Artifact Systems with Data Dependencies and Arithmetic

ELIO DAMAGGIO, Microsoft
ALIN DEUTSCH and VICTOR VIANU, University of California, San Diego

We study 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 postconditions, 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. Previous work identified the class of *guarded* artifact systems and properties, for which verification is decidable. However, the results suffer 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 article, 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 previous work. While the worst-case complexity of verification is nonelementary, we identify various realistic restrictions yielding more palatable upper bounds.

Categories and Subject Descriptors: H.1.m [**Information Systems**]: Models and Principles; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs

General Terms: Verification, Performance

ACM Reference Format:
Damaggio, E., Deutsch, A., and Vianu, V. 2012. Artifact systems with data dependencies and arithmetic. ACM Trans. Datab. Syst. 37, 3, Article 22 (August 2012), 36 pages.
DOI = 10.1145/2338626.2338628 http://doi.acm.org/10.1145/2338626.2338628

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 underspecifying 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 Nigam and Caswell [2003], deployed by IBM in commercial products and consulting services, and further studied in a line of follow-up works [Bhattacharya et al. 2007a, 2005b; Gerede et al. 2007; Gerede and Su 2007; Liu et al. 2007; Kumaran et al. 2008; Küster et al. 2007]. Business artifacts (or

This article is the extended version of the conference paper Damaggio et al. [2011]. It differs from the latter by including technical development, extended examples, and the full proofs.

This research was supported by the National Science Foundation under grant IIS-0916515. E. Damaggio was affiliated with UC San Diego while performing this work.

Author's addresses: E. Damaggio, Microsoft; A. Deutsch (corresponding author) and V. Vianu, University of California San Diego, San Diego, CA; email: deutsch@cs.ucsd.edu.

DOI 10.1145/2338626.2338628 http://doi.acm.org/10.1145/2338626.2338628

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 [Bhattacharya et al. 2007a].

Previous work [Deutsch et al. 2009] 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. The article considers 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 postconditions. Postconditions permit nondeterminism 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. Deutsch et al. [2009] identified the large and useful class of “guarded” artifact systems and properties, for which verification is decidable. However, the positive results of Deutsch et al. [2009] suffer 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 article 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 Deutsch et al. [2009].

We illustrate the variant of the artifact systems model adopted in this article 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, and expanded upon in the Appendix.

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 follows.

> 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.

Previous results from Deutsch et al. [2009] 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 and decrementing 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 arithmetic constraints $\mathcal{C}$ as long as satisfiability of ∃FO formulas over $\mathcal{C}$ is decidable. 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 Deutsch et al. [2009] 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. These descriptions are expressible by ∃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 nonelementary). 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.

1.1. Related Work

Data-aware business process models. The specific notion of artifact was first introduced in Nigam and Caswell [2003] and was further studied, from both practical and theoretical perspectives, in Bhattacharya et al. [2007a, 2007b, 2005], Gerede et al. [2007], Gerede and Su [2007], Liu et al. [2007], Kumaran et al. [2008], Küster et al. [2007], Fritz et al. [2009], Hull et al. [2010], Damaggio et al. [2011b]. Some key roots of the artifact model are present in "adaptive objects" [Kumaran et al. 2003], "adaptive business objects" [Nandi and Kumaran 2005], "business entities", "document-driven" workflow [Wang and Kumar 2005], and "document" engineering [Glushko and McGrath 2005]. The Vortex framework [Hull et al. 1999], and [Dong et al. 1999, 2000] 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 [McIlraith et al. 2001; Martin et al. 2003] describes the semantics of services with input, output, pre-, and postconditions.

Static analysis of data-aware business processes. Work on formal analysis of artifact-based business processes in restricted contexts has been reported in Gerede et al. [2007], Gerede and Su [2007], and Bhattacharya et al. [2007b]. Properties investigated include reachability [Gerede et al. 2007; Gerede and Su 2007], general temporal constraints [Gerede and Su 2007], and the existence of complete execution or dead end [Bhattacharya et al. 2007b]. For the variants considered in each article, verification is generally undecidable; decidability results were obtained only under rather severe restrictions, for example, restricting all preconditions to be "true" [Gerede et al. 2007], restricting to bounded domains [Gerede and Su 2007; Bhattacharya et al. 2007b], or restricting the pre- and postconditions to refer only to artifacts (and not their variable values) [Gerede and Su 2007]. None of the aforesaid papers permits an underlying database, integrity constraints, or arithmetic.

