infinite because many data-centric processes run forever (e.g. supply chain management workflows, e-commerce, e-government and e-health applications, etc). Also, as observed in the contex of verification, infinite runs capture information lost by their finite prefixes. The assumption that runs are infinite is not a restriction even for terminating business processes, since finite runs can be artificially extended to infinite runs by adding self-looping transitions.

The linear-time semantics of an artifact system Γ is provided by the set of all runs. Specifically, we denote by $Runs_D(\Gamma)$ the set of all runs of Γ on database instance D, and by $Runs(\Gamma)$ the union of $Runs_D(\Gamma)$ 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, $TRuns_D(\Gamma)$ 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 $TRuns^*(\Gamma)$ is the tree obtained by placing all trees $TRuns_D(\Gamma)$ (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 $Runs(\Gamma)$.

Example 2. *We provide our running example of an artifact system. This is a simplified version of the example in [12], modeling a basic e-commerce business process. 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 artifact system has the following variables:

`status`, `prod_id`, `ship_type`, `coupon`, `amount_owed`, `amount_paid`.

The `status` *variable tracks the status of the order and can take the following values:*

"edit_prod", "edit_ship", "edit_coupon", "processing", "received_payment", "shipping".

The database schema has the following tables:

`PRODUCTS`(`id`),
`COUPON-DISCOUNTS`(`code`, `prod_id`, `ship_type`, `price`),
`SHIPPING`(`ship_type`, `prod_id`, `price`),

In the example, we allow as syntactic sugar existentially quantified variables in conditions, which can be simulated using additional artifact variables.

The starting configuration has `status` *initialized to "edit_prod", and all other variables to "undefined", modeled by the reserved constant λ. This is easily expressed by the artifact system's pre-condition Π:*

$$\Pi \ : \ \texttt{status} = \textit{“edit\_prod”} \wedge \texttt{prod\_id} = \lambda \wedge \texttt{ship\_type} = \lambda \\ \wedge \ \texttt{coupon} = \lambda \wedge \texttt{amount\_owed} = \lambda \wedge \texttt{amount\_paid} = \lambda \\ \wedge \ \texttt{amount\_refunded} = \lambda$$

The services. *The following services model a few of the business process tasks.*

choose_product *The customer chooses a product among the available ones.*

$\pi : \texttt{status} = \textit{“edit\_prod”}$
$\psi : \texttt{PRODUCTS}(\texttt{prod\_id}') \wedge \texttt{status}' = \textit{”edit\_ship”}$

choose_ship *The customer chooses a compatible shipping option.*

$\pi : \texttt{status} = \textit{“edit\_ship”}$
$\psi : \exists x(\texttt{SHIPPING}(\texttt{ship\_type}', \texttt{prod\_id}, x) \wedge \texttt{status}' = \textit{“edit\_coupon”}$
$\wedge \texttt{prod\_id}' = \texttt{prod\_id})$

no_coupon *The customer does not use a coupon number.*

$\pi : \texttt{status} = \textit{“edit\_coupon”}$
$\psi : \texttt{coupon}' = \lambda \wedge \texttt{SHIPPING}(\texttt{ship\_type}, \texttt{prod\_id}, \texttt{amount\_owed}') \wedge$
$\texttt{status}' = \textit{“processing”} \ \wedge \texttt{prod\_id}' = \texttt{prod\_id} \ \wedge$
$\texttt{ship\_type}' = \texttt{ship\_type}$

apply_coupon *The customer inputs a coupon code.*

$\pi : \texttt{status} = \textit{“edit\_coupon”}$
$\psi : \texttt{COUPON-DISCOUNTS}(\texttt{coupon}', \texttt{prod\_id}, \texttt{ship\_type}, \texttt{amount\_owed}')$
$\wedge \texttt{status}' = \textit{“processing”} \wedge \texttt{prod\_id}' = \texttt{prod\_id} \wedge \texttt{ship\_type}' = \texttt{ship\_type})$

edit_product *The customer returns to the product choice stage.*

π :status = *"edit_shiptype"* $\vee$ status = *"edit_coupon"* $\vee$
status = *"processing"*
ψ : status$'$ = *"edit_prod"*

edit_coupon *The customer returns to the coupon choice stage. The precondition checks that there is a coupon applicable for this product and shipment type.*

π : status = *"processing"*
$\exists x, y(\texttt{COUPON-DISCOUNTS}(x, \texttt{prod\_id}, \texttt{ship\_type}, y)\wedge$
ψ :status$'$ = *"edit_coupon"* $\wedge$ prod_id$'$ = prod_id$\wedge$
ship_type$'$ = ship_type

receive_payment *The customer sends a payment.*

π : status = *"processing"*
ψ :amount_paid$'$ $\neq \lambda \wedge$ status$'$ = *"received_payment"*$\wedge$
prod_id$'$ = prod_id $\wedge$ ship_type$'$ = ship_type$\wedge$
coupon$'$ = coupon $\wedge$ amount_owed$'$ = amount_owed

payment_refused *The amount sent was not correct. The customer sends a new payment.*

π : status = *"received_payment"* $\wedge$ amount_paid $\neq$ amount_owed
ψ :status$'$ = *"received_payment"* $\wedge$ amount_paid$'$ $\neq \lambda$
prod_id$'$ = prod_id $\wedge$ ship_type$'$ = ship_type$\wedge$
coupon$'$ = coupon $\wedge$ amount_owed$'$ = amount_owed

payment_ok *Check payment and ship the product.*

π : status = *"received_payment"* $\wedge$ amount_paid = amount_owed
ψ :status$'$ = *"shipping"* $\wedge$ amount_paid$'$ = amount_paid$\wedge$
prod_id$'$ = prod_id $\wedge$ ship_type$'$ = ship_type$\wedge$
coupon$'$ = coupon $\wedge$ amount_owed$'$ = amount_owed

ship *A paid order gets shipped.*

π : status = *"shipping"*
ψ :status$'$ = *"shipping"* $\wedge$ prod_id$'$ = prod_id$\wedge$
ship_type$'$ = ship_type $\wedge$ coupon$'$ = coupon$\wedge$
amount_owed$'$ = amount_owed$\wedge$
amount_paid$'$ = amount_paid

Observe that **ship** *can be applied forever, thus allowing infinite runs.*

2.2. Constrained artifact systems

In addition to the basic (unconstrained) artifact systems defined earlier, we shall also study artifact systems whose underlying database is constrained by a set Δ of dependencies. We will consider tuple and equality generating dependencies (referred to collectively as *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. Note that inclusion dependencies are instances of TGDs and functional dependencies (FDs) are EGDs.

A *constrained artifact system* is an expression $\Gamma(\Delta)$, where Γ is an artifact system and Δ a set of dependencies over the database schema $\mathcal{DB}$ of Γ. The set of linear runs of $\Gamma(\Delta)$ is defined as $\mathit{Runs}(\Gamma(\Delta)) = \cup\{\mathit{Runs}_D(\Gamma) \mid D \models \Delta\}$. Similarly, the global tree of runs $\mathit{TRuns}^*(\Gamma(\Delta))$ is the tree obtained by placing all trees $\mathit{TRuns}_D(\Gamma)$ (for every database D satisfying Δ) under a common root, connected by edges with a special label *init*.

In our running example, the database schema can be naturally enriched with dependencies as follows (the underlined attributes are keys in each relation):

PRODUCTS(<u>id</u>),
COUPON-DISCOUNTS(<u>code, prod_id, ship_type</u>, price),
SHIPPING(<u>ship_type, prod_id</u>, price),

One might also add natural foreign key constraints expressed as TGDs (e.g. `prod_id` in `COUPON-DISCOUNTS` references `id` in `PRODUCTS`).

In order to clearly distinguish basic artifact systems from constrained artifact systems, we sometimes refer to the former as *unconstrained artifact systems*.

2.3. 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. LTL is propositional logic augmented with temporal operators **G** (always), **F** (eventually), **X** (next) and **U** (until) (e.g., see [37]). Informally, $\mathbf{G}p$ says that p holds at all times in the run, $\mathbf{F}p$ says that p will eventually hold, $\mathbf{X}p$ says that p holds at the next configuration, and $p\mathbf{U}q$ says that q will hold at some point in the future and p holds up to that point. For example, $\mathbf{G}(p \rightarrow \mathbf{F}q)$ says that whenever p holds, q must hold sometime in the future.

More precisely, the semantics of an LTL formula over set P of propositions is defined as follows. LTL specifies properties of infinite words (ω-words) $\{\tau_i\}_{i\geq 0}$ over the alphabet consisting of truth assignments to P. Let $\tau_{\geq j}$ denote $\{\tau_i\}_{i\geq j}$, for $j \geq 0$.

The semantics of the temporal operators **X**, **U** is the following (where $\models$ denotes satisfaction and $j \geq 0$):

- $\tau_{\geq j} \models \mathbf{X}\varphi$ iff $\tau_{\geq j+1} \models \varphi$,
- $\tau_{\geq j} \models \varphi\mathbf{U}\psi$ iff $\exists k \geq j$ such that $\tau_{\geq k} \models \psi$ and $\tau_{\geq l} \models \varphi$ for $j \leq l < k$.

The **G** and **F** operators are defined as follows: $\mathbf{F}\varphi \equiv \mathit{true}\ \mathbf{U}\ \varphi$ and $\mathbf{G}\varphi \equiv \neg(\mathbf{F}\neg\varphi)$.

The extension of LTL used in [12], 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 denote this logic by LTL(QFO). This differs from (and is more accurate than) the notation LTL-FO used in [12] and other previous works.

Definition 7. Let $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ be an artifact schema. A *QFO component* over $\mathcal{A}$ is a quantifier-free FO formula over $\mathcal{DB}$. An LTL(QFO) formula over $\mathcal{A}$ is an expression $\forall \bar{y} \varphi_f$, where:

(i) φ is an LTL formula with propositions P;

(ii) f is a mapping from P to QFO components over $\mathcal{A}$

(iii) φ_f is obtained by replacing each $p \in P$ with $f(p)$;

(iv) $\bar{y}$ is the set of variables occurring in φ_f that are different from $\bar{x} \cup \bar{x}'$.

The semantics of LTL(QFO) formulas is defined as follows. Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, $\xi = \forall \bar{y} \varphi_f$ an LTL(QFO) formula over $\mathcal{A}$, and ρ a run of Γ on database D. Let μ be a valuation of $\bar{y}$ into **dom**. A QFO component $\psi(\bar{x}, \bar{x}', \bar{y})$ of φ_f is satisfied in ρ_i with valuation μ if $D \models \psi(\rho_i, \rho_{i+1}, \mu)$, $i \geq 0$. The run ρ satisfies φ_f with valuation μ if $\{\sigma(\rho_i)\}_{i \geq 0} \models \varphi$, where $\sigma(\rho_i)$ is the truth assignment for P in which p is true iff $f(p)$ is satisfied in ρ_i with valuation μ. Finally, $\rho \models \forall \bar{y} \varphi_f$ if $\rho \models \varphi_f$ with every valuation μ of $\bar{y}$ into **dom**.

We say that an unconstrained artifact system Γ satisfies an LTL(QFO) sentence ξ, denoted $\Gamma \models \xi$, if $\rho \models \xi$ for every $\rho \in \mathit{Runs}(\Gamma)$. Similarly, a constrained artifact system $\Gamma(\Delta)$ satisfies an LTL(QFO) sentence ξ, denoted $\Gamma(\Delta) \models \xi$, if $\rho \models \xi$ for every $\rho \in \mathit{Runs}(\Gamma(\Delta))$.

We illustrate LTL(QFO) with a simple example.

Example 3. *The following LTL(QFO) formula $\forall x \varphi_f$ specifies a desirable business rule for the artifact system of Example 2, using variables* `amount_paid`, `amount_refunded`, `status`, `amount_owed`, *as well as the global variable x.*

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

In more detail, φ is the LTL formula $\mathbf{G}(p \rightarrow \mathbf{F}q)$ and f is defined by

$$\begin{array}{rcl} f(p) & = & \texttt{amount\_paid} = x \wedge \texttt{amount\_paid} = \texttt{amount\_owed} \\ f(q) & = & \texttt{status} = \textit{"shipped"} \vee \texttt{amount\_refunded} = x \end{array}$$

Property $\forall x \varphi_f$ 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 the universally quantified global variable x to relate the value of paid and refunded amounts across distinct steps in the run sequence.

The branching-time extensions CTL(QFO) and CTL*(QFO) 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 [22]).

