Artifact Systems with Data Dependencies and Arithmetic*

Elio Damaggio
University of California San Diego
elio@cs.ucsd.edu

Alin Deutsch
University of California San Diego
deutsch@cs.ucsd.edu

Victor Vianu†
University of California San Diego
vianu@cs.ucsd.edu

ABSTRACT

We revisit the static verification problem for data centric business processes, specified in a variant of IBM's "business artifact" model. Artifacts are records of variables that correspond to business-relevant objects and are updated by a set of services equipped with pre-and-post conditions, that implement business process tasks. The verification problem consists in statically checking whether all runs of an artifact system satisfy desirable properties expressed in a first-order extension of linear-time temporal logic. In previous work we identified the class of *guarded* artifact systems and properties, for which verification is decidable. However, the results suffer from an important limitation: they fail in the presence of even very simple data dependencies or arithmetic, both crucial to real-life business processes. In this paper, we extend the artifact model and verification results to alleviate this limitation. We identify a practically significant class of business artifacts with data dependencies and arithmetic, for which verification is decidable. The technical machinery needed to establish the results is fundamentally different from our previous work. While the worst-case complexity of verification is non-elementary, we identify various realistic restrictions yielding more palatable upper bounds.

Categories and Subject Descriptors

D.2.4 [**Software/Program Verification**]: Model checking; D.3.2 [**Language Classifications**]: Specialized application languages; H.4.1 [**Office Automation**]: Workflow management

General Terms

Theory, Verification

Keywords

verification, model checking, arithmetic, data dependencies, data-aware, integrity constraints, IBM, business process, business entity with lifecycles, business artifacts, workflows

*This work was supported by the National Science Foundation under award III-0916515 and by IBM Research through an Open Collaborative Research grant in conjunction with Project ArtiFact.
†This author was supported in part by the European Research Council under grant Webdam, agreement 226513.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICDT 2011, March 21–23, 2011, Uppsala, Sweden.
Copyright 2011 ACM 978-1-4503-0529-7/11/0003 ...$10.00

1. INTRODUCTION

Business process management is central to the operation of organizations in various domains, ranging from business to governmental, scientific, and beyond. Recent years have witnessed the evolution of business process specification frameworks from the traditional process-centric approach towards data-awareness. Process-centric formalisms focus on control flow while under-specifying the underlying data and its manipulations by the process tasks, often abstracting them away completely. In contrast, data-aware formalisms treat data as first-class citizens. A notable exponent of this class is the *business artifact model* pioneered in [44], deployed by IBM in commercial products and consulting services, and further studied in a line of follow-up works [4, 5, 24, 25, 6, 36, 31, 33]. Business artifacts (or simply "artifacts") model key business-relevant entities, which are updated by a set of services that implement business process tasks. A collection of artifacts and services is called an *artifact system*. This modeling approach has been successfully deployed in practice, yielding proven savings when performing business process transformations [4].

In previous work [17], we addressed the verification problem for artifact systems, which consists in statically checking whether all runs of an artifact system satisfy desirable properties expressed in a first-order extension of linear-time temporal logic. We considered artifact systems in which each artifact contains a record of variables and the services may consult (though not update) an underlying database. Services can also set the artifact variables to new values from an infinite domain, thus modeling external inputs and tasks specified only partially, using pre- and post-conditions. Post-conditions permit non-determinism in the outcome of a service, to capture for instance tasks in which a human makes a final determination about the value of an artifact variable, subject to certain constraints. This setting resulted in a challenging infinite-state verification problem, due to the infinite data domain. [17] identified the large and useful class of "guarded" artifact systems and properties, for which verification is decidable. However, the positive results of [17] suffer from an important shortcoming: they fail in the presence of even very simple data dependencies or arithmetic, both crucial to real-life business processes. In this paper we revisit the static verification problem in order to alleviate this limitation. Specifically, we provide a practically motivated class of artifact systems that allow data dependencies (integrity constraints on the database) and arithmetic operations performed by services, for which verification becomes decidable. The technical machinery needed to establish this result is fundamentally different from that of [17].

We illustrate the variant of the artifact systems model adopted in this paper by means of a running example that models an e-commerce business process in which the customer chooses a product and a shipment method and applies various kinds of coupons

to the order. The business process exhibits a flexibility that, while desirable in practice for a postive customer experience, yields intricate runs, all of which need to be considered in verification. For instance, at any time before submitting a valid payment, the customer may edit the order (select a different product, shipping method, or change/add a coupon) an unbounded number of times. Likewise, the customer may cancel an order for a refund even after submitting a valid payment. The business process is partially modeled in Example 2.8.

The running example features three characteristics that drive the motivation for our work.

1. The system routinely queries an underlying database, for instance to look up the price of a product and the shipping weight restrictions.

2. The validity checks and updates carried out by the services involve arithmetic operations. For instance, to be valid, an order must satisfy such conditions as: (a) the product weight must be within the selected shipment method's limit, and (b) if the buyer uses a coupon, the sum of product price and shipping cost must exceed the coupon's minimum purchase limit.

3. Finally, the correctness of the business process relies on database integrity constraints. For instance, the system must check that a selected triple of product, shipment type and coupon are globally compatible. This check is implemented by several local tests, each running at a distinct instant of the interaction, as user selections become available. Each local test accesses distinct tables in the database, yet they globally refer to the same product, due to the keys and foreign keys satisfied by these tables.

The properties we are interested in verifying are expressed in a first-order extension of linear temporal logic called LTL-FO. This is a powerful language, fit to capture a wide variety of business policies implmented by a business process. For instance, in our running example it allows us to express such desiderata as:

> 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.
>
> A free shipment coupon is accepted only if the available quantity of the product is greater than zero, the weight of the product is in the limit allowed by the shipment method, and the sum of price and shipping cost exceeds the coupon's minimum purchase value.

Our previous results from [17] do not apply in the new context, as we show undecidability even when adding to a guarded artifact system a single functional dependency, or alternately, when the only allowed arithmetic operation consists in incrementing counters.

We identify the condition of *feedback-freedom* which covers a useful class of business processes and yields decidability when (i) the arithmetic operations involve linear expressions with integer coefficients over artifact variables over the domain of rational numbers, while (ii) the database satisfies embedded dependencies for which the chase terminates.

In fact, our decidability result is more general, extending to any set of operations as long as we have decidability of satisfiability of $\exists$FO formulas over the vocabulary $\mathcal{C}$ of these operations. This happens to be the case for linear arithmetic expressions with integer coefficients, but our result is generic: the verification algorithm is modular, calling the satisfiability checker for $\mathcal{C}$ as a black box.

The proof technique is fundamentally different from the one employed in [17] for guardedness, and interesting in its own right. It is based on describing runs symbolically, capturing, for each snapshot s in the run, only the part of the run prefix that is relevant to the artifact variables at s. This description is expressible as $\exists$FO formulae over the database schema and $\mathcal{C}$, called *inherited constraints*. Feedback-freedom determines a static bound on the number of distinct inherited constraints (up to logical equivalence) and therefore on the length of symbolic runs the verifier needs to explore.

The positive decidability results for feedback-free systems come at the cost of high worst-case complexity, hyperexponential in the number of artifact variables (so non-elementary). We identify various realistic restrictions leading to improved upper bounds. These consist of a stricter notion of feedback-freedom (called acyclicity), a bound on the number of related artifact variables, and a restriction on the propagation of artifact variable values from one configuration to the next. For example, acyclicity yields a double-exponential complexity upper bound. We note that no lower bound has yet been proven, for the general case or its restrictions.

Paper outline Our model of artifact systems and the running example are introduced in Section 2. Section 3 presents the property language LTL-FO, and our undecidability results showing that the techniques developed in [17] for guarded artifact systems do not apply in the presence of integrity constraints and arithmetic. We introduce feedback-freedom in Section 4, and show in Section 5 that it leads to decidable verification, without yet considering dependencies. Restrictions of feedback-freedom for improved upper bounds are dicussed in Section 5.2. We extend our decidability result to the presence of dependencies in Section 6. Related work is discussed in Section 7. We end with brief conclusions.

2. FRAMEWORK

The arithmetic constraints considered here are over domain $\mathbb{Q}$, the rational numbers. While databases could use non-numeric data, we assume for uniformity, and without loss of generality, that all structures are over $\mathbb{Q}$. We denote by $\mathcal{C}$ an infinite set of relation symbols, each of which has a fixed interpretation as the set of solutions of a finite set of linear inequalities with integer coefficients. By slight abuse, we sometimes use the same notation for a relation symbol in $\mathcal{C}$ and its fixed interpretation.

DEFINITION 2.1. *An* artifact schema *is a a tuple $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ where $\bar{x}$ is a finite set 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.

DEFINITION 2.2. *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 $\mathbb{Q}$ and D is a finite instance of $\mathcal{DB}$ whose domain is included in $\mathbb{Q}$.*

We denote by $\exists$FO the first-order formulas whose prenex form uses only existential quantification, and by $\text{CQ}^\neg$ the formulas built from literals (positive and negated atoms over $\mathcal{DB} \cup \mathcal{C} \cup \{=\}$) using only conjunction and existential quantification.

DEFINITION 2.3. *A* service *over an artifact schema $\mathcal{A}$ is a pair $\sigma = \langle \pi, \psi \rangle$ where:*

- *$\pi(\bar{x})$, called* pre-condition*, is an $\exists$FO formula using relational symbols in $\mathcal{DB} \cup \mathcal{C}$, with free variables $\bar{x}$;*
- *$\psi(\bar{x}, \bar{x}')$, called* post-condition*, is an $\exists$FO formula on the relational symbols in $\mathcal{DB} \cup \mathcal{C}$, with free variables $\bar{x} \cup \bar{x}'$.*

DEFINITION 2.4. *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 (as above, a $\exists$FO formula over $\mathcal{DB} \cup \mathcal{C}$, with free variables $\bar{x}$).*

DEFINITION 2.5. *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 $A \xrightarrow{\sigma} A'$ when D is understood) iff:*

- $D \cup \mathcal{C} \models \pi(\nu)$, *and*
- $D \cup \mathcal{C} \models \psi(\nu, \nu')$.