Calvanese et al. [2009] 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. Zhao et al. [2009] 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.

Recent work has shown the applicability of our verification results to a rich business process model introduced by IBM under the name of Guard-Stage-Milestone model (GSM) [Hull et al. 2010]. A natural subclass of the GSM model was shown to be simulated by feedback-free artifact systems, thus rendering our verification technique applicable to this subclass [Damaggio et al. 2011b].

Static analysis for semantic Web services is considered in Narayanan and McIlraith [2002], but in a context restricted to finite domains.

More recently, Abiteboul et al. [2009] has studied automatic verification in the context of business processes based on Active XML documents.

A direction related to verification considers automatic refinement of an artifact-based business process (by adding business rules, i.e., constraints on the runs) such that it permits some run that satisfies a given property (a.k.a. goal) [Fritz et al. 2009]. For the sake of decidability, data values are abstracted away: there is no underlying database, no integrity constraints, and the values of the artifact variables are not inspected by the business process specification, business rules, and properties, all of which can only check whether a non-Boolean attribute is initialized and a Boolean attribute is true.

Our work is most closely related to the one in Deutsch et al. [2009], which identifies the class of *guarded* artifact systems and LTL-FO properties, for which verification is decidable. Moreover, this is shown to hold even in the presence of a dense linear order on the data domain (extended to an arbitrary total order in Segoufin and Torunczyk [2011]). The two settings have in common the underlying database and the infinite ordered data domain, as well as the syntax for pre-, postconditions, and properties. However, Deutsch et al. [2009] consider no dependencies and no arithmetic operations. The results in Deutsch et al. [2009] 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 operations consist in incrementing and decrementing counters. The novel proof technique based on describing configuratons using inherited constraints is fundamentally different from the one employed for guardedness.

The works Deutsch et al. [2007], Spielmann [2003], and Abiteboul et al. [2000] are ancestors of Deutsch et al. [2009] 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 Deutsch et al. [2009]. They all disallow the linear order on the domain. Also, they limit artifact values to essentially come from the active domain of the database, thus ruling out external inputs, partially specified services, and arithmetic.

In this article, we adopt for properties the language of first-order logic extended with linear-time temporal logic operators. Similar extensions have been previously used in various contexts [Emerson 1990; Abiteboul et al. 1996; Spielmann 2003; Deutsch et al. 2007, 2006].

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 Burkart et al. [2001] 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 [Bouajjani et al. 2003], rewriting systems with data [Bouajjani et al. 2007a, 2007b], Petri nets with data associated to tokens [Lazić et al. 2007], automata and logics over infinite alphabets [Bouyer et al. 2003; Bouyer 2002; Neven et al. 2004; Demri and Lazić 2006; Jurdzinski and Lazić 2007; Bojanczyk et al. 2006; Bouajjani et al. 2007a], and temporal logics manipulating data [Demri and Lazić 2006; Demri et al. 2008]. However, the restricted use of data and the particular properties verified have limited applicability to the business artifacts setting.

1.2. Article 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 Deutsch et al. [2009] 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 5.5. We extend our decidability result to the presence of dependencies in Section 6. We end with brief conclusions. An appendix provides our full runnning example.

2. FRAMEWORK

The arithmetic constraints considered here are over domain $\mathbb{Q}$, the rational numbers. While databases could use nonnumeric 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 arithmetic relation symbols, each of which has a fixed interpretation as the set of solutions over $\mathbb{Q}$ 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. A database schema $\mathcal{DB}$ consists of a finite set of database relation symbols not in $\mathcal{C}$, with associated arities. An instance over $\mathcal{DB}$ maps each relation symbol of $\mathcal{DB}$ to a finite relation of the same arity over $\mathbb{Q}$.

We next define a notion of schema for artifact systems.

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 database 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. The variables $\bar{x}$ and $\bar{x}'$ will be used to refer to consecutive values of the variables of the artifact.