We note that variants of LTL(QFO) have been introduced in [22, 40] (denoted by LTL-FO). 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 [16, 17].

2.4. Verification of artifact systems

We informally review the results of [18, 12] on verification of LTL(QFO) 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, 34]). 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 a *symbolic* representation of runs rather than the actual runs. This yields the following result.

Theorem 4. *[18] It is* PSPACE*-complete*[3] *to check, given an artifact system* Γ *and an LTL(QFO) property* ξ*, whether* Γ *satisfies* ξ*.*

Unfortunately, Theorem 4 fails even in the presence of simple data dependencies or arithmetic. Specifically, as shown in [18, 12], verification becomes undecidable as soon as 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 1 the value of some atributes. Hence, a restriction is needed to achieve decidability for these extensions. We discuss this next.

Feedback Freedom 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

[3]For fixed database schema.

counter machines, for which the problem of state reachability is known to be undecidable [35]. 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 [12], 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*, reviewed below. It turns out that artifact systems designed in a hierarchical fashion by successive refinement, in the style of the Guard-Stage-Milestone model, naturally satisfy the feedback freedom condition (so there is no need for testing). Indeed, there is evidence that the feedback freedom condition is permissive enough to capture a wide class of applications of practical interest. As discussed in [12], this is confirmed by numerous examples of real-life business processes modeled as artifact systems, encountered in IBM's practice.

We recall next the definition of feedback freedom from [12]. Feedback freedom is defined for a pair $(\Gamma, \forall\bar{y}\varphi_f)$, where Γ is an artifact system and $\forall\bar{y}\varphi_f$ an LTL(QFO) formula over Γ. In the absence of a property, Γ is feedback free if (Γ, true) is feedback free.

Feedback freedom is formalized using a notion of symbolic run consisting of an infinite sequence of consecutive occurrences of artifact variables, together with formulas associated to each transition from one occurrence to the next. The formulas capture the pre-and-post condition of the service applied at each transition in the run, and also specify which QFO components of φ_f are satisfied.

More formally, let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$, and $\forall\bar{y}\varphi_f$ be an LTL(QFO) formula. We denote by $\mathrm{CQ}^\neg$ quantifier-free conjunctions of literals (positive and negated atoms over $\mathcal{DB} \cup \{=\}$).

We associate to each $x \in \bar{x}$ an infinite set of new variables $\{x_i\}_{i>0}$, and we denote $\bar{x}_i = \{x_i \mid x \in \bar{x}\}$. A *symbolic constraint run* ϱ consists of a sequence $\{\psi_i(\bar{x}_i, \bar{x}_{i+1}, \bar{y})\}_{i\geq 0}$ where each $\psi_i(\bar{x}_i, \bar{x}_{i+1}, \bar{y})$ is a $\mathrm{CQ}^\neg$ formula over $\mathcal{A}$ with

variables among $\bar{x}_i \cup \bar{x}_{i+1} \cup \bar{y}$. The formulas ψ_i are obtained from Σ and φ_f as follows. We first define the sets of CQ$^\neg$ formulas below, that capture symbolically the possible transitions in Γ, together with truth assignments to the set of propositions P in φ, expanded in φ_f via f.

1. Δ_Σ. For each service $\langle \pi, \psi \rangle \in \Sigma$, consider the formula ξ obtained from $\pi \wedge \psi$ by putting it in DNF. For each such formula, Δ_Σ contains all disjuncts of ξ.
2. Δ_{φ_f} contains, for each $\sigma \in 2^P$, all disjuncts of the DNF of the formula

$$\bigwedge_{\sigma(p)=1} f(p) \ \wedge \ \bigwedge_{\sigma(p)=0} \neg f(p).$$

A *symbolic transition template* is a conjunction $\psi(\bar{x}, \bar{x}', \bar{y})$ of one formula from Δ_Σ and one from Δ_{φ_f}. Intuitively, the formula chosen from Δ_Σ corresponds to a transition caused by one of the services in Σ, while the formula chosen from Δ_{φ_f} determines a truth assignment σ for the QFO components of φ_f. Note that there are finitely many such formulas associated with Δ_Σ and Δ_{φ_f}. For $i > 0$, each formula ψ_i in the symbolic constraint run is obtained from some symbolic transition template $\psi(\bar{x}, \bar{x}', \bar{y})$ by replacing $\bar{x}$ with $\bar{x}_i$ and $\bar{x}'$ with $\bar{x}_{i+1}$. We refer to ψ_i as a *symbolic transition* generated by ψ. For $i = 0$, ψ_0 is obtained by taking the conjunction of a formula obtained as above with a formula accounting for the pre-condition Π (specifically, a disjunct of the DNF of Π in prenex form, where $\bar{x}$ is replaced with $\bar{x}_0$).

To formalize the feedback-free condition, we associate two undirected graphs G_ψ and E_ψ to each symbolic transition template $\psi = \phi(\bar{x}, \bar{x}', \bar{y})$. The graph G_ψ captures all connections among variables occurring together in the same atom, whereas E_ψ captures equalities alone. Specifically, G_ψ consists of the restriction to $\bar{x}, \bar{x}', \bar{y}$ of the graph containing an edge among every two variables occurring together in an atom of ψ, and E_ψ is the restriction to $\bar{x}, \bar{x}', \bar{y}$ of the transitive closure of the graph containing an edge among every two variables in ψ that appear together in an equality atom of ψ.

Similarly, we define for each symbolic transition ψ_i the graphs E_{ψ_i} and G_{ψ_i} by replacing $\bar{x}$ by $\bar{x}_i$ and $\bar{x}'$ with $\bar{x}_{i+1}$ in E_ψ and G_ψ. Given a symbolic constraint run $\varrho = \{\psi_i\}_{i \geq 0}$, we define $G_\varrho = \cup_{i \geq 0} G_{\psi_i}$ and E_ϱ as $\cup_{i \geq 0} E_{\psi_i}$. We also denote by E^*_ϱ the transitive closure of E_ϱ. The graphs associated with finite symbolic constraint runs are defined analogously.

Clearly, E^*_ϱ is an equivalence relation on the variables of ϱ (its equivalence classes are the connected components of E_ϱ). For each variable x_i, we

denote by $[x_i]$ its equivalence class with respect to E^*_ϱ. The *span* of an equivalence class $[x_i]$ is defined as $span([x_i]) = \{j \mid x_j \in [x_i]\}$. It is clear that $span([x_i])$ is always an interval (possibly infinite). For each variable $y \in \bar{y}$, $span(y) = [0..\infty]$.

Definition 8. $(\Gamma, \forall \bar{y} \varphi_f)$ is *feedback-free* if for every symbolic constraint run prefix $\varrho = \{\psi_i\}_{i \leq n}$, each path from x_i to x_j in G_ϱ contains a node z such that $span([x_i]) \cup span([x_j]) \subseteq span([z])$.

Thus, feedback freedom allows data flow between different occurrences x_i, x_j of a given artifact variable x, but only if the flow is "cut" by some intermediate variable z whose value remains unchanged during the entire lifetimes of x_i and x_j. This restriction prevents dangerous recursive computation leading to undecidability.

It is shown in [12] that feedback freedom of an artifact system together with an LTL(QFO) property can be checked in PSPACE, by reduction to a test of emptiness of a two-way alternating finite-state automaton.

Finally, an artifact system Γ is feedback free if (Γ, true) is feedback free. It can be seen that, similarly to the running example of [12], the artifact system in Example 2 is feedback free.

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 5. *[12] It is decidable, given an artifact system Γ using arithmetic (linear inequalities with integer coefficients) and whose database satisfies a set of key and foreign key constraints, and an LTL(QFO) property ξ such that (Γ, ξ) is feedback free, whether every run of Γ on a valid database satisfies ξ.*

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. We begin by recalling some basics on languages on infinite words. Let Θ be a finite alphabet. An ω-word over Θ is an infinite sequence of symbols from Θ 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 [41]).

A Büchi automaton $\mathcal{B}$ is a non-deterministic finite-state automaton accepting the set of infinite words for which some run of the automaton goes through an accepting state infinitely often, denoted $\mathcal{L}(\mathcal{B})$. A Büchi automaton in which every state is accepting is also referred to as a finite-state transition system (or safety automaton, see [27]). 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.

Definition 9. 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 $\mathcal{S}_{lin}(\Gamma) = \{\mathcal{S}(\rho) \mid \rho \in \mathit{Runs}(\Gamma)\}$. A class $\mathbf{A}$ of artifact systems has effectively ω-regular service views if there is an algorithm that produces, for each Γ in $\mathbf{A}$, a Büchi automaton defining $\mathcal{S}_{lin}(\Gamma)$.

We will show that unconstrained artifact systems have effectively ω-regular service views, and consider constrained artifact systems in Section 5. To show effective regularity for unconstrained artifact systems, we will use a symbolic representation of the runs of Γ. Intuitively, each transition in a run is represented by the isomorphism type of the database restricted to the consecutive artifact tuples involved in the transition. This information is sufficient to determine which service can be applied at each point in the run. Since there are only finitely many such isomorphism types, this will allow us to construct a finite-state transition system defining precisely the service view of Γ. As we shall see, it is straightforward that each sequence of services occurring in an actual run of Γ is also allowed by the transition system. The main technical difficulty is showing the converse: that every sequence of services allowed by the transition system corresponds to a sequence of services in an actual run of Γ.

We next define the notion of isomorphism type and corresponding transition system. 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 (i.e., for each $R \in \mathcal{DB}$ of arity k and $\alpha_i, \beta_i \in \bar{y} \cup C$ such that $(\alpha_i, \beta_i) \in \epsilon$ for $i \leq i \leq k$, $(\alpha_1, \ldots \alpha_k) \in H(R)$ iff $(\beta_1, \ldots \beta_k) \in H(R)$).

Definition 10. 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 $\bar{x}_0 \cup C$,

(iii) (H_i, ϵ_i) and $(H_{i+1}, \epsilon_{i+1})$ agree on $\bar{x}_{i+1} \cup C$,

(iv) the pre-condition of σ_i holds in the restriction of (H_i, ϵ_i) to $\bar{x}_i \cup C$, and

(v) the post-condition of σ_i holds in (H_i, ϵ_i).

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

If a symbolic run faithfully represents an actual run of Γ, we say that the actual run is an *enactment* of the symbolic run. Intuitively, the actual run is obtained by giving concrete values to the variables in the symbolic run. In addition, the actual run is equipped with a finite database consistent with the isomorphism types in the symbolic run (note that the symbolic run has no database). We formalize this next.

Let ϱ be a symbolic run of Γ and $\approx$ 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 11. 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 $D|(\rho_i \cup \rho_{i+1})$ (where H_i/ϵ_i is the quotient structure of H_i with respect to ϵ_i).

Lemma 6. *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 symbolic run ϱ can be easily constructed by induction from ρ. □

Consider the converse of Lemma 6: 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 7. *Let Γ be an artifact system with artifact variables $\bar{x}$. 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 $|\bar{x}|$.*

The proof is non-trivial and requires developing some technical machinery, which we do next. 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 $x \in \bar{x}$ (with the exception of equalities with constants). Moreover, each relational atom occurring in conditions uses only variables from $\bar{x}$ or only variables from $\bar{x}'$. Intuitively, this limits the interaction between consecutive instances to the propagation of a subset of the variables in the transition. We note that this is similar in spirit to the treatment of equality in the normal form for Knowledge and Action Bases developed in [25], and to the singularization technique of [33].

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 (the more difficult part of the proof).

We begin with the definition of the linear-propagation normal form.

Definition 12. An artifact system $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ with $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ is in *linear propagation* form if the following hold for each service (π, ψ) in Σ:

(i) $\pi(\bar{x})$ uses only relational symbols from $\mathcal{DB}$ and equalities of the form $x = c$ (where c is a constant from **dom**).

(ii) $\psi(\bar{x}, \bar{x}')$ is of the form $\psi_1(\bar{x}) \wedge \psi_2(\bar{x}') \wedge \alpha(\bar{x}, \bar{x}')$ where $\psi_1(\bar{x})$ and $\psi_2(\bar{x}')$ use only relational symbols from $\mathcal{DB}$ and equalities of the form $x = c$ (resp. $x' = c$) where c is a constant, and $\alpha(\bar{x}, \bar{x}') = \wedge_{i \in I}(x_i = x'_i)$ for some $I \subseteq [1...k]$.