DEFINITION 2.6. *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 $\rho = \{\rho_i\}_{i \geq 0}$ of valuations of $\bar{x}$ so that $D \cup \mathcal{C} \models \Pi(\rho_0)$ and for each $i \geq 0$, $\rho_i \xrightarrow{\sigma} \rho_{i+1}$ for some $\sigma \in \Sigma$.*

We denote by $Runs_D(\Gamma)$ the set of all runs of Γ on database instance D.

REMARK 2.7. *In [17], artifacts are equipped with state relations in addition to the database and artifact variables. However, under the guarded restriction, the state relations are essentially limited to be finite-state. Note that finite state control can be simulated with artifact variables, by having one variable hold the current state. For instance, this role is played by variable* status *in the example below. We therefore omit explicit states in the present model.*

We next illustrate the expressive power of the artifact system framework by modeling the running e-commerce example. Due to limited space, we only list some of the services involved.

Example 2.8 The running example models an e-commerce business process in which the customer chooses a product and a shipment method and applies various kinds of coupons to the order. There are two kinds of coupons: discount coupons subtract their value from the total (e.g. a $50 coupon) and free-shipment coupons subtract the shipping costs from the total. The order is filled in a sequential manner (first pick the product, then the shipment, then claim a coupon), 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.

We define an artifact with the following variables:

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

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

"edit_product", "edit_ship", "edit_coupon", "processing", "received_payment", "shipping", "shipped", "canceling", "canceled".

Artifact variables `ship_type` and `coupon` record the customer's selection, received as an external input. `amount_paid` is also an external input (from the customer, possibly indirectly via a credit card service). Variable `amount_owed` is set by the system using arithmetic operations that sum up product price and shipment cost, subtracting the coupon value. Variable `amount_refunded` is set by the system in case a refund is activated.

The database includes the following tables (underlined attributes denote keys):

`PRODUCTS`(<u>`id`</u>, `price`, `availability`, `weight`),
`COUPONS`(<u>`code`</u>, `type`, `value`, `min_value`, `free_shiptype`),
`SHIPPING`(<u>`type`</u>, `cost`, `max_weight`),
`OFFERS`(<u>`prod_id`</u>, `discounted_price`, `active`).

The database also satisfies the following foreign keys:

`COUPONS`[`free_shiptype`] $\subseteq$ `SHIPPING`[`type`], and
`OFFERS`[`prod_id`] $\subseteq$ `PRODUCTS`[`id`].

Our framework's domain is $\mathbb{Q}$, however, in order to enhance readability and without loss of generality, we allow non-numeric attributes over arbitrary domains, including in particular enumeration types (as for the `status` artifact variable).

The starting configuration has `status` initialized to "edit_prod", and all other variables to "undefined". By convention, in this example we model undefined variables using the reserved constant λ. (This is syntactic sugar and does not affect the artifact systems model. In the example for instance, any non-positive value can play this role.) The initialization is easily expressed by the artifact system's pre-condition Π.

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

choose_product The customer chooses a product.
π : `status` = "edit_prod"
ψ : $\exists p, a, w$(`PRODUCTS`(`prod_id`$'$, p, a, w) $\wedge$ $a > 0$)
$\wedge$`status`$'$ = "edit_shiptype"

choose_shiptype The customer chooses a shipping option.
π : `status` = "edit_ship"
ψ : $\exists c, l, p, a, w$(`SHIPPING`(`ship_type`$'$, c, l)$\wedge$
`PRODUCTS`(`prod_id`, p, a, w) $\wedge$ $l > w$)$\wedge$
`status`$'$ = "edit_coupon" $\wedge$ `prod_id`$'$ = `prod_id`

apply_coupon The customer optionally inputs a coupon number.
π : `status` = "edit_coupon"
ψ : (`coupon`$'$ = $\lambda \wedge \exists p, a, w, c, l$(`PRODUCTS`(`prod_id`, p, a, w)
$\wedge$`SHIPPING`(`ship_type`, c, l) $\wedge$ `amount_owed`$'$ = $p + c$)$\wedge$
`status`$'$ = "processing" $\wedge$ `prod_id`$'$ = `prod_id`$\wedge$
`ship_type`$'$ = `ship_type`)$\vee$
($\exists t, v, m, s, p, a, w, c, l$(`COUPONS`(`coupon`$'$, t, v, m, s)$\wedge$
`PRODUCTS`(`prod_id`, p, a, w) $\wedge$ `SHIPPING`(`ship_type`, c, l)
$\wedge p + c \geq m \wedge$ (t = "free_shipping" $\rightarrow$
(s = `ship_type` $\wedge$ `amount_owed`$'$ = p))$\wedge$
(t = "discount" $\rightarrow$ `amount_owed`$'$ = $p + c - v$))
$\wedge$`status`$'$ = "processing" $\wedge$ `prod_id`$'$ = `prod_id`$\wedge$
`ship_type`$'$ = `ship_type`)

Notice that the pre-conditions π of the services check the value of the `status` variable. For instance, according to **choose_product**, the customer can only input her product choice while the order is in "edit_prod" status.

Also notice that the post-conditions ψ constrain the next values of the artifact variables (denoted by a prime). For instance, according to **choose_product**, once a product has been picked, the next value of the status variable is "edit_shiptype", which will at a subsequent step enable the **choose_shiptype** service (by satisfying its pre-condition). Similarly, once the shipment type is chosen (as modeled by service **choose_shiptype**), the new status is "edit_coupon", which enables the **apply_coupon** service. The interplay of pre- and post-conditions achieves a sequential filling of the order, starting from the choice of product and ending with the claim of a coupon.

A post-condition may refer to both the current and next values of the artifact variables. For instance, in service **choose_shiptype**, the fact that only the shipment type is picked while the product remains unchanged, is modeled by preserving the product id: the next and current values of the corresponding artifact variable are set equal.

Pre- and post-conditions may query the database. For instance, in service **choose_product**, the post-condition ensures that the product id chosen by the customer is that of an available product (by checking that it appears in a `PRODUCTS` tuple, whose availability attribute is positive).

Finally, notice the arithmetic computation in the post-conditions. For instance, in service **apply_coupon**, the sum of the product

price p and shipment cost c (looked up in the database) is adjusted with the coupon value (notice the distinct treatment of the two coupon types) and stored in the `amount_owed` artifact variable.

Observe that the first post-condition disjunct models the case when the customer inputs no coupon number (the next value $\texttt{coupon}'$ is set to undefined), in which case a different owed amount is computed, namely the sum of price and shipping cost. □

Dependencies We consider integrity constraints of the form

$$\forall \bar{u}\bar{w}\ \phi(\bar{u},\bar{w}) \rightarrow \exists \bar{v}\ \psi(\bar{u},\bar{v})$$

where ϕ and ψ are conjunctions of relational and equality atoms (positive literals over the vocabulary including the relational schema and the equality predicate, respectively). Such sentences are known as *embedded dependencies* and are sufficiently expressive to specify all usual integrity constraints, such as keys, foreign keys, inclusion, join, multivalued dependencies, etc. [22, 2]. In this paper, we refer to embedded dependencies in short as "dependencies". We call ϕ the *premise* and ψ the *conclusion*. We write $A \models \Sigma$ if the instance A satisfies all the dependencies in Σ.

Example 2.9 We illustrate dependencies continuing Example 2.8. The key constraint`PRODUCTS` is expressed by the dependency

$$\begin{aligned}\forall i, p_1, a_1, w_1, p_2, a_2, w_2 \\ &\texttt{PRODUCTS}(i, p_1, a_1, w_1) \wedge \texttt{PRODUCTS}(i, p_2, a_2, w_2) \\ &\rightarrow p_1 = p_2 \wedge a_1 = a_2 \wedge w_1 = w_2.\end{aligned}$$

The foreign key constraint on the `OFFERS` table is expressed by

$$\forall i, d, v\ \texttt{OFFERS}(i, d, v) \rightarrow \exists p, a, w\ \texttt{PRODUCTS}(i, p, a, w). \quad \square$$

3. TEMPORAL PROPERTIES OF ARTIFACT SYSTEMS

In order to specify temporal properties we use an extension of LTL (linear-time temporal logic). Recall that LTL is propositional logic augmented with temporal operators X (next) and U (until) (e.g., see [45]). The extension we use, called[1] LTL-FO, is obtained from LTL by interpreting propositions as FO statements about particular artifact instances in the run. The different statements may share some global variables, that are universally quantified.

DEFINITION 3.1. *Let $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ be an artifact schema. An* FO component *over $\mathcal{A}$ is a quantifier-free FO formula over $\mathcal{DB} \cup \mathcal{C}$. An LTL-FO 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 FO 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-FO formulas is defined as follows. Let $\langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, $\forall \bar{y} \varphi_f$ an LTL-FO formula over $\mathcal{A}$, and ρ a run of $\langle \mathcal{A}, \Sigma, \Pi \rangle$ on database D. Let μ be a valuation of $\bar{y}$ into $\mathbb{Q}$. An FO component $\psi(\bar{x}, \bar{x}', \bar{y})$ of φ_f is satisfied in ρ_i with valuation μ if $D \cup \mathcal{C} \models \psi(\rho_i, \rho_{i+1}, \mu)$, $i \geq 0$. 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 $\mathbb{Q}$.

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

We say that an artifact system Γ satisfies an LTL-FO sentence φ, denoted $\Gamma \models \varphi$, if all runs of Γ satisfy φ. Note that the database is fixed for each run, but may be different for different runs.

We illustrate LTL-FO in the context of Example 2.8.