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 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 *precondition*, 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 *postcondition*, 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 nonempty set of services over $\mathcal{A}$, and Π is a precondition (as before, 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 with respect to 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. (i) In the formal model given before, it is assumed for simplicity that each artifact system has a single artifact schema. The model and results can be easily extended to systems with multiple artifact schemas. Indeed, these can be reduced to the single-artifact case by taking the cross-product of all artifacts. (ii) In Deutsch et al. [2009], 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 that follows shortly. 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. The full example is provided in the Appendix.

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 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:

$\texttt{status}, \texttt{prod\_id}, \texttt{ship\_type}, \texttt{coupon}, \texttt{amount\_owed}, \texttt{amount\_paid}, \texttt{amount\_refunded}.$

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):

$\texttt{PRODUCTS}(\underline{\texttt{id}}, \texttt{price}, \texttt{availability}, \texttt{weight}),$
$\texttt{COUPONS}(\underline{\texttt{code}}, \texttt{type}, \texttt{value}, \texttt{min\_value}, \texttt{free\_shiptype}),$
$\texttt{SHIPPING}(\underline{\texttt{type}}, \texttt{cost}, \texttt{max\_weight}),$
$\texttt{OFFERS}(\underline{\texttt{prod\_id}}, \texttt{discounted\_price}, \texttt{active}).$

The database also satisfies the following foreign keys:

$\texttt{COUPONS}[\texttt{free\_shiptype}] \subseteq \texttt{SHIPPING}[\texttt{type}]$ and
$\texttt{OFFERS}[\texttt{prod\_id}] \subseteq \texttt{PRODUCTS}[\texttt{id}].$

Recall that in our formal framework, the data domain for all relations is defined to be $\mathbb{Q}$. However, in order to enhance readability and without loss of generality, we allow nonnumeric 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 nonpositive value can play this role.) The initialization is easily expressed by the artifact system's precondition Π.

The services. The following services model a few of the business process tasks (see Appendix A for a full list).

choose_product. The customer chooses a product.

$\pi : \texttt{status} = \text{“edit\_prod”}$
$\psi : \exists p, a, w(\texttt{PRODUCTS}(\texttt{prod\_id}', p, a, w) \wedge a > 0)$
$\wedge \texttt{status}' = \text{”edit\_shiptype”}$

choose_shiptype. The customer chooses a shipping option.

$\pi : \texttt{status} = \text{“edit\_ship”}$
$\psi : \exists c, l, p, a, w(\texttt{SHIPPING}(\texttt{ship\_type}', c, l) \wedge$
$\texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w) \wedge l > w) \wedge$
$\texttt{status}' = \text{“edit\_coupon”} \wedge \texttt{prod\_id}' = \texttt{prod\_id}$

apply_coupon. The customer optionally inputs a coupon number.

π : $\texttt{status} = $ "edit_coupon"
ψ : $(\texttt{coupon}' = \lambda \wedge \exists p, a, w, c, l(\texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w) \wedge \texttt{SHIPPING}(\texttt{ship\_type}, c, l) \wedge \texttt{amount\_owed}' = p + c) \wedge \texttt{status}' = \text{“processing”} \wedge \texttt{prod\_id}' = \texttt{prod\_id} \wedge \texttt{ship\_type}' = \texttt{ship\_type}) \vee (\exists t, v, m, s, p, a, w, c, l(\texttt{COUPONS}(\texttt{coupon}', t, v, m, s) \wedge \texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w) \wedge \texttt{SHIPPING}(\texttt{ship\_type}, c, l) \wedge p + c \geq m \wedge (t = \text{“free\_shipping”} \rightarrow (s = \texttt{ship\_type} \wedge \texttt{amount\_owed}' = p)) \wedge (t = \text{“discount”} \rightarrow \texttt{amount\_owed}' = p + c - v)) \wedge \texttt{status}' = \text{“processing”} \wedge \texttt{prod\_id}' = \texttt{prod\_id} \wedge \texttt{ship\_type}' = \texttt{ship\_type})$

Notice that the preconditions 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 postconditions 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 precondition). 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 postconditions achieves a sequential filling of the order, starting from the choice of product and ending with the claim of a coupon.

A postcondition 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 postconditions may query the database. For instance, in service **choose_product**, the postcondition 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 postconditions. 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 postcondition 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. [Fagin 1982; Abiteboul et al. 1995]. In this article, we refer to embedded dependencies in short as "dependencies". We call ϕ the *premise* and ψ the *conclusion*. We write $A \models \mathcal{I}$ if the instance A satisfies all the dependencies in $\mathcal{I}$.