Example 8. *The artifact system in Example 2 is not in linear-propagation form. Indeed, the services* **payment_refused** *and* **payment_ok** *violate condition (i), and services* **choose_ship, no_coupon, apply_coupon** *violate condition (ii).*

Definition 13. Let Γ_1, Γ_2 be artifact systems. We say that Γ_2 *symbolically simulates* Γ_1 if there is a mapping h from $SRuns(\Gamma_1)$ to $SRuns(\Gamma_2)$ such that:

- for each symbolic run ϱ_1 of Γ_1, if there exists an enactment (D, ρ_2, f_2) of $h(\varrho_1)$ then there exists an enactment (D, ρ_1, f_1) of ϱ_1 (with the *same* database).

Intuitively, symbolic simulation is akin to a reduction. Its purpose is to allow "translating" enactments of one artifact system into enactments of another. Suppose Γ_2 symbolically simulates Γ_1. If we can show that every symbolic run of Γ_2 has an enactment, then the definition of symbolic simulation implies that every symbolic run of Γ_1 also has an enactment. Note that this does *not* imply that Γ_2 has the same service view as Γ_1, or that Γ_2 can replace Γ_1 for the purpose of model checking.

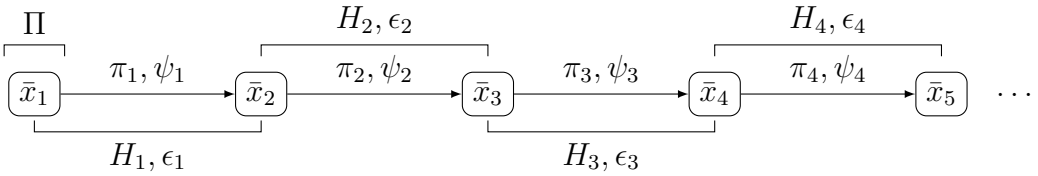
Observe that symbolic simulation is transitive.

Lemma 9. *Let* $\Gamma_1 = \langle\langle \bar{x}, \mathcal{DB}_1\rangle, \Sigma_1, \Pi_1\rangle$ *be an artifact system. There exists an artifact system* $\Gamma_2 = \langle\langle \bar{z}, \mathcal{DB}_2\rangle, \Sigma_2, \Pi_2\rangle$ *in linear propagation form that symbolically simulates* Γ_1.

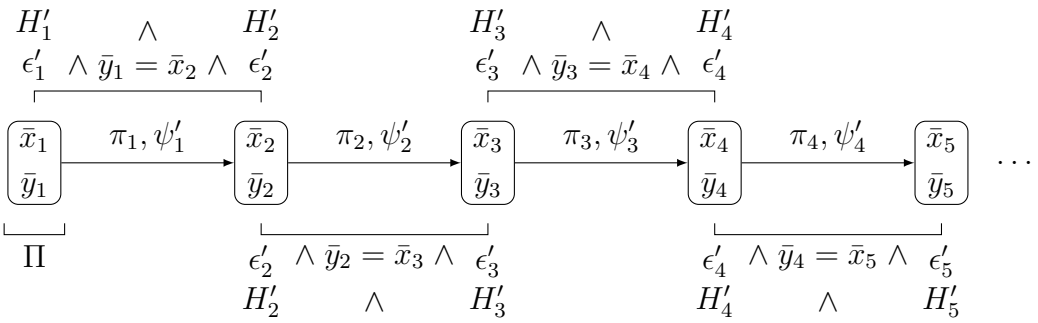
PROOF. The construction is done in several stages.

1. We first eliminate from pre-and-post conditions of services of Γ_1 relational atoms $R(\bar{z})$ where $\bar{z}$ contains variables from both $\bar{x}$ and $\bar{x}'$. This is done by adding new artifact variables as follows. For each $x \in \bar{x}$, let $y(x)$ be a distinct new variable. The artifact system Γ_2 has artifact variables $\bar{x} \cup y(\bar{x})$.

Pre-conditions of services of Γ_1 remain unchanged, and each post-condition $\psi(\bar{x}, \bar{x}')$ is rewritten as $\psi(\bar{x}, y(\bar{x})) \wedge \bigwedge_{x \in \bar{x}} y(x) = x'$. Clearly, every relational atom in Γ_2 uses only variables in $\bar{x} \cup y(\bar{x})$. The symbolic simulation h_1 of Γ_1 by Γ_2 is the mapping from symbolic runs of Γ_1 to those of Γ_2 induced in the natual manner by the above modification. We illustrate this next. The following represents a symbolic run $\varrho_1 = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$ of Γ_1. Recall that the H_i may contain relational atoms with variables from both $\bar{x}_i$ and $\bar{x}_{i+1}$.



For each $i \geq 0$ and $x \in \bar{x}$, let $y_i(x)$ be a new variable denoting the occurrence of $y(x)$ in the i-th configuration of the symbolic runs of Γ_2. Let $\bar{y}_i$ denote $y_i(\bar{x})$, $i \geq 0$. The symbolic run $\varrho_2 = h_1(\varrho_1)$ of Γ_2 is $\{((\bar{x}_i, \bar{y}_i), H_i' \wedge H_{i+1}', \epsilon_i' \wedge \epsilon_{i+1}' \wedge (\bar{x}_{i+1} = \bar{y}_i), \sigma_i')\}_{i \geq 0}$ represented below. In $h_1(\varrho_1)$, $H_i' = H_i[\bar{y}_i/\bar{x}_{i+1}]$, $\epsilon_i' = \epsilon_i[\bar{y}_i/\bar{x}_{i+1}]$ and $\psi_i' = \psi_i[\bar{y}_i/\bar{x}_{i+1}] \wedge (\bar{x}_{i+1} = \bar{y}_i)$.



Consider an enactment (D, ρ_2, θ_2) of Γ_2. An enactment (D, ρ_1, θ_1) of Γ_1 is easily constructed by redirecting θ_2 from $\bar{y}_i$ to $\bar{x}_{i+1}$ in the obvious way. Thus, h_1 is a symbolical simulation of Γ_1 by Γ_2.

2. Let Γ_2 be the specification resulting from step (1). Let $\bar{u} = \bar{x} \cup y(\bar{x})$, so that $\Gamma_2 = (\langle \bar{u}, \mathcal{DB} \rangle, \Sigma_2, \Pi_2)$. In the second stage, we eliminate from pre- and-post-conditions of services in Σ_2, and from Π_2, all equalities of the form $u = v$ and $u' = v'$ for $u, v \in \bar{u}, u \neq v$ (called *local equalities*). Intuitively,

rather than stating these equalities explicitly, we keep track of them using one additional artifact variable p and use in relational atoms just one representative variable from each equality class. More precisely, let $\mathcal{E}$ and $\mathcal{E}'$ be the sets of partitions (equality types) of $\bar{u}$ and $\bar{u}'$. We assume an arbitrary ordering on the variables in $\bar{u}$. For each $u \in \bar{u}$ and $\epsilon \in \mathcal{E}$, let u_ϵ be the smallest variable in $\bar{u}$ in the same class of ϵ as u (and similarly $u'_{\epsilon'}$ for $u' \in \bar{u}'$ and $\epsilon' \in \mathcal{E}'$). For a formula ψ over $\bar{u} \cup \bar{u}'$, and $\epsilon \in \mathcal{E}, \epsilon' \in \mathcal{E}'$, let $\psi_{\epsilon \cup \epsilon'}$ be the formula obtained by eliminating all local equalities and replacing each $u \in \bar{u}$ with u_ϵ and each $u' \in \bar{u}'$ by $u'_{\epsilon'}$. For a quantifier-free formula ψ over $\bar{u} \cup \bar{u}'$, let $\delta_\psi = \{(\epsilon, \epsilon') \mid \epsilon \in \mathcal{E}, \epsilon' \in \mathcal{E}', \epsilon \cup \epsilon' \text{ is compatible with } \psi\}$. Similarly, if ψ is over $\bar{u}$ alone, $\delta^0_\psi = \{\epsilon \mid \epsilon \in \mathcal{E}, \epsilon \text{ is compatible with } \psi\}$. For each $\epsilon \in \mathcal{E}$, we denote by φ_ϵ the conjunction of (in)equalities specifying ϵ (and similarly for $\epsilon' \in \mathcal{E}'$).

We are now ready to define $h_2(\Gamma_2)$. Let $\Gamma_3 = (\langle(\bar{u}, p), \mathcal{DB}\rangle, \Sigma_3, \Pi_3)$ where:

- $\Pi_3(\bar{u}) = \bigvee_{\epsilon \in \delta^0_{\Pi_2}} (p = \epsilon \wedge (\Pi_2)_\epsilon)$
- Σ_3 is obtained by rewriting each service (π, ψ) of Γ_2 to (π', ψ'), where $\pi' = \bigvee_{\epsilon \in \delta^0_\pi} (p = \epsilon \wedge \pi_\epsilon)$ and $\psi' = \bigvee_{(\epsilon, \epsilon') \in \delta_\psi} (p = \epsilon \wedge p' = \epsilon' \wedge \psi_{\epsilon \cup \epsilon'})$.

The simulation h_2 from symbolic runs of Γ_2 to those of Γ_3 is induced in the natural way from the above construction.

3. We finally eliminate equalities of the form $u_\epsilon = v'_{\epsilon'}$ with $u, v \in \bar{u}, (u_\epsilon)' \neq v'_{\epsilon'}$. Note that the variable p in Γ_3 recording equality types does not occur in such atoms. Intuitively, we ensure that $(u_\epsilon)' = v'_{\epsilon'}$ by permuting variables appropriately. To do this, we must keep track of the current permutation in an additional artifact variable, and modify service post-conditions according to the current permutation. This induces again a simulation from symbolic runs of Γ_3 to symbolic runs of the new specification. We omit the details.

The desired artifact system and final simulation is obtained by putting together the specifications and simulations of (1-3). □

Example 10. *We illustrate some aspects of the above construction for the following services of Example 2:*

choose_ship *The customer chooses a compatible shipping option.*

π : status = *"edit_ship"*
ψ : $\exists x$(SHIPPING(ship_type$'$, prod_id, x) $\wedge$ status$'$ = *"edit_coupon"*
$\wedge$prod_id$'$ = prod_id)

payment_ok *Check payment and ship the product.*

π : status = *"received_payment"* $\wedge$ amount_paid = amount_owed
ψ :status$'$ = *"shipping"* $\wedge$ amount_paid$'$ = amount_paid$\wedge$
prod_id$'$ = prod_id $\wedge$ ship_type$'$ = ship_type$\wedge$
coupon$'$ = coupon $\wedge$ amount_owed$'$ = amount_owed

As noted earlier, **payment_ok** *violates condition* (i) *of linear-propagation form because of the equality* amount_paid = amount_owed. *Also,* **choose_ship** *violates condition* (ii) *because of the use of* ship_type$'$, prod_id *in the atom* SHIPPING(ship_type$'$, prod_id, x) *of* ψ. *We first eliminate the use of* ship_type$'$ *in* SHIPPING(ship_type$'$, prod_id, p) *by introducing a new artifact variable* ship_type$_0$ *and rewriting* ψ *as* $\exists x$(SHIPPING(ship_type$_0$, prod_id, x)$\wedge$
ship_type$_0$ = ship_type$'_0$ $\wedge$ ship_type$'_0$ = ship_type$'$ $\wedge$ status$'$ = *"edit_coupon"* $\wedge$prod_id$'$ = prod_id). *Then we eliminate the explicit equality* ship_type$'_0$ = ship_type$'$ *as well as* amount_paid = amount_owed *from the pre-condition of* **payment_ok** *by introducing a new variable* p *recording the truth values of the two equalities (represented by special constants). Equality tests* amount_paid = amount_owed *and* ship_type$'_0$ = ship_type$'$ *are then replaced by a disjunction of equality tests of the form* $p = c$, *respectively* $p' = c$, *where the* c*'s are constants designating the compatible truth assignments. If no condition is imposed on* p *in a transition, then one of the constants is assigned to it nondeterministically.*

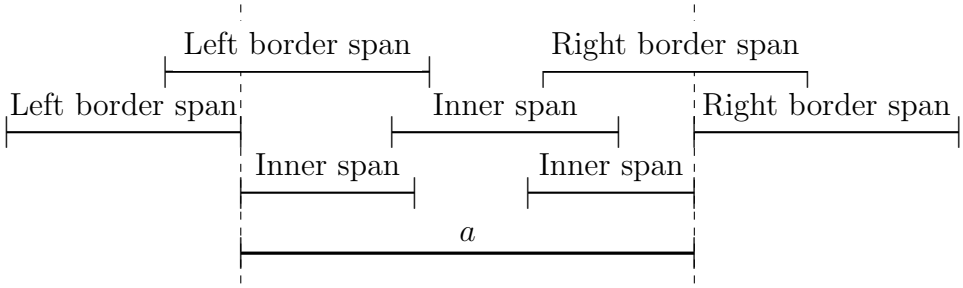
We now continue with the proof of Theorem 7. Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ with $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ be an artifact system. By Lemma 9, we can assume that Γ is in linear propagation form. Recall that for a symbolic run $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$ 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 wihout loss of generality 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 $i \leq j$).