Example 3.2 We show a few properties that specify desirable business rules for the running example.

$$\begin{aligned}(\varphi_1)\ \forall x \mathbf{G}((&\texttt{amount_paid} = x \wedge \texttt{amount_paid} = \texttt{amount_owed}) \\ &\rightarrow \mathbf{F}(\texttt{status} = \text{"shipped"} \vee \texttt{amount_refunded} = x))\end{aligned}$$

Property φ_1 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. φ_1 is obtained from LTL property $\varphi = \mathbf{G}(p \rightarrow \mathbf{F}q)$ via the mapping f_1, where $f_1(p) = \texttt{amount_paid} = x \wedge \texttt{amount_paid} = \texttt{amount_owed}$ and $f_1(q) = \texttt{status} = \text{"shipped"} \vee \texttt{amount_refunded} = x$. Note the use of universally-quantified variable `x` to relate the value of paid and refunded amounts across distinct steps in the run sequence.

$$\begin{aligned}(\varphi_2)\ \forall\, v, m, s, p, a, w, c, l (&\mathbf{G}(\texttt{prod_id} \neq \lambda \wedge \texttt{ship_type} \neq \lambda \\ &\wedge \texttt{COUPONS}(\texttt{coupon}, \text{"free_ship"}, v, m, s)) \wedge \\ &\texttt{PRODUCTS}(\texttt{prod_id}, p, a, w) \wedge \texttt{SHIPPING}(\texttt{ship_type}, c, l) \\ &\rightarrow \underbrace{a > 0}_{(i)} \wedge \underbrace{w \leq l}_{(ii)} \wedge \underbrace{p + c \geq m}_{(iii)})\end{aligned}$$

Property φ_2 verifies the consistency of orders that use coupons for free shipping. The premise of the implication lists the conditions for a completely specified order that uses such coupons. The conclusion checks the following business rules (i) available quantity of the product is greater than zero, (ii) the weight of the product is in the limit allowed by the shipment method, and (iii) the total order value satifies the minimum for the application of the coupon.

Note that this property holds only due to the integrity constraints on the schema. Indeed, observe that (i) is guaranteed by the post-condition of service **choose_product**, (ii) by **choose_shiptype**, and (iii) by **apply_coupon**. In the post-conditions, the checks are perfomed by looking up in the database the weight/price/cost/limit attributes associated to the customer's selection of product id and shipment type (stored in artifact variables). The property performs the same lookup in the database, and it is guaranteed to retrieve the same tuples only because product id and shipment type are keys for `PRODUCTS`, respectively `SHIPPING`. The verifier must take these key declarations into account, to avoid generating a spurious counter-example in which the tuples retrieved by the service post-conditions are distinct from those retrieved by the property, despite agreeing on product id and shipment type. □

We note right away that one can easily eliminate the global variables $\bar{y}$ of the LTL-FO formula $\forall \bar{y} \varphi_f$.

LEMMA 3.3. *Given Γ and $\forall \bar{y} \varphi_f$ as above, one can construct in linear time an artifact system Γ' such that $\Gamma \models \forall \bar{y} \varphi_f$ iff $\Gamma' \models \varphi_f$.*

Indeed, Γ' is obtained from Γ by simply adding $\bar{y}$ to its artifact variables and propagating their values at each transition. Thus,we can assume that the LTL-FO formulas to be verified have no global variables. Clearly, $\Gamma \models \varphi_f$ iff there is no run of Γ satisfying $\neg \varphi_f$. The verification problem will focus on the latter formulation.

Not surprisingly, model checking is undecidable for artifact systems and LTL-FO properties, even if the system uses only a database (satifying given FDs) and no arithmetic constraints, or only arithmetic constraints and no database.

THEOREM 3.4. *(i) It is undecidable, given an artifact system $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$, a set F of FDs over $\mathcal{DB}$, and an LTL-FO property φ such that φ and all pre-and-post conditions of Σ and Π use only relations in $\mathcal{DB}$, whether φ holds for all runs of Γ on databases satisfying F.*

(ii) It is undecidable, given an artifact system $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$ and an LTL-FO property φ such that φ and all pre-and-post conditions of Σ and Π use only constraints in $\mathcal{C}$, whether $\Gamma \models \varphi$.

Proof: Part (i) follows from Theorem 4.2 in [17]. Part (ii) is shown by defining an artifact system that simulates a counter machine in conjunction with a property that further forbids reachability of a given state of the machine (details ommitted). Since state reachability is undecidable for counter machines [40], the result follows. □

REMARK 3.5. *Note that, in the absence of dependencies and arithmetic, the artifact systems discussed here fall in the class of "guarded" artifact systems introduced in [17] towards decidable static verification. Theorem 3.4 shows that the results of [17] do not transfer to the new setting. As detailed below, overcoming the technical challenges introduced by arithmetic and dependencies requires developing a fundamentally different proof technique and a novel syntactic restriction that yields decidability.*

4. FEEDBACK-FREE ARTIFACT SYSTEMS

We next define the feedback-free syntactic restriction, applying jointly to an artifact system and a property to be verified. The original intuition behind the notion of feedback freedom comes from the proof of Theorem 3.4(ii), which shows that artifact systems can simulate counter machines. Such an artifact system uses a service that performs the increment operation on a counter variable, and allows the run to "feed back" the incremented value into the same service (by updating the same counter variable) for an unbounded number of times. Decrement is handled similarly. This simulation of unbounded counters is responsible for undecidability. The feedback-freedom restriction is designed to limit the data flow between occurrences of the same artifact variable at different times in runs of the system that satisfy the desired property. This precludes the ability to perform the kind of unbounded computations needed to simulate counter machines. The intuition is further discussed after the formal definition of feedback-freedom.

Symbolic runs.

In order to formalize the feedback free condition, we use the notion of *symbolic run*. This will also provide the central component of our verification algorithm presented in the next section.

Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be an artifact system, where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$, and φ_f be an LTL-FO formula with no global variables. Recall that our aim is to develop a procedure for checking whether there exists a run of Γ satisfying $\neg\varphi_f$.

To each $x \in \bar{x}$ we associate 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 run ϱ consists of a sequence $\{\psi_i(\bar{x}_i, \bar{x}_{i+1})\}_{i\geq 0}$ where each $\psi_i(\bar{x}_i, \bar{x}_{i+1})$ is a CQ$^\neg$ formula over $\mathcal{A}$ with free variables among $\bar{x}_i \cup \bar{x}_{i+1}$. 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 propositions in φ, expanded in φ_f via f. As earlier, P denotes the set of propositions in φ.

1. Δ_Σ. For each service $\langle \pi, \psi \rangle \in \Sigma$, consider the formula $\exists \bar{z} \xi$ obtained from $\pi \wedge \psi$ by putting it in prenex form and its quantifier-free body in DNF. For each such formula, Δ_Σ contains all formulas of the form $\exists \bar{z} \xi_d$, where ξ_d is a disjunct of ξ.
2. Δ_{φ_f} containing, 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 $\psi(\bar{x}, \bar{x}')$ 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 FO 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 run is obtained from some symbolic transition template $\psi(\bar{x}, \bar{x}')$ 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 $\bar{x}_0$). We denote by σ_i the truth assignment to the propositions P of φ defined by $\sigma_i(p) = \sigma(f(p))$. We say that the symbolic run $\varrho = \{\psi_i\}_{i\geq 0}$ satisfies $\neg\varphi_f$, denoted $\varrho \models \neg\varphi_f$, iff $\{\sigma_i\}_{i\geq 0}$ satisfies $\neg\varphi$.

To formalize the feedback-free condition, we associate two undirected graphs G_ψ and E_ψ to each symbolic transition template $\psi = \exists \bar{z}(\phi(\bar{x}, \bar{x}', \bar{z}))$. 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 $\bar{x}, \bar{x}'$ of the transitive closure of the graph containing an edge among every two variables occurring together in an atom of ψ, and E_ψ is the restriction to $\bar{x}, \bar{x}'$ 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 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 runs are defined analogously.

Clearly, E^*_ϱ is an equivalence relation on the variables of 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).

Example 4.1 We illustrate symbolic transition templates and the associated graphs $G_\psi, E_\psi, G_\varrho, E_\varrho$ using the artifact system from Example 2.8, and the following property

$$\varphi_f = \mathbf{F}(\texttt{status} = \text{“shipped”} \vee \texttt{status} = \text{“canceled”}).$$

The corresponding Δ_{φ_f} is

$\Delta_{\varphi_f} = \{\texttt{status} = \text{“shipped”} \vee \texttt{status} = \text{“canceled”},$
$\neg(\texttt{status} = \text{“shipped”} \vee \texttt{status} = \text{“canceled”})\}.$

To build Δ_Σ, we need to rewrite the conjunction of pre- and post-condition for each service into prenex DNF, and add each disjunct as a separate formula to Δ_Σ. The pre- and post-conditions of services **choose_product** and **choose_shiptype** are conjunctive, so only trivial prenex normal form rewriting is needed, which we omit. For service **apply_coupon**, we obtain five disjuncts $\bigvee_{i=1}^5 \xi_i$ for the DNF of $\pi \wedge \psi$. We show ξ_2 below; ξ_1 corresponds to the case

when the customer inputs no coupon number, while $\xi_2, \ldots, \xi_5$ correspond to various coupon type combinations (discount, free shipping). ξ_2 is the case of a discount coupon of value v:

ξ_2: $\texttt{prod_id}' = \texttt{prod_id} \wedge \texttt{ship_type}' = \texttt{ship_type}$
$\wedge\ \texttt{status} = \text{“edit_coupon”} \wedge \texttt{status}' = \text{“processing”} \wedge$
$\exists\, t, v, m, s, p, a, w, c, l\ ($
$\texttt{COUPONS}(\texttt{coupon}', t, v, m, s) \wedge \texttt{PRODUCTS}(\texttt{prod_id}, p, a, w)$
$\wedge \texttt{SHIPPING}(\texttt{ship_type}, c, l) \wedge \neg(t = \text{“free_shipping”})$
$\wedge\ p + c \geq m \wedge\ \texttt{amount_owed}' = p + c - v)$

Given the above sets $\Delta_\Sigma, \Delta_{\varphi_f}$, one of the resulting symbolic transition templates is for instance

$$\psi = \xi_2 \wedge \neg(\texttt{status} = \text{“shipped”} \vee \texttt{status} = \text{“canceled”}).$$

Figure 1(a) depicts the graphs G_ψ and E_ψ for all transition templates. In the case of **apply_coupon**, we indicate which disjunct ξ_i is used. The dashed (blue) edges correspond to equality atoms and belong to both E_ψ and G_ψ. Solid (red) edges correspond to relational atoms and belong to G_ψ only.

For instance, in the case of **choose_shiptype**, the dashed edge from prod_id to prod_id′ reflects the fact that the product id remains unchanged, as specified by the corresponding equality atom in the post-condition. The solid edge from prod_id to ship_type reflects the folowing transitive connection between the two artifact variables: prod_id is directly connected to non-artifact variable w by co-occurrence in the PRODUCTS atom; w is directly connected to l via the arithmetic constraint; l is directly connected to ship_type′ via the SHIPPING atom.

Since the status artifact variable does not appear in any constraint with another variable (neither in the services nor in the property), it does not affect the graphs of the symbolic transition templates, and it is dropped to avoid clutter. For the same reason we do not show the entire transitive closure of edges for either the G_ψ and E_ψ graphs.

Other services include **edit_product, edit_shiptype** and **edit_coupon**, whose specification we omit. However, the graphs already show that they perform expected tasks: when the user edits a coupon, product and shipment type are preserved (as depicted by the two dashed edges); when the shipment type is edited, only the product id is preserved, coupon information is forgotten, requiring subsequent re-input once the shipment type is known; finally, when the product id is edited, nothing is preserved, as new shipment type and coupon information will need to be re-input subsequently.

Now consider a run ϱ whose prefix corresponds to the following sequence of events: the customer chooses a product, then a shipment type, then applies a discount coupon, then changes her mind and edits the shipment type, chooses a new shipment type, and applies a discount coupon again. A resulting computation graph G_ϱ and its restriction to equality edges E_ϱ are depicted in Figure 1(b). The parenthesis annotating service **apply_coupon** means that disjunct ξ_2 is used, since it corresponds to a free shipping coupon. Notice that G_ϱ (E_ϱ) is obtained as the concatenation of the corresponding G_ψ (E_ψ) graphs from Figure 1(a). □

We now define feedback-freedom. We assume the notation developed above.

DEFINITION 4.2. (Γ, φ_f) *is* feedback-free *iff for every symbolic run prefix* $\varrho = \{\psi_i\}_{i \leq n}$*, each path from* x_i *to* x_j *in* G_ϱ *contains a node* y *such that* $span([x_i]) \cup span([x_j]) \subseteq span([y])$.

By extension, we say that $(\Gamma, \forall \bar{y} \varphi_f)$ is feedback-free if (Γ', φ_f) is feedback-free, with Γ' obtained from $(\Gamma, \forall \bar{y} \varphi_f)$ as in Lemma 3.3.

Example 4.3 We illustrate some of the checks required to establish feedback freedom for the artifact system in the running example, on the symbolic run ϱ of Example 4.1 (we cannot, of course, enumerate all runs).

Let prod_id play the role of y from Definition 4.2, ship_type that of x, and instants 3 and 7 the roles of i, respectively j.

Consider the graph G_ϱ shown in Figure 1(b), and nodes $\texttt{ship_type}_3$ and $\texttt{ship_type}_7$, corresponding to artifact variable ship_type at instants 3 and 7. Recall that the dashed (blue) edges depict equality atoms, and connected components thereof repesent equivalence classes. We have :

$[\texttt{ship_type}_3] = \{\texttt{ship_type}_3, \texttt{ship_type}_4\}$,
$span([\texttt{ship_type}_3]) = [3, 4]$,
$[\texttt{ship_type}_7] = \{\texttt{ship_type}_7, \texttt{ship_type}_6\}$,
$span([\texttt{ship_type}_7]) = [6, 7]$.

Notice that every path from $\texttt{ship_type}_3$ to $\texttt{ship_type}_7$ must pass through node $\texttt{prod_id}_4$, and that

$$span([\texttt{prod_id}_4]) \supseteq [2, 7] \supseteq span([\texttt{ship_type}_3]) \cup span([\texttt{ship_type}_7]).$$

□

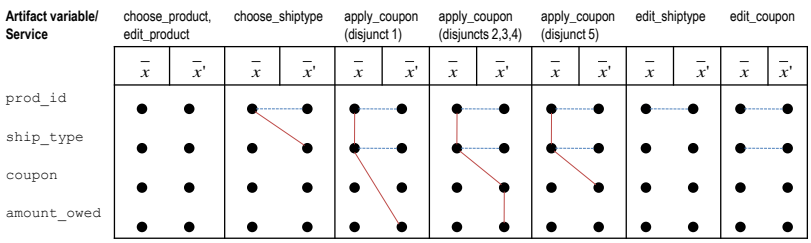
As discussed earlier, feeback-freedom is meant to restrict the ability to perform computation such as needed to simulate a counter machine, requiring repeated increments/decrements of the same variable. This is done by preventing unbounded updates to a variable's current value that depend on its history. Instead, while feedback-free processes still support updating an artifact variable (x) an unbounded number of times, feedback-freedom guarantees that each updated value is independent of how the value of x evolved historically from step i to step j ($i < j$). Indeed, x_j depends only on the values of *other* variables (say y), which are preserved throughout the computation from i to j ($span([y]) \supseteq span([x_i]) \cup span([x_j])$).

Example 4.4 In Example 4.3, the arithmetic constraint satisfied by the shipment type considered at instant 7 does not depend on the previous shipment choice at instant 3, and can be described directly in relation to the product id, which remains constant throughout. This independence would hold even in a run in which the customer repeatedly alternated between making up and changing her mind about the shipment type, possibly re-considering (and again discarding) the same shipment types several times. Similarly, the current balance can be computed directly on the current order snapshot, being independent of the previous ones. □

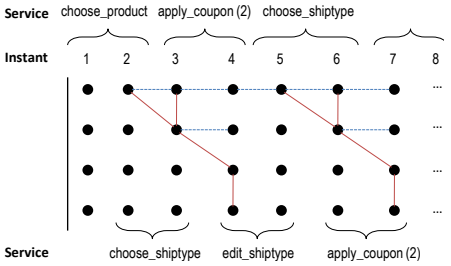
We formalize this intuition in Section 5, where we show that under feedback-freedom it suffices for the verification algorithm to keep only a "compressed" history of bounded size, in form of "inherited constraints" on the artifact variable values at every step.

We claim that the feedback freedom condition is permissive enough to capture a wide class of applications of practical interest. Indeed, this is confirmed by numerous examples of practical business processes modeled as artifact systems, that we encountered due to our collaboration with IBM. Many of these, including typical e-commerce applications, satisfy the feedback freedom condition. The underlying reason seems to be that business processes are usually not meant to "chain" an unbounded number of tasks together, with the output of each task being input to the next. Instead, the unboundedness is usually confined to two forms, both consistent with feeback-freedom:

1. Allowing a certain task to undo and retry an unbounded number of times, with each retrial independent of previous ones, and



(a) Graphs G_ψ and E_ψ for the symbolic transition templates



(b) Computation graph G_ϱ

Figure 1: Graphs in Example 4.1

depending only on a context that remains unchanged throughout the retrial phase. A typical example is repeatedly providing credit card information until the payment goes through, while the order details remain unchanged. Another is the situation in which an order is filled according to sequentially ordered phases, where the customer can repeatedly change her mind within each phase while the input provided in the previous phases remains unchanged (e.g. changing her mind about the shipment type for the same product, the rental car reservation for the same flight, etc.)

2. Allowing a task to batch-process an unbounded collection of inputs, each processed independently, within an otherwise unchanged context (e.g. sending invitations to an event to all attendants on the list, for the same event details).

Testing feedback-freedom We can show that testing feedback-freedom of (Γ, φ_f) can be done in PSPACE. The proof reduces feedback-freedom to testing emptiness of a two-way alternating finite-state automaton, which is in turn reduced to emptiness of a non-deterministic automaton $A_{(\Gamma,\varphi_f)}$.

THEOREM 4.5. *Feedback-freedom of (Γ, φ_f) can be checked in* PSPACE.

REMARK 4.6. *The automata-theoretic approach to testing feedback-freedom can be used to refine the notion of feedback-free by taking into account additional restrictions on the allowed runs of the artifact system. For example, if additional control is specified by a Büchi automaton $\mathcal{B}$, testing feedback-freedom for runs satisfying the additional control reduces to testing emptiness of the cross-product automaton $\mathcal{B} \times A_{(\Gamma,\varphi_f)}$ (with $A_{(\Gamma,\varphi_f)}$ easily turned first into a Büchi automaton).*

5. VERIFICATION OF FEEDBACK-FREE SYSTEMS

The main result of this section is that model checking LTL-FO properties is decidable if the artifact system together with the property are feedback-free. In Section 6, we extend this result to the presence of integrity constraints on database relations.

THEOREM 5.1. *It is decidable, given an artifact system Γ and an LTL-FO formula $\forall \bar{y} \varphi_f$ such that $(\Gamma, \forall \bar{y} \varphi_f)$ is feedback-free, whether $\forall \bar{y} \varphi_f$ holds for every run ρ of Γ.*

The proof requires developing some technical machinery, and is outlined in the remainder of the section. We ignore data dependencies for now, and later show how to take them into account.

Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ where $\mathcal{A} = \langle \bar{x}, \mathcal{DB} \rangle$. Recall that, by Lemma 3.3, one can eliminate the global variables $\bar{y}$ of the LTL-FO formula $\forall \bar{y} \varphi_f$. Thus, we can assume the LTL-FO formula to be verified is simply φ_f. Clearly, $\Gamma \models \varphi_f$ iff there is no run of Γ that satisfies $\neg \varphi_f$. We will prove the theorem by showing decidability of the latter property.