Intuitively, the proof of Theorem 7 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. The characteristics are rather involved, because the lanes of all artifact variables must be taken into account simultaneously in order to obtain a consistent collapse. This yields an enactment of the symbolic run over the finite database whose elements are the collapsed classes. To achieve this, we need several definitions and technical lemmas.

Definition 14. Let $x, y \in \bar{x}, x \neq y$. Let $a \in lane(x), b \in lane(y)$, $span(a) = [i, j]$ and $span(b) = [i', j']$.

- b is a *left-border* span of a if $i' < i \leq j' < j$.
- b is a *right-border* span of a if $i < i' \leq j < j'$.
- b is a *border span* of a if it is a left-border or a right-border span of a.
- b is an *inner span* of a if $i \leq i' \leq j' \leq j$.

Remark 11. Left-border spans, right-border spans and inner spans of a are illustrated below.



We next define the *alternation number* of one lane with respect to another. This rather technical notion is key in capturing the relationship between the lanes of different artifact variables, controlling the simultaneous collapse of equivalence classes of different lanes so that overall consistency is ensured.

Definition 15. Let $x, y \in \bar{x}$ be two artifact variables. Let $(\tilde{x}_i)_{i \in \mathbb{N}}$ be the (ordered) values of $lane(x)$, $(\tilde{y}_i)_{i \in \mathbb{N}}$ be the (ordered) values of $lane(y)$. We define the *alternation number* $AN_{x,y}$ of $lane(x)$ with respect to $lane(y)$ by induction:

- $AN_{x,y}(\tilde{x}_0) = 0$
- if there exists j such that $\tilde{y}_j$ is a border span of $\tilde{x}_i$ then

$$AN_{x,y}(\tilde{x}_{i+1}) = AN_{x,y}(\tilde{x}_i) + 1 \; mod \; (4)$$

- otherwise $AN_{x,y}(\tilde{x}_{i+1}) = AN_{x,y}(\tilde{x}_i)$

We will need some basic properties of spans and alternation numbers.

Proposition 12. *Let $x, y \in \bar{x}$ be distincts artifact variables, $\tilde{x} \in lane(x)$ and $\tilde{y}, \tilde{y}' \in lane(y)$.*

- *If $\tilde{y}$ is a left-border span of $\tilde{x}$, then $\tilde{x}$ is a right-border span of $\tilde{y}$.*
- *$\tilde{x}$ has at most one left border span and one right border span on a given lane.*
- *If $\tilde{y}$ is a left-border span of $\tilde{x}$ and $\tilde{y}'$ a right-border or inner span of $\tilde{x}$ then $AN_{y,x}(\tilde{y}) \neq AN_{y,x}(\tilde{y}')$.*

Proof. The first two properties are trivial. Consider the third. We define the *unbounded alternation number* $AN^u_{x,y}$ of $lane(x)$ with respect to $lane(y)$. This is defined similarly to the alternation number, only without the modulo operation:

- $AN^u_{x,y}(\tilde{x}_0) = 0$
- if there exists j such that $\tilde{y}_j$ is a border span of $\tilde{x}_i$ then

$$AN^u_{x,y}(\tilde{x}_{i+1}) = AN^u_{x,y}(\tilde{x}_i) + 1$$

- otherwise $AN^u_{x,y}(\tilde{x}_{i+1}) = AN^u_{x,y}(\tilde{x}_i)$

We note that $\{AN^u_{x,y}(\tilde{x}_i)\}_{i\geq 0}$ is a non-decreasing sequence, and that for all $i \geq 0$, $AN_{x,y}(\tilde{x}_i) = AN^u_{x,y}(\tilde{x}_i) \; mod \; (4)$.

Let $x, y \in \bar{x}$ be two distinct artifact variables, $\tilde{x} \in lane(x)$ and $\tilde{y}, \tilde{y}' \in lane(y)$.

In order to prove the last property, we will show that if $\tilde{y}$ is a left-border span of $\tilde{x}$ and $\tilde{y}'$ a right-border or inner span of $\tilde{x}$ then

$$0 < AN^u_{y,x}(\tilde{y}') - AN^u_{y,x}(\tilde{y}) < 4$$

This would imply that $AN_{y,x}(\tilde{y}) \neq AN_{y,x}(\tilde{y}')$.

Let $(\tilde{y}_i)_{i\in\mathbb{N}}$ be the (ordered) value of the y lane. Assume that there exists i, j integers such that $\tilde{y}_i$ is the left border span of $\tilde{x}$ and $\tilde{y}_j$ is the right border span of $\tilde{x}$.

The cases where $\tilde{y}_{i+1}$, $\tilde{y}_{i+2}$ or $\tilde{y}_{i+3}$ is a right border span of $\tilde{x}$ are easy because then we have $j - i < 4$, which implied the result trivialy (since the difference between two consecutive values of $AN^u_{y,x}$ is bounded by 1). Hence assume $j - i \geq 4$. There are two possible configurations for the left part of $\tilde{x}$.

Case 1 **Case 2**

In case 1, $\tilde{y}_i$ has a border span on the x lane but $\tilde{y}_{i+1}$ has not. It follows that

$$AN^u_{y,x}(\tilde{y}_{i+2}) = AN^u_{y,x}(\tilde{y}_i) + 1$$

In case 2, $\tilde{y}_i$ has a border span on the x lane, and so has $\tilde{y}_{i+1}$. Consequently

$$AN^u_{y,x}(\tilde{y}_{i+2}) = AN^u_{y,x}(\tilde{y}_i) + 2$$

Thus we obtain our first inequality

$$0 < AN^u_{y,x}(\tilde{y}_{i+2}) - AN^u_{y,x}(\tilde{y}_i) < 3$$

Now there are two different configuration for the right part of x.

Case 1 **Case 2**

For all integers l such that $i + 2 < l < j - 1$, $\tilde{y}_l$ has no border span on the x lane. Hence we have

$$AN^u_{y,x}(\tilde{y}_{i+2}) = AN^u_{y,x}(\tilde{y}_l) = AN^u_{y,x}(\tilde{y}_{j-1})$$

In the case 2, $\tilde{y}_{j-1}$ has a border span on the x lane, but not in the case 1. So

$$0 \leq AN^u_{y,x}(\tilde{y}_j) - AN^u_{y,x}(\tilde{y}_{j-1}) \leq 1$$

By combining the previous inequalities we obtain

$$\forall i < l \leq j,\ 0 < AN^u_{y,x}(\tilde{y}_l) - AN^u_{y,x}(\tilde{y}_i) < 4$$

This concludes the proof of the lemma. □

We are now ready to define the equivalence relation among elements of a lane that allows the finite collapse.

Definition 16. Let $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i\geq 0}$ be a symbolic run. Let $y \in \bar{x}$ and $(\tilde{y}_i)_{i\in\mathbb{N}}$ be the (ordered) elements of the y lane. We define $\tilde{y}_i \sim \tilde{y}_j$ if the following hold:

- the relational portions of the isomorphism types of $\tilde{y}_i$ and $\tilde{y}_j$ are identical at the beginning of their span (up to renaming). More precisely, if $i_1 = min(span(\tilde{y}_i))$ and $i_2 = min(span(\tilde{y}_j))$, then

$$(H_{i_1})_{|\bar{x}_{i_1}} = \left((H_{i_2})\right)_{|\bar{x}_{i_2}} [\bar{x}_{i_1}/\bar{x}_{i_2}]$$

- i and j have the same parity
- for every $z \in \bar{x}$ and $y \neq z$, $AN_{y,z}(\tilde{y}_i) = AN_{y,z}(\tilde{y}_j)$
- for all a, b such that $\{a, b\} = \{i, j\}$:

  $\forall z \in \bar{x}, \forall \tilde{z} \in lane(z)$ if $\tilde{z}$ is a left-border span of $\tilde{y}_a$ then there exists $\tilde{z}' \in lane(z)$ such that $\tilde{z}'$ is a left-border span of $\tilde{y}_b$ and $AN_{z,y}(\tilde{z}) = AN_{z,y}(\tilde{z}')$

- for all a, b such that $\{a, b\} = \{i, j\}$:

  $\forall z \in \bar{x}, \forall \tilde{z} \in lane(z)$ if $\tilde{z}$ is an inner span of $\tilde{y}_a$ then there exists $\tilde{z}' \in lane(z)$ such that $\tilde{z}'$ is an inner span of $\tilde{y}_b$ and $AN_{z,y}(\tilde{z}) = AN_{z,y}(\tilde{z}')$

Finally, for $\tilde{y}$ and $\tilde{z}$ on different lanes, $\tilde{y} \not\sim \tilde{z}$.

The following is immediate.

Lemma 13.

- $\sim$ *is an equivalence relation.*
- $\sim$ *is of finite index (exponential in* $|x|$*).*

Let $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i\geq 0}$ be a symbolic run, R a relation of $\mathcal{DB}$ of arity l and $y_1, \ldots, y_l$ variables in $\bar{x}$. For all $1 \leq i \leq l$, let $\tilde{y}_i \in lane(y_i)$. We say that $R(\tilde{y}_1, \ldots, \tilde{y}_l)$ holds in ϱ (denoted by $\varrho \models R(\tilde{y}_1, \ldots, \tilde{y}_l)$) if there exists $j \in \bigcap_{1\leq i\leq l} span(\tilde{y}_i)$ such that $H_j \models R(\tilde{y}_1, \ldots, \tilde{y}_l)$.

Note the following: if for all $1 \leq i \leq l$ we have $\tilde{y}_i \approx \bar{y}_i$, then $\varrho \models R(\tilde{y}_1, \ldots, \tilde{y}_l)$ iff $\varrho \models R(\bar{y}_1, \ldots, \bar{y}_l)$ (this follows easily from the definition of symbolic run).

We can now show the following key lemma.

Lemma 14. *Let $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i\geq 0}$ be a symbolic run and $R \in \mathcal{DB}$ be a relation of arity l. For each $j, 1 \leq j \leq l$, let $\tilde{x}_j$ and $\tilde{x}'_j$ be such that $\tilde{x}_j \sim \tilde{x}'_j$. Furthermore assume that $\bigcap_{1\leq j\leq l} span(\tilde{x}_j) \neq \emptyset$ and $\bigcap_{1\leq j\leq l} span(\tilde{x}'_j) \neq \emptyset$. Then we have:*

$$\varrho \models R(\tilde{x}_1, \ldots, \tilde{x}_l) \text{ iff } \varrho \models R(\tilde{x}'_1, \ldots, \tilde{x}'_l)$$

Proof. Suppose $\{\tilde{x}_j\}_{1\leq i\leq l}, \{\tilde{x}'_j\}_{1\leq i\leq l}$ satisfy the hypothesis of the lemma.

Let $J = \{j \mid \forall 1 \leq i \leq l, min(span(\tilde{x}_i)) \leq min(span(\tilde{x}_j))\}$. Let $j_{max} \in J$ be such that for all $j \in J$, $min(span(\tilde{x}'_j)) \leq min(span(\tilde{x}'_{j_{max}}))$. Let z be such that $\tilde{x}_{j_{max}} \in lane(z)$.