The verification algorithm makes use of the symbolic runs introduced earlier. We claim that symbolic runs provide a representation of all actual runs of an artifact system. Let $\varrho = \{\psi_i\}_{i \geq 0}$ be a symbolic run of Γ. To each such symbolic run and each database instance D we associate a set of actual runs on D as follows. Let $var(\varrho) = \{\bar{x}_i \mid i \geq 0\}$ and $\Delta_\varrho = \{\psi_i | i \geq 0\}$. Note that the set of free variables of formulas in Δ_ϱ is $var(\varrho)$. Let

$$Runs_D(\varrho) = \{\{\nu(\bar{x}_i)\}_{i \geq 0} \mid \nu \text{ is a valuation of } var(\varrho) \text{ into } \mathbb{Q} \text{ such that } D \cup \mathcal{C} \models \Delta_\varrho\}$$

We say that ϱ is *satisfiable* if there exists a database instance D such that $Runs_D(\varrho) \neq \emptyset$. We use analogous defnitions and notation for the case when ϱ is a *prefix* of a symbolic run. In particular, for $j > 0$, we denote by $\varrho|_j$ the prefix $\{\psi_i\}_{i<j}$ of ϱ and refer to $Runs_D(\varrho|_j)$ with the obvious meaning.

The following establishes the desired connection between symbolic runs and actual runs.

LEMMA 5.2. *(i) For each database instance D, $Runs_D(\Gamma) = \cup\{Runs_D(\varrho) \mid \varrho \text{ is a symbolic run of } \Gamma\}$. (ii) There exists a run of Γ satisfying $\neg\varphi_f$ iff there exists a satisfiable symbolic run of Γ satisfying $\neg\varphi_f$.*

In view of the above, it is sufficient to check the existence of a satisfiable symbolic run of Γ satisfying $\neg\varphi_f$. To prove decidability, we show that it is enough to consider prefixes of symbolic runs of statically bounded length.

Consider a symbolic run $\{\psi_i\}_{i \geq 0}$. We define for each $j > 0$ the *inherited constraint* of configuration j, denoted η_j, which summarizes the constraints on configuration j imposed by the prefix leading to it, i.e. $\{\psi_i\}_{i<j}$. The inherited constraint $\eta_j(\bar{x}_j)$ is defined as $\exists \bar{z} \bigwedge_{i<j} \psi_i$ where $\bar{z}$ are all variables other than $\bar{x}_j$. The following is immediate from the definitions.

LEMMA 5.3. *Let $\varrho = \{\psi_i\}_{i \geq 0}$ be a symbolic run and D a database instance. Then for every $j > 0$,*

$$\{\rho_j \mid \{\rho_i\}_{0 \leq i \leq j} \in Runs_D(\varrho|_j)\} = \{\rho_j \mid D \cup \mathcal{C} \models \eta_j(\rho_j)\}.$$

In other words, η_j defines precisely the set of valuations of $\bar{x}$ reachable in the last configuration of a run in $Runs_D(\varrho|_j)$. Note that this is generally a strict superset of $\{\rho_j \mid \{\rho_i\}_{i \geq 0} \in Runs_D(\varrho)\}$.

We next show the key fact that for a feedback-free system there are only finitely many inherited constraints up to logical equivalence. This is done by rewriting each such constraint as a $CQ^\neg$ formula of bounded quantifier rank. Recall that the quantifier rank of a formula is the maximum number of quantifiers along a path from root to leaf in the syntax tree of the formula, and that there are

finitely many non-equivalent formulas of given quantifier rank over a given vocabulary (e.g., see [35]). The number of non-equivalent formulas is hyperexponential in the quantifier rank.

LEMMA 5.4. *For each η_j one can construct an equivalent CQ$^\neg$ formula $\bar{\eta}_j$ of quantifier rank bounded by $k^2 + q$, where $k = |\bar{x}|$ and q is the maximum quantifier rank of the formulas used in pre-and-post conditions of services in Σ.*

The proof (omitted) shows how to rewrite η_j as a formula η_j^* of quantifier rank at most $k^2 + q$, whose variables are the equivalence classes of variables in $\varrho|_j$ induced by the equality graph $E^*_{\varrho|_j}$. The construction of η_j^* is non-deterministic, so several outcomes are possible, all with the same quantifier rank. Finally, $\bar{\eta}_j$ is obtained from η_j^* by replacing its free variables with $\bar{x}_j$. We will refer to η_j^* in the sequel.

Reduced inherited constraints It is well known that the number of formulas of given quantifier rank is finite, up to logical equivalence. The notion of equivalence is in fact a strong syntactic one, upon which we elaborate next. For every CQ$^\neg$ formula α, we can define a reduction procedure yielding a logically equivalent formula *red*(α), obtained essentially by recursively merging isomorphic subformulas. It can be seen that the number of distinct reduced formulas of given quantifier rank d is bounded by a hyperexponential in d. This also yields our upper bound for inherited constraints.

We denote by Hyp the class of hyperexponential functions. Each function in Hyp is defined inductively by $hyp(0) = 1$ and $hyp(n+1) = 2^{c \cdot hyp(n)}$ for some constant c.

LEMMA 5.5. *Let Γ and $\forall \bar{y} \varphi_f$ be as above. The number of distinct reduced inherited constraints in configurations of symbolic runs is bounded by $h(k^2)$ for some $h \in Hyp$.*

Symbolic lassos We next use the finiteness of the reduced inherited constraints to characterize the existence of symbolic runs satisfying $\neg\varphi_f$ using finite prefixes of a certain form, which we call *symbolic lassos*. Let $B_{\neg\varphi}$ be the Büchi automaton corresponding to $\neg\varphi$. Recall that, for a symbolic run $\{\psi_i\}_{i\geq 0}$, we denote by $\{\sigma_i\}_{i\geq 0}$ the sequence of truth assignments to propositions P in φ such that $\sigma_i(p)$ holds iff $\varphi_i \models f(p)$. A run of $B_{\neg\varphi}$ on $\{\sigma_i\}_{i\geq 0}$ is a sequence $\{q_i\}_{i\geq 0}$ of states of $B_{\neg\varphi}$ such that $(init, \sigma_0, q_0)$ is a transition of $B_{\neg\varphi}$ for some initial state *init* and $(q_i, \sigma_{i+1}, q_{i+1})$ is a transition of $B_{\neg\varphi}$ for each $i \geq 0$. A similar definition of run applies to finite sequences $\{\sigma_i\}_{i\leq l}$.

DEFINITION 5.6. *A* symbolic lasso *is a finite prefix $\{\psi_i\}_{i<j+n}$ of a symbolic run such that:*

- $red(\eta_j^*) = red(\eta_{j+n}^*)$,
- *for each $u, v \in \bar{x}$, $[u_j] = [v_j]$ iff $[u_{j+n}] = [v_{j+n}]$,*
- *for each $u \in \bar{x}$, $[u_j] = [u_{j+n}]$ or $[u_j] \neq [v_{j+n}]$ for each $v \in \bar{x}$,*
- *there exists a run $\{q_i\}_{i\leq j+n}$ of $B_{\neg\varphi}$ on $\{\sigma_i\}_{i\leq j+n}$ such that for some accepting state r, $q_j = q_{j+n} = r$.*

We can show the following.

LEMMA 5.7. *There exists a run of Γ satisfying $\neg\varphi_f$ iff there exists a satisfiable symbolic lasso.*

Decision procedure The above development provides a decision procedure for satisfaction of LTL-FO properties of feedback-free systems, which we outline next. Recall that the LTL-FO property can be assumed to have no global variables by Lemma 3.3. The input to the algorithm is an artifact system $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ and LTL-FO property φ_f over $\mathcal{A}$ such that (Γ, φ_f) is feedback-free. We begin by constructing the sets of formulas $\Delta = \Delta_\Sigma \cup \Delta_{\varphi_f}$ and the Büchi automaton $B_{\neg\varphi}$. We use a procedure Büchi-Next which, given a state p of $B_{\neg\varphi}$ and a truth assignment σ to the propositions of φ returns one next state of $B_{\neg\varphi}$.

The algorithm non-deterministically searches for a satisfiable symbolic lasso as follows:

1. flag := 0;
2. initialize a symbolic run prefix ϱ to $\{\psi_0\}$, with corresponding truth assignment σ_0 to the propositions of φ and set s to some output of Büchi-Next(q_0, σ_0) for some initial state q_0 of $B_{\neg\varphi}$;
3. set γ to $red(\eta^*(\varrho))$, where $\eta(\varrho)$ is the inherited constraint for the prefix ϱ;
4. initialize the set $\mathcal{R}$ of reduced configurations to $\{(s, \gamma)\}$;
5. if flag = 0 and s is an accepting state of $B_{\neg\varphi}$ then non-deterministically continue or set $(\bar{s}, \bar{\gamma}) := (s, \gamma)$, flag:= 1, and $\mathcal{R} := \emptyset$;
6. non-deterministically generate from Δ a symbolic transition ψ with corresponding truth assignment σ to the propositions in φ;
7. set s to Büchi-Next(s, σ), $\gamma := red(\eta^*(\varrho\psi))$, and $\varrho := \varrho.\psi$;
8. if $(s, \gamma) \in \mathcal{R}$ then output NO and stop; otherwise, set $\mathcal{R} := \mathcal{R} \cup \{(s, \gamma)\}$;
9. if flag = 1, $(s, \gamma) = (\bar{s}, \bar{\gamma})$, and $\bar{\gamma}$ is satisfiable, output YES and stop. Otherwise, go to 5.

To see that this provides a decision procedure, we need to show that (i) it terminates and provides the correct answer (i.e. it outputs YES on some execution on (Γ, φ_f) iff $\Gamma \models \varphi_f$), and (ii) the satisfiability test in step 9 is effective.

To see (i), note that, from the definition of inherited constraint $\eta(\varrho)$ and the equivalence with $red(\eta^*(\varrho))$, it follows that:

(§) if ϱ_1, ϱ_2 are satisfiable prefixes of symbolic runs such that $red(\eta^*(\varrho_1)) = red(\eta^*(\varrho_2))$ and ψ is a symbolic transition, then
- $\varrho_1.\psi$ is satisfiable iff $\varrho_2.\psi$ is satisfiable, and
- the sets of non-deterministically constructed $red(\eta^*(\varrho_1.\psi))$ and $red(\eta^*(\varrho_2.\psi))$ are equal.

This means that the search for a symbolic lasso can be confined to prefixes with no repeated configurations (s, γ) apart from the knot $(\bar{s}, \bar{\gamma})$, which is enforced by step 8. Since there are finitely many reduced inherited constraints for symbolic runs of (Γ, φ_f) this bounds the running time of the above procedure.

For (ii), we discuss the procedure for checking satisfiability of reduced inherited constraints.

We show that satisfiability of a CQ$^\neg$ formula over $\mathcal{DB} \cup \mathcal{C}$ can be decided in a modular fashion, by independently checking satisfiability of formulas over $\mathcal{DB}$ and over $\mathcal{C}$. The significance of the result is that it enables a generic model checking algorithm that takes as parameter the fixed interpretation of $\mathcal{C}$, as long as it comes with a domain-specific satisfiability checker $SAT_\mathcal{C}$. $SAT_\mathcal{C}$ is called as a black box by the model checker.

More precisely, let q be a prenex normal form CQ$^\neg$ formula over $\mathcal{DB} \cup \mathcal{C}$: $q = \exists \bar{z} \xi(\bar{u})$ where ξ is quantifier-free, of free variables $\bar{u}$ ($\bar{z} \subseteq \bar{u}$). Let $\xi|_{\mathcal{DB}}$ and $\xi|_\mathcal{C}$ be the restrictions of ξ to schemas $\mathcal{DB}$ and $\mathcal{C}$, respectively, and denote with $\bar{y} = vars(\xi|_{\mathcal{DB}}) \cap vars(\xi|_\mathcal{C})$ the variables they have in common. Denote with $\bar{c}$ all constants appearing in $\xi|_{\mathcal{DB}}$. Recall that an equality type $eq(\bar{t})$ over a set $\bar{t}$ of

terms (variables or constants) is a satisfiable conjunction of equality and non-equality atoms over $\bar{t}$ such that every pair of terms from $\bar{t}$ occurs in some atom of eq. The following claim follows immediately from the preservation of $CQ^{\neg}$ formulas under isomorphisms:

($\ddagger$) q is satisfiable if and only if there exists an equality type $eq(\bar{y}, \bar{c})$, such that $\xi|_{\mathcal{C}} \wedge eq(\bar{y}, \bar{c})$ and $\xi|_{\mathcal{DB}} \wedge eq(\bar{y}, \bar{c})$ are satisfiable.

Notice that the satisfiability check for $\xi|_{\mathcal{C}} \wedge eq(\bar{y}, \bar{c})$ in the claim is domain-specific (i.e. depends on the fixed interpretation of $\mathcal{C}$), being settled by calling $SAT_{\mathcal{C}}$. Also recall that $\xi|_{\mathcal{DB}} \wedge eq(\bar{y}, \bar{c})$ is an existentially-quantified formula. Therefore its satisfiability reduces to checking that: (a) no pair of terms appears both in an equality and a non-equality atom, and (b) no tuple of terms appears both in a positive and a negative literal with the same relational symbol. This establishes decidability of verification for LTL-FO properties of feedback-free systems, completing the proof of Theorem 5.1.

5.1 Complexity

The complexity analysis of the decision procedure involves several orthogonal components:

Computing reduced inherited constraints This involves the recursive construction in the proof of Lemma 5.4. It is easily seen that it requires polynomial time in the size of inherited constraint γ, which is bounded by the size of the the symbolic run prefix ϱ.

Performing the satisfiability check (in step 9 of the decision procedure), which consists in

(i) Retrieving the prenex normal form of $\bar{\gamma}$, which is $\eta(\varrho)$, then guessing an equality type eq on the number of variables $\eta(\varrho)|_{\mathcal{C}}$ and $\eta(\varrho)|_{\mathcal{DB}}$ have in common, and the number of constants mentioned in $\eta(\varrho)|_{\mathcal{DB}}$. This can be done in NP in the number of common variables and of constants, which is bounded by the size of $\eta(\varrho)$.

(ii) Running $SAT_{\mathcal{C}}$ on $\eta(\varrho)|_{\mathcal{C}} \wedge eq$. This step depends on the fixed interpretation of $\mathcal{C}$. If $\mathcal{C}$ is interpreted as linear arithmetic inequalities, the test reduces to solving a linear programming problem. This yields polynomial time in the size of $\eta(\varrho)|_{\mathcal{C}} \wedge eq$ [30], which in turn is polynomially (quadratically) bounded by the size of $\eta(\varrho)$. Indeed, each pair of terms in $\eta(\varrho)|_{\mathcal{C}}$ must be related explicitly in eq by either an equality or a non-equality atom.

(iii) Checking satisfiability of $\eta(\varrho)|_{\mathcal{DB}} \wedge eq$. This is doable in polynomial time in the size of $\eta(\varrho)|_{\mathcal{DB}} \wedge eq$, which is again polynomially bounded by the size of $\eta(\varrho)$.

The search for the symbolic lasso This step is polynomial in the number of reduced inherited constraints and the states of $B_{\neg\varphi}$ visited during the search. The test that the current inherited constraint γ is the same as the knot candidate $\bar{\gamma}$ is polynomial in the size of $\bar{\gamma}$.

Since the size of reduced inherited constraints γ is upper bounded by the length of the run ϱ, which in turn is bounded by the number of distinct reduced inherited constraints, by Lemma 5.4 we obtain:

PROPOSITION 5.8. *Static verification for feedback-free pairs of LTL-FO properties and artifact systems is decidable in time upper bounded by $h(k^2)$, for some $h \in Hyp$.*

5.2 Subclasses with Improved Upper Bounds

The above analysis shows that the complexity of the decision procedure is dominated by the number of distinct inherited constraints, which upper bounds the symbolic run length. We identify next three sub-classes of feedback-free artifact systems that occur naturally and lead to a better bound.

Bounded width The construction in the proof of Lemma 5.4 introduces the useful notion of *width of a set of variables* in a symbolic run. Recall that the width $w(\bar{y})$ of $\bar{y}$ is defined as $max\{|\bar{v}| \mid \bar{v} \subseteq \bar{x}_i$, and each $v \in \bar{v}$ is in an equivalence class $y \in \bar{y}\}$. By extension, the width of a subgraph of G_j is the width of its set of nodes.

DEFINITION 5.9. *We say that (Γ, φ_f) has width bounded by w if it is feedback-free, and if for each symbolic run, the width of each connected component of G_j is bounded by w for each $j \geq 0$.*

Intuitively, the width bound indicates the maximum number of variables in $\bar{x}$ that are mutually related in a configuration of a symbolic run. We can show the following.

COROLLARY 5.10. *If (Γ, φ_f) has width bounded by w, then the number of distinct reduced inherited constraints is bounded by $(h(w^2))^k$, where $h \in Hyp$.*

Proof: The bound is an immediate consequence of the construction in the proof of Lemma 5.4, and the fact that constraints corresponding to distinct connected components of G_j are independent. □

Thus, Corollary 5.10 provides a smaller bound on the number of reduced inherited constraints if the connected components generated in symbolic runs have small width.

Linear propagation An artifact system and a property exhibit *linear propagation* if the feedback-freedom restriction is satisfied for a more restrictive definition of variable equivalence classes. Equivalence classes are generated exclusively by equalities of the form $x = x'$, and any other equalities are treated as arithmetic constraints.

Note that every linear-propagation equivalence class involves the values of a *single* variable. In the graphical representation of symbolic transition templates and runs, all equality edges are horizontal. This is the case in our running example.

PROPOSITION 5.11. *Let (Γ, φ_f) exhibit linear propagation. The number of distinct reduced inherited constraints in configurations of symbolic runs is bounded by $h(k)$, for some function $h \in Hyp$.*

COROLLARY 5.12. *If linear-propagation pair (Γ, φ_f) has width bounded by w, then there are at most $(h(w))^k$ distinct reduced inherited constraints, for some $h \in Hyp$.*

Proof: The proof follows immediately from Proposition 5.11 and the observation that constraints corresponding to distinct connected components are independent. □

Acyclicity Recall that feedback-freedom restricts the way in which the value of variable x at step j can be connected to the value of x at preceding step i. We investigate a more stringent restriction, which disallows any such connection (except for preservation of the value of x from i to j).

DEFINITION 5.13. *(Γ, φ_f) is* acyclic *iff for every symbolic run prefix $\varrho = \{\psi_i\}_{i \leq n}$, if x_i and x_j are connected in G_ϱ, then $[x_i] = [x_j]$.*

Note that acyclicity trivially implies feedback-freedom: in Definition 4.2, the role of y is played by x_i.

PROPOSITION 5.14. *Let (Γ, φ_f) be acyclic. The number of distinct reduced inherited constraints in configurations of the same symbolic run is bounded by a doubly-exponential function of k.*

To illustrate the difference between acyclicity and feedback freedom, consider again our running example. As discussed earlier, feedback freedom allows changing the shipment type unboundedly many times for the same product. In contrast, acyclicity disallows such runs. If the customer wants to change the shipment type, she must forget all her choices and starts filling the order from scratch (select a product again, then a shipment type). This puts the two shipment type choices in disconnected components of the computation graph.

6. VERIFICATION WITH DEPENDENCIES

We show next that model checking for feedback-free pairs of LTL-FO properties and artifact systems is decidable even in the presence of expressive database integrity constraints modeled by dependencies.