We will show that

$$(\dagger)\ \ \forall 1 \leq j \leq l, min(span(\tilde{x}'_j)) \leq min(span(\tilde{x}'_{j_{max}}))$$

Assume towards a contradiction that this is false, and let j_{abs} be such that $\forall 1 \leq j \leq l, min(span(\tilde{x}'_j)) \leq min(span(\tilde{x}'_{j_{abs}}))$ and $min(span(\tilde{x}'_{j_{max}})) < min(span(\tilde{x}'_{j_{abs}}))$. Let y be such that $\tilde{x}_{j_{abs}} \in lane(y)$. Observe the following:

(i) Since $\bigcap_{1\leq j\leq l} span(\tilde{x}'_j) \neq \emptyset$ and $\forall 1 \leq j \leq l, min(span(\tilde{x}'_j)) \leq min(span(\tilde{x}'_{j_{abs}}))$ exactly one of the following holds:

- $\tilde{x}'_{j_{max}}$ is an inner span of $\tilde{x}'_{j_{abs}}$
- $\tilde{x}'_{j_{abs}}$ is an inner span of $\tilde{x}'_{j_{max}}$
- $\tilde{x}'_{j_{max}}$ is a left-border span of $\tilde{x}'_{j_{abs}}$

(ii) Since $\bigcap_{1\leq j\leq l} span(\tilde{x}_j) \neq \emptyset$ and $\forall 1 \leq j \leq l, min(span(\tilde{x}_j)) \leq min(span(\tilde{x}_{j_{max}}))$ exactly one of the following is true:

– $\tilde{x}_{j_{abs}}$ is an inner span of $\tilde{x}_{j_{max}}$
– $\tilde{x}_{j_{max}}$ is an inner span of $\tilde{x}_{j_{abs}}$
– $\tilde{x}_{j_{abs}}$ is a left-border span of $\tilde{x}_{j_{max}}$

Suppose first that $\tilde{x}'_{j_{max}}$ is an inner span of $\tilde{x}'_{j_{abs}}$. This implies that $min(span(\tilde{x}'_{j_{abs}})) \leq min(span(\tilde{x}'_{j_{max}}))$, which contradicts the fact that

$$min(span(\tilde{x}'_{j_{max}})) < min(span(\tilde{x}'_{j_{abs}})).$$

Now assume that $\tilde{x}'_{j_{abs}}$ is an inner span of $\tilde{x}'_{j_{max}}$. There are three cases:

- if $\tilde{x}_{j_{abs}}$ is an inner span of $\tilde{x}_{j_{max}}$ then $min(span(\tilde{x}_{j_{max}})) = min(span(\tilde{x}_{j_{abs}}))$ and so $j_{abs} \in J$. By definition of j_{max} we know that $min(span(\tilde{x}'_{j_{abs}})) \leq min(span(\tilde{x}'_{j_{max}}))$, contradicting the fact that

$$min(span(\tilde{x}'_{j_{max}})) < min(span(\tilde{x}'_{j_{abs}})).$$

- if $\tilde{x}_{j_{max}}$ is an inner span of $\tilde{x}_{j_{abs}}$; since $\tilde{x}_{j_{abs}} \sim \tilde{x}'_{j_{abs}}$, there is some $\tilde{z}'$ on $lane(y)$ such that $\tilde{z}'$ is an inner span of $\tilde{x}'_{j_{abs}}$. This implies that $span(\tilde{x}'_{j_{abs}}) = span(\tilde{x}'_{j_{max}})$. This contradicts the fact that

$$min(span(\tilde{x}'_{j_{max}})) < min(span(\tilde{x}'_{j_{abs}})).$$

- if $\tilde{x}_{j_{abs}}$ is a left-border span of $\tilde{x}_{j_{max}}$ then since $\tilde{x}_{j_{max}} \sim \tilde{x}'_{j_{max}}$, there is some $\tilde{z}'$ on $lane(y)$ such that $\tilde{z}'$ is a left-border span of $\tilde{x}'_{j_{max}}$ and $AN_{y,z}(\tilde{x}_{j_{abs}}) = AN_{y,z}(\tilde{z}')$. Since $\tilde{x}'_{j_{abs}}$ is an inner span of $\tilde{x}'_{j_{max}}$, we know by Proposition 12 that $AN_{y,z}(\tilde{x}'_{j_{abs}}) \neq AN_{y,z}(\tilde{z}')$. This contradicts the fact that $AN_{y,z}(\tilde{x}'_{j_{abs}}) = AN_{y,z}(\tilde{x}_{j_{abs}})$ (because $\tilde{x}'_{j_{abs}} \sim \tilde{x}_{j_{abs}}$).

Finally, the case when $\tilde{x}'_{j_{max}}$ is a left-border span of $\tilde{x}'_{j_{abs}}$ is similar to the above (omitted).

This concludes the proof of (†). Next, for all $1 \leq i \leq l$, let z_i be the lane of $\tilde{x}_i$ (and of $\tilde{x}'_i$). Let $m = min(span(\tilde{x}_{j_{max}}))$ and $m' = min(span(\tilde{x}'_{j_{max}}))$. By definition of j_{max}, $\forall 1 \leq j \leq l, min(span(\tilde{x}_j)) \leq min(span(\tilde{x}_{j_{max}}))$. Thus for all $1 \leq i \leq l$, $m \in span(\tilde{x}_i)$. Hence

$$\varrho \models R(\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_l) \text{ iff } H_m \models R((z_1)_m, (z_2)_m, ..., (z_l)_m)$$

Additionally, (†) implies that

$$\varrho \models R(\tilde{x}'_1, \tilde{x}'_2, ..., \tilde{x}'_l) \text{ iff } H_{m'} \models R((z_1)_{m'}, (z_2)_{m'}, ..., (z_l)_{m'})$$

Moreover, since $\tilde{x}_{j_{max}} \sim \tilde{x}'_{j_{max}}$ we know that $(H_m)_{|\bar{x}_m}$ coincides with $(H_{m'})_{|\bar{x}_{m'}}[\bar{x}_m/\bar{x}_{m'}]$ on $\bar{x}_m$, so

$$\varrho \models R(\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_l) \text{ iff } \varrho \models R(\tilde{x}'_1, \tilde{x}'_2, ..., \tilde{x}'_l)$$

This concludes the proof of Lemma 14. □

Lemma 14 allows us to define an enactment (D, ρ, θ) for every symbolic run $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i \geq 0}$ of an artifact system Γ in linear propagation form. The domain of D consists of the equivalence classes of $\sim$ (of which there are finitely many), and for each $R \in \mathcal{DB}$ of arity l, the tuples it contains are defined as above. The mapping θ sends x_i to the equivalence class of $[x_i]_{\approx}$ under $\sim$, and ρ is the image of ϱ under θ. It is easy to check that (D, ρ, θ) so defined is an enactment of ϱ. Finally, Lemma 9 allows to transfer enactments of symbolic runs of artifact systems in linear propagation form to enactments of symbolic runs of arbitrary artifact systems. Observe that the final enactment is exponential in $|\bar{x}|$. This completes the proof of Theorem 7.

Effective regularity of the service view We can now show the effective regularity of the service view of Γ. From Lemma 6 and Theorem 7 it follows that $\mathcal{S}_{lin}(\Gamma) = \{\mathcal{S}(\varrho) \mid \varrho \in \mathit{SRuns}(\Gamma)\}$. 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 10 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 $\{\mathcal{S}(\varrho) \mid \varrho \in \mathit{SRuns}(\Gamma)\}$. Thus, we have the following.

Theorem 15. *The class of unconstrained artifact systems has effectively ω-regular service views.*

Verification vs. effective ω-regularity We note that the effective ω-regularity of $\mathcal{S}_{lin}(\Gamma)$ implies decidability of LTL(QFO) properties of artifact systems, but is strictly stronger. Decidability of verification follows from ω-regularity because for each Γ and LTL(QFO) 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 [39] which imply that artifact systems equipped with a total order do not have ω-regular service views, yet verification of LTL(QFO) properties is decidable.

Universal test databases The above results, notably Theorem 7, 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 k such that for every constant-free LTL(QFO) formula ξ over Γ, $\Gamma \models \xi$ iff $Runs_{D^*}(\Gamma) \models \xi$.*

Proof. Consider an LTL(QFO) formula ξ over Γ. As shown in [12] (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 6 and Theorem 7 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. $\square$

Thus, D^* acts as a universal test database (akin to an Armstrong relation) for satisfaction of constant-free LTL(QFO) 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(QFO) property violated by Γ. In contrast, the counter-example databases constructed by the algorithms in [18, 12] 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.

Theorem 16 also provides a potentially fruitful bridge to a line of research that considers the verification of data-centric systems for a *fixed* initial database (e.g., see [14, 23, 24], and the survey [8]). It shows that, for artifact systems and properties satisfying the conditions of the theorem, verification of runs over arbitrary databases can be reduced to verification of runs over the fixed universal test database D^*.

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 contents 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). Figure 2 shows a finite-state transition system representing the branching-time service view of the artifact system in Example 2.

Analogously to the linear case, we say that a class $\mathbf{A}$ of artifact systems has effectively regular branching-time service views if there is an algorithm that, for each Γ in $\mathbf{A}$, produces a finite-state transition system defining $\mathcal{TS}^*(\Gamma)$.

As we shall see, it turns out that unlike linear-time service views, branching-time service views of unconstrained artifact systems are generally *not* regular. One might hope that effective regularity holds for artifacts obeying the natural feedback-free property that has proven so effective in overcoming undecidability of LTL(QFO) properties for specifications with dependencies and arithmetic [12]. Unfortunately, this is not the case. Indeed, we show that

even very simple CTL(QFO) properties are not decidable for feedback-free artifacts, thus implying that they do not have effectively regular branching-time service views.

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

Proof. The proof that CTL(QFO) model checking of feedback-free artifact systems is undecidable is by reduction from the Post Correspondence Problem (PCP) [38]. This is done in several stages. First, we sketch a proof of the result of [18] (Theorem 4.2, stated there without proof) that checking LTL(QFO) 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(QFO) property. Using this, we wish to mimick the reduction from the PCP that works for databases with FDs. However, there is a glitch: LTL(QFO) 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.

We begin by recalling the following result from [18] and sketch a proof that we will next adapt to the branching-time setting.

Lemma 18. *[18] It is undecidable, given a set* Δ *of FDs, an artifact system* Γ*, and an LTL(QFO) formula* ξ*, whether* $\Gamma(\Delta) \models \xi$.

Proof. Consider a PCP instance consisting of two sequences $(u_1, u_2, ..., u_k)$, $(v_1, v_2, ..., v_k)$ of words over the alphabet $\{0, 1\}$. We sketch the construction of an artifact system Γ whose database satisfies one FD, and an LTL(QFO) formula ξ such that $\Gamma \models \xi$ iff there is no solution to the PCP instance.

The database $\mathcal{DB}$ consists of a unary relation S and a binary relation R satisfying two FDs $1 \to 2$ and $2 \to 1$. Hence R can be viewed as a directed graph in which every node has out-degree 1. Thus, every simple path in R can be uniquely identified by its initial vertex. Moreover, each vertex x can be labelled by a 1 (if $S(x)$ holds) or a 0 (if $S(x)$ does not hold). This way, we can associate a word over $\{0, 1\}$ to each path in R.

The artifact system Γ implements a walk along a simple path in R, starting from some nondeterministically chosen node x_0, and checks if the word w

spelled by the labels along the path is a solution to the PCP. To do so, Γ non-deterministically picks a sequence $i_1, \ldots i_n$ of indexes from $[1, ..., k]$, and stores the positions in w reached by $u_{i_1} \ldots u_{i_n}$, resp. $v_{i_1} \ldots v_{i_n}$, in two variables x_u and x_v. Initially, $x_u = x_v = x_0$. Upon chosing a new index $i_j = m$, Γ tries to match the corresponding words u_m and v_m in parallel against w, starting from the current x_u, resp. x_v, using additional variables as cursors on u_m and v_m. The variables x_u and the cursor for u_m advance in lock-step, being incremented only if they point to the same character, and similarly for x_v and v_m. A solution to the PCP is found if $x_u = x_v \wedge x_u \neq x_0$. The LTL(QFO) property ξ is then $\mathbf{G}(\neg(x_u = x_v \wedge x_u \neq x_0))$. $\square$

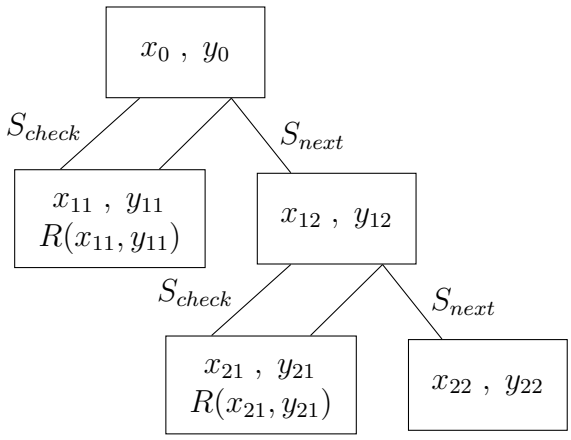
We now proceed with the proof of Theorem 17, by adapting the previous reduction from the PCP.