Given a set $\mathcal{I}$ of dependencies on the database schema $\mathcal{DB}$, we say that an artifact system Γ satisfies an LTL-FO sentence φ *under* $\mathcal{I}$, denoted $\Gamma \models_{\mathcal{I}} \varphi$, if for every database D satisfying $\mathcal{I}$ and every run ρ of Γ on D, φ holds on ρ.

We next establish decidability under a set of dependencies, provided that the chase with these dependencies terminates. The chase is a fundamental algorithm that has been widely used in databases. It takes as input an initial instance A and a set of dependencies $\mathcal{I}$ and produces (if it terminates, which is not guaranteed), a finite model of $\mathcal{I}$ and A that satisfies a universality property (see [2] for details).

We borrow from [39] the notation $CT_{\forall\exists}$ for the class of dependency sets $\mathcal{I}$ such that for every instance A there exists a terminating chase sequence of A with $\mathcal{I}$.

THEOREM 6.1. *It is decidable, given artifact system Γ and an LTL-FO sentence $\forall\bar{y}\varphi_f$ such that $(\Gamma, \forall\bar{y}\varphi_f)$ is feedback-free, and given set $\mathcal{I} \in CT_{\forall\exists}$ of dependencies on $\mathcal{DB}$, whether Γ satisfies $\forall\bar{y}\varphi_f$ under $\mathcal{I}$.*

While membership of a set of dependencies in $CT_{\forall\exists}$ is in general known to be undecidable (see for instance [18]), recent research has proposed sufficient syntactic restrictions. Examples include *weak acyclicity* [23], stratification [18], and the *termination hierarchy* [39] which is a hierarchy of successive relaxations of weak acyclicity and stratification that nevertheless suffice for the existence of a terminating chase sequence.

COROLLARY 6.2. *If $\mathcal{I}$ lies in the terminaton hierarchy and $(\Gamma, \forall\bar{y}\varphi_f)$ is feedback-free, then $\Gamma \models_{\mathcal{I}} \forall\bar{y}\varphi_f$ is decidable.*

The proof of Theorem 6.1 is given after introducing a few useful definitions and results.

Let $q(\bar{u})$ be a CQ$^\neg$ formula over $\mathcal{DB} \cup \mathcal{C}$ with free variables $\bar{u}$. We say that q is *$\mathcal{I}$-satisfiable* if there exists $D \models \mathcal{I}$ and a valuation ν of $\bar{u}$ such that $D \cup \mathcal{C} \models q(\nu)$. We say that symbolic run ϱ is *$\mathcal{I}$-satisfiable* if there exists $D \models \mathcal{I}$ such that $Runs_D(\varrho) \neq \emptyset$. The definition extends naturally to prefixes of symbolic runs, and in particular to symbolic lassos.

Decision procedure The decision procedure we exhibit in proving Theorem 6.1 is the one presented in Section 5 for the dependency-free case, modified as follows: in step 9, the test of satisfiability of $\bar{\gamma}$ is replaced with an $\mathcal{I}$-satisfiability test.

The remainder of the section outlines the proof that the above modification yields a decision procedure for model checking under dependencies, provided that the set of dependencies lies in the termination hierarchy.

We extend Claim (‡) from Section 5 to the presence of dependencies. We first show that $\mathcal{I}$-satisfiability of a CQ$^\neg$ formula over $\mathcal{DB} \cup \mathcal{C}$ can be decided in a modular fashion, by independently checking satisfiability of formulas over $\mathcal{DB}$ and over $\mathcal{C}$.

More precisely, let q be a prenex normal form CQ$^\neg$ formula over $\mathcal{DB} \cup \mathcal{C}$: $q = \exists\bar{z}\xi(\bar{u})$ where ξ is quantifier-free, of free variables $\bar{u}$ ($\bar{z} \subseteq \bar{u}$). Let $\xi|_{\mathcal{DB}}$ and $\xi|_{\mathcal{C}}$ be the restrictions of ξ to schemas $\mathcal{DB}$ and $\mathcal{C}$, respectively, and denote with $\bar{y} = vars(\xi|_{\mathcal{DB}}) \cap vars(\xi|_{\mathcal{C}})$ the variables they have in common. Denote with $\bar{c}$ all constants appearing in $\xi|_{\mathcal{DB}}$ or $\mathcal{I}$.

LEMMA 6.3. *q is $\mathcal{I}$-satisfiable if and only if there exists an equality type $eq(\bar{y}, \bar{c})$, such that $\xi|_{\mathcal{C}} \wedge eq(\bar{y}, \bar{c})$ is satisfiable and $\xi|_{\mathcal{DB}} \wedge eq(\bar{y}, \bar{c})$ is $\mathcal{I}$-satisfiable.*

As observed in Section 5, the satisfiability check for $\xi|_{\mathcal{C}} \wedge eq(\bar{y}, \bar{c})$ in Lemma 6.3 is domain-specific (i.e. depends on the fixed interpretation of $\mathcal{C}$), being settled by calling $SAT_{\mathcal{C}}$.

We next show that $\mathcal{I}$-satisfiability of formulas over $\mathcal{DB}$ reduces to chasing with $\mathcal{I}$ and an appropriately selected set $\mathcal{I}_{\mathcal{DB}}$ of dependencies whose construction is determined by the schema $\mathcal{DB}$.

We recall from [18] an extension of the chase to *disjunctive embedded dependencies (DEDs)*. Here, we are only interested in the particular case when the DED conclusion consists of the empty (always false) disjunction $\bot$. As soon as a chase step derives $\bot$, the chase sequence terminates, yielding the result $\bot$. We then say that the chase *fails*. A terminating chase sequence is a finite sequence of chase steps which either fails or yields an instance that satisfies all dependencies.

Define the set of dependencies

$$\mathcal{I}_{\mathcal{DB}} := \{\delta_{\neq}\} \cup \{\delta_{\neg}^P \mid P \in \mathcal{DB}\}$$

where

$$\delta_{\neq} \quad : \quad \forall x \forall y\, x = y \wedge x \neq y \rightarrow \bot$$
$$\delta_{\neg}^P \quad : \quad \forall \bar{x}\, P(\bar{x}) \wedge \neg P(\bar{x}) \rightarrow \bot.$$

LEMMA 6.4. *If $\mathcal{I} \in CT_{\forall\exists}$, then*

(i) $\mathcal{I} \cup \mathcal{I}_{\mathcal{DB}} \in CT_{\forall\exists}$, and

(ii) a formula $q \in$ CQ$^\neg$ over $\mathcal{DB}$ is $\mathcal{I}$-satisfiable if and only if the chase of q with $\mathcal{I} \cup \mathcal{I}_{\mathcal{DB}}$ does not fail.

REMARK 6.5. *We could have applied the more general decision procedure from [18], for satisfiability of CQ$^\neg$ under a set of dependencies with negated literals. The result in [18] is based on chasing with more expressive, disjunctive dependencies, yielding a tree of chase sequences. When applied to our setting, this would result in an exponential number of chase sequences. Lemma 6.4 shows that this exponential blow-up can be avoided by resorting to the standard (non-disjunctive) chase, and by exploiting the fact that the dependencies in $\mathcal{I}$ contain only positive literals.*

The following result establishes the decidability of $\mathcal{I}$-satisfiability for finite prefixes of symbolic runs.

LEMMA 6.6. *It is decidable, given a symbolic run prefix ϱ and $\mathcal{I} \in CT_{\forall\exists}$ on $\mathcal{DB}$, whether ϱ is $\mathcal{I}$-satisfiable.*

Proof: Lemmas 5.3, 6.3 and 6.4 imply that the following is a decision procedure for $\mathcal{I}$-satisfiability of ϱ. Let η_n be the inherited constraint for configuration n of ϱ (in prenex normal form), and ξ be its quantifier-free body. Let $\xi|_{\mathcal{DB}}$, $\xi|_{\mathcal{C}}$, $\bar{y}$, $\bar{c}$, eq be as in Lemma 6.3, and $\mathcal{I}_{\mathcal{DB}}$ as in Lemma 6.4. Return YES if and only if $SAT_{\mathcal{C}}$ returns YES on $\xi|_{\mathcal{C}} \wedge eq$ and the chase of $\xi|_{\mathcal{DB}} \wedge eq$ with $\mathcal{I} \cup \mathcal{I}_{\mathcal{DB}}$ does not fail. □

Complexity The complexity upper bound obained in the absence of dependencies is virtually unaffected by the presence of sets of dependencies from the termination hierarchy.

First, recall that the satisfiability check for the restiction of the inherited constraints to $\mathcal{C}$ is settled by calling $SAT_{\mathcal{C}}$, thus inheriting the complexity of the particular instantiation of $\mathcal{C}$.

Second, the complexity of the $\mathcal{I}$-satisfiability check for the inherited constraints restricted to $\mathcal{DB}$ inherits the complexity of the chase with sets of dependencies from the termination hierarchy.

This complexity is polynomial in the size of the inherited constraint, with the polynomial's degree bounded by the size of $\mathcal{I}$ [39, 18, 23]. The bound is very conservative, and a refined analysis shows that it depends on the longest path in a graph that reflects how a chase step with one dependency can trigger another.

Given that we expect multiple verification instances over the same database schema with integrity constraints, a reasonable assumption (often adopted in the literature) is to consider schema and dependencies fixed. This yields a truly polynomial complexity of the chase. The same truly polynomial complexity holds, even if $\mathcal{DB}$ and $\mathcal{I}$ are not fixed, if the dependencies in $\mathcal{I}$ are *equality-generating dependencies* [2]. It also holds if only $\mathcal{DB}$ is fixed but not $\mathcal{I}$, if it consists only of *full* dependencies [2]. Equality-generating dependencies allow only equality atoms in the conclusion, and capture as particular cases the class of functional dependencies. Full dependencies contain no existentially quantified variables.

7. RELATED WORK

Data-aware business process models The specific notion of artifact was first introduced in [44] and was further studied, from both practical and theoretical perspectives, in [4, 5, 24, 25, 6, 36, 31, 33, 21]. Some key roots of the artifact model are present in "adaptive objects"[32], "adaptive business objects" [41], "business entities", "document-driven" workflow [47] and "document" engineering [26]. The Vortex framework [28, 20, 27] also allows the specification of database manipulations and provides declarative specifications for when services are applicable to a given artifact. The artifact model considered here is closely related to that of semantic web services in general. In particular, the OWL-S proposal [38, 37] describes the semantics of services with input, output, pre- and post-conditions.

Static analysis of data-aware business processes Work on formal analysis of artifact-based business processes in restricted contexts has been reported in [24, 25, 6]. Properties investigated include reachability [24, 25], general temporal constraints [25], and the existence of complete execution or dead end [6]. For the variants considered in each paper, verification is generally undecidable; decidability results were obtained only under rather severe restrictions, e.g., restricting all pre-conditions to be "true" [24], restricting to bounded domains [25, 6], or restricting the pre- and post-conditions to refer only to artifacts (and not their variable values) [25]. None of the above papers permit an underlying database, integrity constraints, or arithmetic.

[14] adopts an artifact model variation with arithmetic operations but no database (and therefore no integrity constraints). It proposes a criterion for comparing the expressiveness of specifications using the notion of *dominance*, based on the input/output pairs of business processes. Decidability is shown only by restricting runs to bounded length. [48] addresses the problem of the existence of a run that satisfies a temporal property, for a restricted case with no database, no arithmetic, and only propositional LTL properties.

Static analysis for semantic web services is considered in [42], but in a context restricted to finite domains.

More recently, [3] has studied automatic verification in the context of business processes based on Active XML documents.

Our work is most closely related to the one in [17], which identifies the class of *guarded* artifact systems and LTL-FO properties, for which verification is decidable. The two settings have in common the underlying database and the infinite data domain with a dense linear order, as well as the syntax for pre-, post-conditions, and properties. However, [17] considers no dependencies and no arithmetic operations. Our previous results do not apply in the new context, as Theorem 3.4 shows undecidability even when adding to a guarded artifact system a single functional dependency, or alternately, when the only allowed arithmetic operation consists in incrementing counters. The novel proof technique based on describing configuratons using inherited constraints is fundamentally different from the one employed for guardedness.

The works [19, 46, 1] are ancestors of [17] from the context of verification of electronic commerce applications. Their models could conceptually (if not naturally) be encoded as artifact systems, but they correspond only to particular cases of the model in [17]. They all disallow the linear order on the domain. Also, limit artifact values to essentially come from the active domain of the database, thus ruling out external inputs, partially-specified services, and arithmetic.

Infinite-state systems We expect our results to be of interest to the verification community at large, since artifact systems are a particular case of infinite-state systems. Research on automatic verification of infinite-state systems has recently focused on extending classical model checking techniques (e.g., see [13] for a survey). However, in much of this work the emphasis is on studying recursive control rather than data, which is either ignored or finitely abstracted. More recent work has been focusing specifically on data as a source of infinity. This includes augmenting recursive procedures with integer parameters [9], rewriting systems with data [10, 8], Petri nets with data associated to tokens [34], automata and logics over infinite alphabets [12, 11, 43, 15, 29, 7, 8], and temporal logics manipulating data [15, 16]. However, the restricted use of data and the particular properties verified have limited applicability to the business artifacts setting.

8. CONCLUSIONS

We identified the practically significant class of feedback-free business artifact systems for which verification of useful temporal properties is decidable. This alleviates limitations of our previous results on guarded artifacts, which disallow data dependencies and arithmetic, nonetheless essential in practical business processes. Our new results required developing technical machinery that is entirely different from the one we used for guarded artifact systems.

For the moment, we were only able to prove a hyperexponential upper bound on the complexity of verification, with improved bounds for several restricted but realistic cases. In the absence of lower bounds, there is hope that better upper bounds can be obtained. More importantly, it appears that practical specifications often obey restrictions leading to drastically lower complexity. This can be effectively exploited through appropriate heuristics. We plan to further explore the practical potential of our approach using real-world business artifact specifications made available through

our collaboration with IBM. More broadly, feedback-freedom and acyclicity are interesting data-flow notions in their own right, that may be fruitfully used with any data-aware business process model, beyond artifact systems.

9. REFERENCES

[1] S. Abiteboul, V. Vianu, B.S. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *JCSS*, 61(2):236–269, 2000. Extended abstract in PODS 98.

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

[3] Serge Abiteboul, Luc Segoufin, and Victor Vianu. Static analysis of active xml systems. *ACM Trans. Database Syst.*, 34(4), 2009.

[4] K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F. Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4):703–721, 2007.

[5] K. Bhattacharya et al. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Systems Journal*, 44(1):145–162, 2005.

[6] K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *Proc. Int. Conf. on Business Process Management (BPM)*, 2007.

[7] Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *LICS*, pages 7–16, 2006.

[8] A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *FCT'07*, volume 4639 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007.

[9] A. Bouajjani, P. Habermehl, and R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 295:85–106, 2003.

[10] A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS'07*, volume 4424 of *Lecture Notes in Computer Science*, pages 690–705. Springer, 2007.

[11] P. Bouyer. A logical characterization of data languages. *Information Processing Letters*, 84(2):75–85, 2002.

[12] P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2):137–162, 2003.

[13] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification of infinite structures. In *Handbook of Process Algebra*, pages 545–623. Elsevier Science, 2001.

[14] Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Jianwen Su. Artifact-centric workflow dominance. In *ICSOC/ServiceWave*, pages 130–143, 2009.

[15] Stéphane Demri and Ranko Lazić. LTL with the Freeze Quantifier and Register Automata. In *LICS*, pages 17–26, 2006.

[16] Stéphane Demri, Ranko Lazić, and Arnaud Sangnier. Model checking freeze LTL over one-counter automata. In *Proceedings of the 11th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'08)*, 2008.

[17] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.

[18] Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

[19] Alin Deutsch, Liying Sui, and Victor Vianu. Specification and verification of data-driven web applications. *JCSS*, 73(3):442–474, 2007.

[20] G. Dong, R. Hull, B. Kumar, J Su, and G Zhou. A framework for optimizing distributed workflow executions. In *Proc. Intl. Workshop on Database Programming Languages (DBPL)*, pages 152–167, 1999.

[21] R. Hull et al. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proc. of 7th Intl. Workshop on Web Services and Formal Methods (WS-FM)*, 2010. To appear.

[22] Ronald Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982.

[23] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003.

[24] C. E. Gerede, K. Bhattacharya, and J. Su. Static analysis of business artifact-centric operational models. In *IEEE International Conference on Service-Oriented Computing and Applications*, 2007.

[25] C. E. Gerede and J. Su. Specification and verification of artifact behaviors in business process models. In *Proceedings of 5th International Conference on Service-Oriented Computing (ICSOC)*, Vienna, Austria, September 2007.

[26] R.J. Glushko and T. McGrath. *Document Engineering: Analyzing and Designing Documents for Business Informatics and Web Services*. MIT Press, Cmabridge, MA, 2005.

[27] R. Hull, F. Llirbat, B. Kumar, G. Zhou, G. Dong, and J. Su. Optimization techniques for data-intensive decision flows. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 281–292, 2000.

[28] R. Hull, F. Llirbat, E. Simon, J. Su, G. Dong, B. Kumar, and G. Zhou. Declarative workflows that support easy modification and dynamic browsing. In *Proc. Int. Joint Conf. on Work Activities Coordination and Collaboration*, 1999.

[29] Marcin Jurdzinski and Ranko Lazić. Alternation-free modal mu-calculus for data trees. In *LICS*, pages 131–140, 2007.

[30] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *STOC*, pages 302–311, 1984.

[31] S. Kumaran, R. Liu, and F. Y. Wu. On the duality of information-centric and activity-centric models of business processes. In *Proc. Intl. Conf. on Advanced Information Systems Engineering (CAISE)*, 2008.

[32] S. Kumaran, P. Nandi, T. Heath, K. Bhaskaran, and R. Das. ADoc-oriented programming. In *Symp. on Applications and the Internet (SAINT)*, pages 334–343, 2003.

[33] J. Küster, K. Ryndina, and H. Gall. Generation of BPM for object life cycle compliance. In *Proceedings of 5th International Conference on Business Process Management (BPM)*, 2007.

[34] R. Lazić, Th. Newcomb, J. Ouaknine, A. Roscoe, and J. Worrell. Nets with tokens which carry data. In *ICATPN'07*, volume 4546 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2007.

[35] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[36] R. Liu, K. Bhattacharya, and F. Y. Wu. Modeling business contexture and behavior using business artifacts. In *CAiSE*, volume 4495 of *LNCS*, 2007.

[37] D. Martin et al. OWL-S: Semantic markup for web services, W3C Member Submission, November 2003.

[38] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[39] Michael Meier, Michael Schmidt, Fang Wei, and Georg Lausen. Semantic query optimization in the presence of types. In *PODS*, pages 111–122, 2010.

[40] Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[41] P. Nandi and S. Kumaran. Adaptive business objects – a new component model for business integration. In *Proc. Intl. Conf. on Enterprise Information Systems*, pages 179–188, 2005.

[42] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *Intl. World Wide Web Conf. (WWW2002)*, 2002.

[43] F. Neven, T. Schwentick, and V. Vianu. Finite State Machines for Strings Over Infinite Alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004.

[44] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[45] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[46] M. Spielmann. Verification of relational transducers for electronic commerce. *JCSS.*, 66(1):40–65, 2003. Extended abstract in PODS 2000.

[47] J. Wang and A. Kumar. A framework for document-driven workflow systems. In *Business Process Management*, pages 285–301, 2005.

[48] Xiangpeng Zhao, Jianwen Su, Hongli Yang, and Zongyan Qiu. Enforcing constraints on life cycles of business artifacts. In *TASE*, pages 111–118, 2009.