First, we force the relation R to satisfy the FDs $1 \rightarrow 2$ and $2 \rightarrow 1$. Let S_{key} be the service whose pre-condition is $\exists x, y, z(R(x, y) \wedge R(x, z) \wedge y \neq z)$. We see that this service can not be applied if and only if the relation R satisfy a key dependency on its first attribute. Hence by adding to the CTL(QFO) formula the atom $\neg S_{key}$ we can force R to satisfy this key dependency.

We would like to mimick the reduction for the linear case by following a simple path in R while checking if a solution to the PCP is found. Note first that a direct application of the previous construction does not work, because walking through consecutive nodes in R creates a violation of feedback freedom.

We show how to circumvent feedback freedom by using different branches in the tree of runs, to make the necessary connections. Suppose x contains a node, and we wish that x' contain its successor. Requiring directly $R(x, x')$ would violate feedback freedom. Instead, we do this indirectly, using an additional variable y. Clearly, $R(x, x')$ is equivalent to $R(x, y) \wedge (x' = y)$. However, using this formula still leads to violation of feedback freedom. Instead, we separate this condition in two parts, tested in different branches using two services: S_{check} with post-condition $R(x, y)$, and S_{next} with post-condition $x' = y$.

The following represents a fragment of the symbolic tree of runs that achieves this (edges indicate equalities among variables).



The x variables on the right branch of the tree (i.e. x_0, x_{12}, x_{22}) are consecutive nodes on a path in R, yet each branch in the tree is feedback free. Using this as a building block, the construction used in the linear case is easily simulated. The CTL(QFO) formula stating the existence of a solution to the PCP is of the form[4] $\xi = \neg S_{key} \wedge \mathbf{E}(\varphi \ \mathbf{U} \ S_{match})$ where φ states that the services S_{check} and S_{next} occur along the path as described above. Moreover, (Γ, ξ) is feedback free by construction. □

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

Corollary 19. *Feedback-free artifact systems*[5] *do not have effectively regular branching-time service views.*

Note that Corollary 19 does not exclude the possibility that $\mathcal{TS}^*(\Gamma)$ might be regular for feedback-free Γ. However, it says that even if this holds, a transition system defining $\mathcal{TS}^*(\Gamma)$ cannot be *effectively* constructed for each such 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. To prevent

[4]Formally, the application of a service is detected by an additional artifact variable used as a flag, turned on by the post-condition of the service.

[5]An artifact system Γ is feedback free if (Γ, true) is feedback free.

this, we introduce a natural extension of feedback-freedom to trees of runs, called *global feedback freedom*. Recall the definition of feedback freedom from Section 2.4. 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 Since we are not dealing with verification, we define global feedback-freedom for artifact systems alone, without associated properties.

The definition of symbolic run of an artifact system can be extended to the tree of symbolic runs in the natural way, as follows. The nodes of the tree are all prefixes of symbolic runs, and there is an edge from a node ϱ to all nodes extending ϱ by one transition. We denote the tree of symbolic runs of Γ by $\mathit{TSRuns}(\Gamma)$.

In order to define global feedback freedom we need to extend the notion of computation graph from linear symbolic runs to $\mathit{TSRuns}(\Gamma)$. Assume for simplicity that all pre-and-post conditions of services are CQ$^\neg$ formulas (otherwise, Γ can be brought to this form by putting the conditions in DNF and defining one service for each pair of disjuncts from the pre-and-post condition). Likewise, we assume that the global pre-condition Π of Γ is a CQ$^\neg$ formula (otherwise, $\mathit{TSRuns}(\Gamma)$ is decomposed in a finite union of trees, one for each disjunct of Π in DNF). Note that every symbolic run $\varrho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{i\geq 0}$ has an associated symbolic constraint run $s(\varrho) = \{\sigma_i(\bar{x}_i, \bar{x}_{i+1})\}_{i\geq 0}$. We define the computation graph associated to each prefix $\mathit{TSRuns}_n(\Gamma)$ of depth n of $\mathit{TSRuns}(\Gamma)$. The computation graph of $\mathit{TSRuns}_n(\Gamma)$ is essentially the union of the computation graphs for the symbolic constraint runs corresponding to each of its branches (maximal paths). To make this more formal, we need a naming convention for variables in different branches of the tree. For each prefix $\rho = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{0\leq i\leq n}$ of a symbolic run, and $x \in \bar{x}$, let x_ρ and x_ϵ be new variables. We define inductively an injective mapping δ_ρ on $\cup_{i=0}^{n+1}\bar{x}_i$ as follows: (i) if $n = 0$ then $\delta_\rho(\bar{x}_0) = \bar{x}_\epsilon$, and $\delta_\rho(\bar{x}_1) = \bar{x}_\rho$; (ii) if $n > 0$ and $\rho = \rho' \cdot (\bar{x}_n, H_n, \epsilon_n, \sigma_n)$ then $\delta_\rho|(\cup_{i=0}^{n}\bar{x}_i) = \delta_{\rho'}|(\cup_{i=0}^{n}\bar{x}_i)$ and $\delta_\rho(\bar{x}_{n+1}) = \bar{x}_\rho$. Note that δ_ρ is defined on all $\bar{x}_i$ where $i \leq |\rho|$. For each branch $\beta = \{(\bar{x}_i, H_i, \epsilon_i, \sigma_i)\}_{0\leq i\leq n}$, let G_β and E_β be the computation graph, respectively equality graph, of the set of formulas $\{\Pi(\bar{x}_\epsilon)\} \cup \{\sigma_i(\delta_\beta(\bar{x}_i, \bar{x}_{i+1}))\}_{0\leq i\leq n}$. Let $\mathcal{G}_n$ be union of all G_β, and $\mathcal{E}_n$ the transitive closure of the union of all E_β, where β are the branches in $\mathit{TSRuns}_n(\Gamma)$. For each node ρ in $\mathit{TSRuns}_n(\Gamma)$ and variable x_ρ, we denote by $[x_\rho]$ its equivalence class with respect to $\mathcal{E}_n$. The span of $[x_\rho]$ in a branch β, denoted $\mathit{span}_\beta([x_\rho])$, is defined as in the linear case. $\mathit{TSRuns}_n(\Gamma)$ is *globally feedback free* if for every $x \in \bar{x}$, and nodes ρ_1, ρ_2

of $TSRuns_n(\Gamma)$ such that ρ_1 is a prefix of ρ_2, if there is a path from x_{ρ_1} to x_{ρ_2} in $\mathcal{G}_n$ then there exists $y \in \bar{x}$ and an ancestor ρ_3 of ρ_1 such that y_{ρ_3} occurs on the path, $span_{\rho_2}([x_{\rho_1}]) \subseteq span_{\rho_2}([y_{\rho_3}])$ and $span_{\rho_2}([x_{\rho_2}]) \subseteq span_{\rho_2}([y_{\rho_3}])$. Finally, Γ is globally feedback free if $TSRuns_n(\Gamma)$ is globally feedback free for every $n \geq 0$.

While the above definition is quite technical, it is satisfied, just like feedback freedom, by natural examples of practical business processes such as those discussed in [12], encountered in our collaboration with IBM. It can be verified that the artifact system in Example 2 is globally feedback free. The artifact system constructed in the proof of Theorem 17 is feedback free but *not* globally feedback free. We also note the following.

Remark 20. An artifact model called Hierarchical Artifact System (HAS), abstracting core elements of GSM, is studied in [20]. HAS extends the basic artifact model with task hierarchy, concurrency, updatable artifact relations, arithmetic, and database constraints (keys and foreign keys). The model is subject to a set of restrictions on the data flow among tasks, which allows verifying a significant class of temporal properties. The definition of global feedback freedom applies to the HAS model equipped with artifact variables, but without updatable artifact relations. It can be shown that such HAS are globally feedback free.

We will show the following.

Theorem 21. *The class of globally feedback-free artifact systems has effectively regular branching-time service views.*

The proof requires some technical development, which we present 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 $\pi(\bar{x}) \wedge \psi(\bar{x}, \bar{y})$.

We next define the notion of n-type of a vector $\bar{x}$ of artifact variables. An n-type is a formula associated with a subtree of depth n of $TSRuns(\Gamma)$, rooted at $\bar{x}$. The formula is obtained by composing the pre-and-post conditions of all services along the paths of the subtree. Thus, the n-type characterizes the $\bar{x}$'s that can sit at the root of such subtrees. We show that there are only finitely many non-equivalent n-types for $n \geq 0$, which then allows showing that there are only finitely many subtrees of $TRuns^*(\Gamma)$ up to bisimulation.

Definition 17. 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 $\emptyset \neq \mathcal{T}_\sigma \subseteq \mathcal{T}_n$.

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[6] $\tau_n \in \mathcal{T}_n$ 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 25). Before however, we need the following key lemma. Recall that the quantifier rank of an FO formula is the maximum number of quantifiers on a path from root to leaf in the syntax tree of the formula (e.g., see [30])

[6]With respect to logical implication.

Lemma 22. *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 $\leq b$.*

Proof. We use the term *(symbolic) pre-run* to designate a finite prefix of a (symbolic) run. The notion of *enactment* is adapted in the obvious way from runs to pre-runs. In order to show that the type of a reachable tuple is of bounded quantifier rank we relate $\tau_n(D, \nu)$ to the computation graphs of trees of symbolic runs defined earlier, then use global feedback freedom.

We say that a symbolic pre-run ϱ reaches ν in D if it has an enactment (D, ρ, h_ρ) in which ν is the last valuation in ρ. Suppose ϱ reaches ν in D. For each $n \geq 0$, we denote by $\Sigma_n(D, \nu)$ and $\mathcal{T}_n(D, \nu)$ the set of services, resp. subset of $\mathcal{T}_n$, such that

$$\tau_{n+1}(D, \nu)(\bar{x}) = \bigwedge_{\sigma \in \Sigma_n(D,\nu)} \bigwedge_{\tau \in \mathcal{T}_n(D,\nu)} \exists \bar{y}(\sigma(\bar{x}, \bar{y}) \wedge \tau(\bar{y}))$$

(recall that $\mathcal{T}_0 = \{\ true\ \}$). We rename the variables in $\tau_n(D, \nu)$ as in the earlier definition of computation graph for a tree of symbolic runs. Let ξ be the symbolic constraint pre-run of ϱ. Then we define $\tau_{n+1}(D, \nu)(\bar{x}_\xi)$ by

$$\bigwedge_{\sigma \in \Sigma_n(D,\nu)} \bigwedge_{\tau \in \mathcal{T}_n(D,\nu)} \exists \bar{x}_{(\xi.\sigma)}(\sigma(\bar{x}_\xi, \bar{x}_{(\xi.\sigma)}) \wedge \tau(\bar{x}_{(\xi.\sigma)}))$$

Observe that this renaming provides a connection between the computation graph of the (n)-type of a reachable tuple and the global computation graph. Indeed, the following is immediate from the definition.

Lemma 23. *Let ν be a reachable valuation in D, and ϱ be a symbolic pre-run that reaches ν. Let ξ be the symbolic constraint pre-run of ϱ. Then for all $n \geq 0$, the computation graph of $\tau_n(D, \nu)(\bar{x}_\xi)$ is included in $\mathcal{G}_{|\varrho|+n}$.*

We next show that $\Phi = \tau_n(D, \nu)(\bar{x}_\xi)$ is equivalent to a formula of quantifier rank bounded by $2 \times k^2$. By pushing all atoms inside the formula, Φ is equivalent to $\exists \bar{y}\phi(\bar{y}, \bar{z})$ where the free variables of ϕ are $\bar{z} \subseteq \bar{x}_\xi$. To obtain the desired result, we show that $\Phi' = \exists \bar{z} \exists \bar{y}\phi(\bar{y}, \bar{z})$ can be rewritten with quantifier rank bounded by $2 \times k^2$.

If ψ is a conjunction of literals then the computation graph G_ψ of ψ is the graph whose nodes are $var(\psi)$ and where there is an edge between two

variables u and v if and only if they both appear in a literal of ψ. Remark that the formula ϕ defined in the previous paragraph is a conjunction of literals.

In the following we denote by $\leq_p$ the relation between symbolic constraint pre-run "is a prefix of". Recall that if x_ξ is a variables of $var(\phi)$ then $[x_\xi]$ is the equivalence class of x_ξ for the equivalence relation defined by the transitive closure of $\mathcal{E}_n$. By extension, we denote by $[\phi]$ the formula obtained from ϕ by replacing each variable x by its equivalence class $[x]$.

We next define a class of formulas that will correspond to connected subgraphs of $G_{[\phi]}$:

Definition 18. Let $\exists \bar{u}\psi'$ be a formula. We say that $\exists \bar{u}\psi'$ is a connected subformula of $[\phi]$ if :

- $\bar{u} \subseteq var([\phi])$
- $(G_{\psi'})_{|\bar{u}}$ is connected.
- ψ' is a conjunction of a subset of the conjuncts of $[\phi]$

We will show that each connected subformula $\exists \bar{u}\psi'$ of $[\phi]$ can be rewritten into a formula of the form $\exists \bar{v} \bigwedge_{0\leq i\leq l} (\exists \bar{u}_i \phi_i(\bar{u}_i))$ where all the $\exists \bar{u}_i \phi_i(\bar{u}_i)$ are connected subformula and $\bar{v}$ is "small". Intuitively the set $\bar{v}$ consists of the minimum set of common variables between the $\{\exists \bar{u}_i \phi_i(\bar{u}_i)\}_{0\leq i\leq l}$.

We first define the rewriting process and bound the size of $\bar{v}$. Then we define a measure on connected subformulas and show that it strictly decreases at each application of the rewriting step (i.e. the measure of $\exists \bar{u}\psi'$ is strictly larger than the measure of $\exists \bar{u}_i \phi_i(\bar{u}_i)$) and that $\exists[\bar{z}]\exists[\bar{y}][\phi]$ can decomposed into a conjunction of connected subformulas such that the measure on each of them is bounded by a constant. This will yield an equivalent formula of bounded quantifier rank.

Observe that by definition of $\tau_n(D,\nu)(\bar{x}_\xi)$, if u_{ξ_1} and v_{ξ_2} have an edge in G_ϕ then either ξ_1 and ξ_2 are the same symbolic constraint pre-run, or one is the child of the other. As an immediate consequence, if $\mathcal{C}$ is a connected subgraph of G_ϕ then there exists a symbolic constraint pre-run ξ_p and an artifact variable $u \in \bar{x}$ such that u_{ξ_p} is a node of $\mathcal{C}$ and for all $u_\xi \in node(\mathcal{C})$, ξ_p is a prefix of ξ. We call ξ_p the *minimum prefix* of $\mathcal{C}$.

Let $\exists \bar{u}\psi'$ be a connected subformula of $[\phi]$. Assume that $\bar{u} \neq \emptyset$. Let ξ_p be the minimal prefix of $\exists \bar{u}\psi'$, and $\bar{v} = [\bar{x}_{\xi_p}] \cap \bar{u}$. Note that, since there $|\bar{x}_{\xi_p}| \leq k$,

we know that $|\bar{v}| \leq k$. Let $\bar{w}$ be the free variables of $\exists \bar{u}\psi'$ and $\bar{u}_0, ..., \bar{u}_l$ be the connected component of $(G_{\psi'})_{|\bar{u}\setminus\bar{v}}$. For all $0 \leq i \leq l$ let $\psi_i = \psi'_{|\bar{u}_i \cup \bar{v} \cup \bar{w}}$. It is easy to check that $\exists \bar{u}_i \psi_i$ are connected subformulas of $[\phi]$.

Since for all $0 \leq i \leq l$, the formula ψ_i is a restriction of ψ' to $\bar{u}_i \cup \bar{v} \cup \bar{w}$, we know that $\exists \bar{v} \bigwedge_{0 \leq i \leq l} (\exists \bar{u}_i \psi_i(\bar{u}_i))$ is implied by $\exists \bar{u}\psi'$. Conversely, let $R(a_0, ..., a_l)$ be a conjunct of ψ'; then all the variables of $\bar{a}\setminus(\bar{v} \cup \bar{w})$ are connected in $(G_{\psi'})_{\bar{u}\setminus\bar{v}}$, so there is $0 \leq i \leq l$ such that $\bar{a} \subseteq \bar{u}_i \cup \bar{v} \cup \bar{w}$. Thus, $R(a_0, ..., a_l)$ is a conjunct of $\exists \bar{u}_i \psi_i$. This shows that $\exists \bar{u}\psi'(\bar{y}')$ is equivalent to $\exists \bar{v} \bigwedge_{0 \leq i \leq l} (\exists \bar{u}_i \psi_i(\bar{u}_i))$.

Now we need to define a measure on connected subformulas and show that it decreases when a rewriting step is performed. Let $\exists \bar{u}\psi'$ be a connected subformula of $[\phi]$ and ξ_p be its minimal prefix. The measure of $\exists \bar{u}\psi'$ will be of the form $\underset{\xi_p \leq_p \beta}{max} (m_1(\exists \bar{u}\psi', \beta) + m_2(\exists \bar{u}\psi', \beta))$, where m_1 and m_2 are defined next.

To define m_1 we use an extension of the notion of width used in [12]. Specifically, the width of β in $(G_{\psi'})_{|\bar{u}}$ (denoted by $width_{(G_{\psi'})_{|\bar{u}}}(\beta)$) is defined as the number of nodes of $(G_{\psi'})_{|\bar{u}}$ that are in $[\bar{x}_\beta]$. Since $\bar{x}$ has k variables, $width_{(G_{\psi'})_{|\bar{u}}}(\beta) \leq k$. We then define $m_1(\exists \bar{u}\psi', \beta) = width_{(G_{\psi'})_{|\bar{u}}}(\beta)$.

The measure $m_2(\exists \bar{u}\psi', \beta)$ is the *support* of β in $(G_{\psi'})_{|\bar{u}}$, denoted by $support_{(G_{\psi'})_{|\bar{u}}}(\beta)$, defined as the maximum number of variables in $\bar{x}$ occurring in some path of $(G_{\psi'})_{|\bar{u}}$ starting at β. More formally:

$$support_{(G_{\psi'})_{|\bar{u}}}(\beta) = \underset{\beta \leq_p \beta''}{max} \left(|\{x \in \bar{x} \mid \exists \beta', \beta \leq_p \beta' \leq_p \beta'' \text{ and } [x_{\beta'}] \text{ is a node of } (G_{\psi'})_{|\bar{u}}\}|\right)$$

We show the following.

Lemma 24. *Let $\exists \bar{u}\psi'$ be a connected subformula of $[\phi]$. Assume that $\bar{u} \neq \emptyset$. Let $\exists \bar{v} \bigwedge_{0 \leq i \leq l} (\exists \bar{u}_i \phi_i(\bar{u}_i))$ be the rewriting of $\exists \bar{u}\psi'$ defined earlier and ξ_p be the minimal prefix of $\exists \bar{u}\psi'$. Then for all $0 \leq i \leq l$ and β, if there exists $u \in \bar{x}$ such that $u_\beta \in \bar{u}_i$ then we have*

$$m_1(\exists \bar{u}\psi', \beta) + m_2(\exists \bar{u}\psi', \beta) < m_1(\exists \bar{u}_i \phi_i(\bar{u}_i), \beta) + m_2(\exists \bar{u}_i \phi_i(\bar{u}_i), \beta).$$

Proof. Let ψ' be a formula with $var(\psi') \subseteq var([\phi])$ and β be a pre-run. We denote by $(G_{\psi'})_\beta$ the restriction of $G_{\psi'}$ to the variables in $\{[x_\varrho] \mid x \in \bar{x} \wedge \varrho \leq_p \beta\}$.

It is easy to check that $width_{(G_{\psi'})_{|\bar{u}}}(\beta) \leq width_{(G_{\phi_i})_{|\bar{u}_i}}(\beta)$ and $support_{(G_{\psi'})_{|\bar{u}}}(\beta) \leq support_{(G_{\phi_i})_{|\bar{u}_i}}(\beta)$. So it is sufficient to prove that one of the inequalities is strict.

Let β be a symbolic pre-run with $\xi_p \leq_p \beta$ and $0 \leq i \leq l$. Let $[v_{\xi_p}] \in \bar{v}$ be such that $span_\beta([v_{\xi_p}])$ is maximal. If $\beta \in span_\beta([v_{\xi_p}])$ then we know that $width_{(G_{\psi'})_{|\bar{u}}}(\beta) < width_{(G_{\phi_i})_{|\bar{u}_i}}(\beta)$.

Now assume that $\beta \notin span_\beta([v_{\xi_p}])$. Assume that there is $\beta \leq_p \beta'$ such that $[v_{\beta'}] \in \bar{u}_i$. Since a span on a branch is an "interval" and $\beta \notin span_\beta([v_{\xi_p}])$ we know that $v_{\beta'} \notin [v_{\xi_p}]$. Observe that since $\bar{u}_i$ is a connected component of $(G_{\psi'})_{|\bar{u}\setminus\bar{v}}$ and $(G_{\psi'})_{|\bar{u}}$ is connected, $(G_{\psi'})_{|\bar{u}_i\cup\bar{v}}$ is connected. Hence there is a path between $[v_{\xi_p}]$ and $[v_{\beta'}]$ in $(G_{\psi'})_{|\bar{u}_i\cup\bar{v}}$.

Since ψ' is a conjunction of literals of $[\phi]$, if there is an edge between $[x_\varrho]$ and $[y_{\varrho'}]$ in $G_{\psi'}$ then there is a path between x_ϱ and $y_{\varrho'}$ in G_ϕ such that all elements of the path are in $[x_\varrho] \cup [y_{\varrho'}]$. So there is a path between v_{ξ_p} and $v_{\beta'}$ in G_ϕ such that all nodes on the path are in $\left(\cup_{u\in(\bar{u}_i\cup\bar{v})} u\right)$. Moreover, Lemma 23 implies that there exists m such that G_ϕ is included in the global computation graph $\mathcal{G}_m$. Since $\mathcal{G}_m$ is globally feedback free, there exist ξ_g with $\xi_g \leq_p \xi_p$ and $x \in \bar{x}$ such that x_{ξ_g} is on the path and

$$span_{\beta'}([v_{\beta'}]) \cup span_{\beta'}([v_{\xi_p}]) \subseteq span_{\beta'}([x_{\xi_g}]).$$

Since x_{ξ_g} is on the path, $[x_{\xi_g}] \in \bar{u}_i \cup \bar{v}$. Moreover since ξ_p is a minimal prefix of $\exists\bar{u}.\psi'$, it follows that $\xi_p \leq_p \xi_g$, so $\xi_g = \xi_p$ and $[x_{\xi_g}] \in \bar{v}$. Additionally, $\beta \notin span_\beta([v_{\xi_p}])$ and $\beta \leq_p \beta'$, so $span_{\beta'}([v_{\beta'}])$ and $span_{\beta'}([v_{\xi_p}])$ are disjoint, which implies that the above inclusion is strict. This contradicts the maximality of $span_\beta([v_{\xi_p}])$. So there is no $\beta \leq_p \beta'$ such that $[v_{\beta'}] \in \bar{u}_i$. This implies that $support_{(G_{\psi'})_{|\bar{u}}}(\beta) < support_{(G_{\phi_i})_{|\bar{u}_i}}(\beta)$, which concludes the proof of Lemma 24. □

Finally, it remains to initialize the process by decomposing $(G_{[\phi]})_{|[\bar{y}]\cup[\bar{z}]}$ into connected components $\bar{y}_0, \ldots, \bar{y}_r$ so that $[\phi]$ is equivalent to $\bigwedge_{0\leq i\leq r}\left(\exists\bar{y}_i[\phi]_{|\bar{y}_i}\right)$. Now all the formulas $\exists\bar{y}_i[\phi]_{|\bar{y}_i}$ are connected subformulas of $[\phi]$, so we can apply Lemma 24. Each application of the lemma to $\exists\bar{u}\psi'$ yields new connected subformulas $\{\exists\bar{u}_i\psi_i\}_{0\leq i\leq l}$ and $\bar{v}$ (with less than k variables) such that $\exists\bar{u}\psi'(\bar{y}')$ is equivalent to $\exists\bar{v}\bigwedge_{0\leq i\leq l}(\exists\bar{u}_i\phi_i(\bar{u}_i))$. Furthermore, it can be easily shown that the last condition yields an upper bound of $2 \times k$ on the number of nested applications of the lemma. At each step we add at most k existential quantifiers, so the quantifier rank of the resulting formula is bounded by $2 \times k^2$. This concludes the proof of Lemma 22. □

Using Lemma 22, we show the following.

Lemma 25. *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 $TRuns_D^{\nu_1}(\Gamma)$ and $TRuns_D^{\nu_2}(\Gamma)$ are bisimilar.*

Proof. For a node ρ of $TRuns_D(\Gamma)$ we denote its label by $\lambda(\rho)$. We define a sequence of equivalence relations $(\sim_n)_{n\geq 0}$ on nodes of $TRuns_D(\Gamma)$ as follows:

- $\sim_0$ is the cross product of all nodes
- $\rho_1 \sim_{n+1} \rho_2$ if for every edge $\rho_1 \xrightarrow{\sigma} \rho_1'$ there exists an edge $\rho_2 \xrightarrow{\sigma} \rho_2'$ such that $\rho_1' \sim_n \rho_2'$, and conversely

Intuitively, $\rho_1 \sim_n \rho_2$ says that the subtrees of depth n rooted at ρ_1 and ρ_2 are bisimilar. Clearly, $\sim_{n+1}$ refines $\sim_n$ for every $n \geq 0$. The following is easily shown by induction:

$$(\dagger) \text{ for every } n \geq 0, \rho_1 \sim_n \rho_2 \text{ iff } \tau_n(D, \lambda(\rho_1)) = \tau_n(D, \lambda(\rho_2))$$

By Lemma 22, there are finitely many non-equivalent formulas $\tau_n(D, \lambda(\rho))$ for nodes ρ in $TRuns_D(\Gamma)$. Therefore, there are finitely many distinct relations $\sim_n$ for $n \geq 0$. Since $(\sim_n)_{n\geq 0}$ is a decreasing sequence (with respect to refinement), there exists $N > 0$ such that $\sim_n = \sim_N$ for every $n \geq N$. In particular, $\sim_{N+1} = \sim_N$. It easily follows that $\sim_N$ is a bisimilarity relation on $TRuns_D(\Gamma)$. Also, for all nodes ρ_1, ρ_2 such that $\tau_n(D, \lambda(\rho_1)) = \tau_n(D, \lambda(\rho_2))$ for every $n \geq 0$, $\rho_1 \sim_N \rho_2$ so $TRuns_D^{\lambda(\rho_1)}(\Gamma)$ and $TRuns_D^{\lambda(\rho_2)}(\Gamma)$ are bisimilar. □

From Lemma 22 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 22, 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 19. The type of a database D is $\tau(D) = \{\tau^*(D, \nu) \mid D \models \Pi(\nu)\}$.

We have the following.

Lemma 26. *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 22 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 21.

Remark 27. Theorem 21 continues to hold for artifact systems extended with arithmetic (e.g., linear inequalities with integer coefficients over $\mathbb{Q}$). To see this, augment $\mathcal{DB}$ with a finite set $\mathcal{C}$ of relation symbols with fixed interpretations as linear constraints, 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 22 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.

As noted earlier, effective regularity of the branching-time service views of a class of systems generally implies decidability of its CTL*(QFO) 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*(QFO) property. Taking into account the property is done similarly to feedback-freedom (details omitted). We can then show the following.

Theorem 28. *It is decidable, given an artifact system Γ and a CTL*(QFO) 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 focus on constrained artifact systems $\Gamma(\Delta)$ where Δ is a set of equality or tuple-generating dependencies (see definitions in Section 2.2).

Linear-time service views We first consider linear-time service views. Let $\Gamma(\Delta)$ be a constrained artifact system. The *linear-time service view* of $\Gamma(\Delta)$ is $\mathcal{S}^{\Delta}_{lin}(\Gamma) = \{\mathcal{S}(\rho) \mid \rho \in \mathit{Runs}(\Gamma(\Delta))\}$. We say that a class $\mathbf{A}$ of artifact systems constrained by a class $\mathbf{D}$ of dependencies has effectively ω-regular linear-time service views if there is an algorithm which, given $\Gamma \in \mathbf{A}$ and $\Delta \in \mathbf{D}$, produces a Büchi automaton defining $\mathcal{S}^{\Delta}_{lin}(\Gamma)$. Observe that the requirement of an effective construction is essential: it is possible in principle that $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ may be regular for each $\Gamma \in \mathbf{A}$ and $\Delta \in \mathbf{D}$, without $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ being effectively constructible from Γ and Δ.

We can show the following.

Theorem 29. *Artifact systems constrained by EGDs do not have effectively ω-regular linear-time service views. Moreover, this holds even if the EGDs are limited to a single FD.*

Proof. It can be shown, similarly to Lemma 18, that it is undecidable, given an artifact system Γ, a set Δ of EGDs, and a service σ, whether there exists a run of $\Gamma(\Delta)$ in which service σ is used. This holds even if Δ consists of a single FD. The result follows. □

Note that, similarly to Corollary 19, Theorem 29 leaves open the possibility that $\mathcal{S}^{\Delta}_{lin}(\Gamma)$ might be ω-regular.

Remark 30. 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 29 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 [5, 41]).

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

Theorem 31. *Artifact systems constrained by embedded TGDs have effectively ω-regular linear-time service views.*

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 7, 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 32. *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 33. *Artifact systems constrained by acyclic sets of full TGDs have effectively ω-regular linear-time service views.*

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 an atom 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 7, 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 $SRuns(\Gamma) \cap \mathcal{L}(\mathcal{B}^c)$, where $\mathcal{B}^c$ defines the complement of $\mathcal{L}(\mathcal{B})$. Finally, $\mathcal{S}^{\Delta^*}_{lin}(\Gamma) = h(SRuns(\Gamma) \cap \mathcal{L}(\mathcal{B}^c))$, 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 ω-regular and its specification can be effectively constructed. □

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 [12]. One might hope that

feedback-free systems are similarly well-behaved with respect to linear service views. Indeed, in contrast to Theorems 29 and 32, we have the following.

Theorem 34. *Feedback-free artifact systems constrained by sets of EGDs and full TGDs have effectively ω-regular linear-time service views.*

PROOF. The approach is similar to that of [12] 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 [12], 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 [30], 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 21, the views are effectively regular for globally feedback-free systems.

Let Γ be an artifact system and Δ a set of dependencies. The branching-time service view of $\Gamma(\Delta)$, denoted $\mathcal{TS}^*(\Gamma(\Delta))$, is obtained as before from $TRuns^*(\Gamma(\Delta))$ by ignoring the contents of the nodes and retaining only the service labels of the edges. We say that a class $\mathbf{A}$ of artifact systems constrained by a class $\mathbf{D}$ of dependencies has effectively regular branching-time service views if there is an algorithm which, given $\Gamma \in \mathbf{A}$ and $\Delta \in \mathbf{D}$, produces a finite-state transition system specifying $\mathcal{TS}^*(\Gamma(\Delta))$.

We can show the following.

Theorem 35. *Globally feedback-free artifact systems constrained by sets of EGDs and full TGDs have effectively regular branching-time service views.*

PROOF. Recall the proof of Theorem 21 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 $\mathcal{TS}^*(\Gamma(\Delta))$ is regular, and a specification can be effectively constructed from Γ and Δ. □

Remark 36. Theorem 35 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 conditions. 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 violations of LTL(QFO) properties. As a side-effect of results on branching-time service views, we showed that CTL(QFO) properties are undecidable for artifact systems, but model-checking CTL*(QFO) 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 [43]. Besides the technical problems *per se*, this brings into play interesting HCI and usability issues.

7. Acknowledgements

This work was supported in part by the National Science Foundation under award IIS - 1422375.

References

[1] Abiteboul, S., Hull, R., Vianu, V., 1995. Foundations of Databases. Addison Wesley.

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

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

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

[5] Boasson, L., Nivat, M., 1980. Adherences of languages. J. Comput. System Sci. 20 (3).

[6] Bozzon, A., Brambilla, M., Ceri, S., Mauri, A., 2013. Reactive crowdsourcing. In: 22nd International World Wide Web Conference, WWW, Rio de Janeiro, Brazil. pp. 153–164.

[7] Bozzon, A., Brambilla, M., Ceri, S., Mauri, A., Volonterio, R., 2014. Pattern-based specification of crowdsourcing applications. In: Web Engineering, 14th International Conference, ICWE 2014, Toulouse, France. pp. 218–235.

[8] Calvanese, D., De Giacomo, G., Montali, M., 2013. Foundations of data-aware process analysis: a database theory perspective. In: ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS).

[9] Chao, T., et al., 2009. Artifact-based transformation of IBM Global Financing: A case study. In: Int'l. Conf. on Business Process Management (BPM).

[10] Clarke, E., Grumberg, O., Peled, D., 2000. Model Checking. MIT Press.

[11] CMMN, 2013. Case Management Model and Notation, FTF Beta 1. OMG Document Number dtc/2013-01-01, Object Management Group. URL `http://www.omg.org/spec/CMMN/1.0/Beta1/`

[12] Damaggio, E., Deutsch, A., Vianu, V., 2012. Artifact systems with data dependencies and arithmetic. ACM Trans. Database Syst. 37 (3), Preliminary version in *ICDT 2011*.

[13] Damaggio, E., Hull, R., Vaculín, R., 2013. On the equivalence of incremental and fixpoint semantics for business artifacts with Guard-Stage-Milestone lifecycles. Information Systems 38, 561–584.

[14] De Giacomo, G., Masellis, R. D., Rosati, R., 2012. Verification of conjunctive artifact-centric services. Int. J. Cooperative Inf. Syst. 21 (2), 111–140.

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

[16] Demri, S., Lazić, R., 2006. LTL with the Freeze Quantifier and Register Automata. In: ACM/IEEE Symp. on Logic in Computer Science (LICS).

[17] Demri, S., Lazić, R., Sangnier, A., 2008. Model checking freeze LTL over one-counter automata. In: FoSSaCS.

[18] Deutsch, A., Hull, R., Patrizi, F., Vianu, V., 2009. Automatic verification of data-centric business processes. In: Int'l. Conf. on Database Theory (ICDT).

[19] Deutsch, A., Hull, R., Vianu, V., 2014. Automatic verification of data-driven systems. Sigmod Record 43 (3), 5–17.

[20] Deutsch, A., Li, Y., Vianu, V., 2016. Verification of hierarchical artifact systems. In: ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS).

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

[22] Emerson, E. A., 1990. Temporal and modal logic. In: Leeuwen, J. V. (Ed.), Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics. North-Holland Pub. Co./MIT Press, pp. 995–1072.

[23] Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M., 2013. Verification of relational data-centric dynamic systems with external services. In: ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS).

[24] Hariri, B. B., Calvanese, D., De Giacomo, G., Masellis, R. D., Felli, P., 2011. Foundations of relational artifacts verification. In: Intl. Conf. on Business Process Management (BPM). pp. 379–395.

[25] Hariri, B. B., Calvanese, D., Montali, M., Giacomo, G. D., Masellis, R. D., Felli, P., 2013. Description logic knowledge and action bases. J. of Art. Intelligence Res. 46, 651–686.

[26] Hull, R., Damaggio, E., Masellis, R. D., Fournier, F., Gupta, M., III, F. H., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculín, R., 2011. Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In: ACM Int'l. Conf. on Distributed Event-Based Systems (DEBS).

[27] Isaak, D., Löding, C., 2012. Efficient inclusion testing for simple classes of unambiguous -automata. Inf. Process. Lett. 112 (14-15).

[28] Koutsos, A., Vianu, V., 2015. Process-centric views of data-driven business artifacts. In: Int'l. Conf. on Database Theory (ICDT).

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

[30] Libkin, L., 2004. Elements of Finite Model Theory. Springer.

[31] Lomuscio, A., Michaliszyn, J., 2014. 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.

[32] Marin, M., Hull, R., Vaculín, R., 2012. Data centric BPM and the emerging case management standard: A short survey. In: BPM Workshop.

[33] Marnette, B., 2009. Generalized schema-mappings: from termination to tractability. In: ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS).

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

[35] Minsky, M. L., 1967. Computation: finite and infinite machines. Prentice-Hall.

[36] Nigam, A., Caswell, N. S., 2003. Business artifacts: An approach to operational specification. IBM Systems Journal 42 (3).

[37] Pnueli, A., 1977. The temporal logic of programs. In: FOCS.

[38] Post, E. L., 1947. Recursive unsolvability of a problem of Thue. J. of Symbolic Logic 12, 1–11.

[39] Segoufin, L., Torunczyk, S., 2011. Automata based verification over linearly ordered data domains. In: STACS.

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

[41] Thomas, W., 1990. Handbook of theoretical computer science (vol. b). Ch. Automata on Infinite Objects.

[42] van der Aalst, W., Song, M., 2004. Mining social networks: Uncovering interaction patterns in business processes. In: Business Process Management. Vol. 3080 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 244–260.

[43] van der Aalst, W., ter Hofstede, A., 2005. YAWL: Yet another workflow language. Information Systems 30 (4).

[44] Zhu, W., et al., 2014. Advanced case management with IBM case manager. Available at `http://www.redbooks.ibm.com/abstracts/sg247929.html?Open`.