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

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

The foreign key constraint on the OFFERS table is expressed by

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

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 **G** (always), **F** (eventually), **X** (next), and **U** (until) (e.g., see Pnueli [1977]). 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 (excluding) that point. For example, $\mathbf{G}(p \rightarrow \mathbf{F}q)$ says that whenever p holds, q must hold sometime in the future.

The extension of LTL that we use, called[1] LTL-FO, is obtained from LTL by replacing propositions with quantifier-free FO statements about particular artifact records in the run. The statements use the artifact variables and the database. In addition, they may use *global* variables, shared by different statements and allowing to refer to values in different records. The global variables are universally quantified over the entire property (see Example 3.2).

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 $\mathbb{Q}$.

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.

(φ_1) $\forall x \mathbf{G}((\texttt{amount\_paid} = x \wedge \texttt{amount\_paid} = \texttt{amount\_owed}) \rightarrow \mathbf{F}(\texttt{status} = \text{"shipped"} \vee \texttt{amount\_refunded} = x))$

[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.

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 $f_1(q) = \texttt{status} =$ "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 postcondition of service **choose_product**, (ii) by **choose_shiptype**, and (iii) by **apply_coupon**. In the postconditions, 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 postconditions 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 before, one can construct in linear time an artifact system Γ' such that $\Gamma \models \forall \bar{y} \varphi_f$ iff $\Gamma' \models \varphi_f$.*

PROOF. Let $\Gamma' = \langle \mathcal{A}', \Sigma', \Pi \rangle$, where $\mathcal{A}' = \langle \bar{x} \cup \bar{y}, \mathcal{DB} \rangle$ and $\Sigma' = \{(\pi, \psi \bigwedge_{y \in \bar{y}} (y = y') \mid (\pi, \psi) \in \Sigma\}$. Thus, Γ' is obtained from Γ by adding $\bar{y}$ to its artifact variables and propagating their values at each transition. It is clear that $\Gamma \models \forall \bar{y} \varphi_f$ iff $\Gamma' \models \varphi_f$. $\square$

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 postconditions 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 postconditions of* Σ *and* Π *use only constraints in* $\mathcal{C}$*, whether* $\Gamma \models \varphi$.

Proof. Part (i) follows from Theorem 4.2 in Deutsch et al. [2009]. Part (ii) is shown by defining an artifact system that simulates a counter (Minski) machine in conjunction with a property that forbids reachability of a given state of the machine. Informally, a counter machine consists of the following.

—a finite set Q of states, with an initial state q_0;
—two integer counters c_1 and c_2;
—a finite set of instructions of the form

$$\langle \gamma, p \rangle \rightarrow \langle q,\ \text{inc/dec}(c_i) \rangle, i \in \{1, 2\},$$

where $p, q \in Q$, and γ is *true* or $c_j = 0$, $j \in \{1, 2\}$.

Intuitively an instruction as given allows a transition from state p to state q if γ holds. In the transition, counter c_i is incremented or decremented by one (the machine blocks under a decrement instruction if $c_i = 0$). A configuration is a triple (s, m, n), where $s \in Q$ and m, n are nonnegative integers. The initial configuration is $(q_0, 0, 0)$. Transitions between configurations are caused by the preceding instructions in the obvious way. It is known that it is undecidable, given a counter machine M and a state $r \in Q$, whether r is reachable in M (i.e., whether M can reach from the initial configuration some configuration (r, m, n) for $m, n \geq 0$).

Given a counter machine M and a state r of M, we construct an artifact system $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ and an LTL-FO property φ_f such that r is reachable in M iff $\Gamma \not\models \varphi_f$. The artifact schema $\mathcal{A}$ has variables $(\mathit{state}, c_1, c_2)$ and empty database schema. The precondition Π is $\mathit{state} = q_0 \wedge c_1 = 0 \wedge c_2 = 0$. For each instruction $\langle \gamma, p \rangle \rightarrow \langle q,\ \text{inc/dec}(c_i) \rangle$ of M, Σ contains a service with precondition $\gamma \wedge \mathit{state} = p$ and postcondition $\mathit{state}' = q \wedge c_i' = c_i + / - 1 \wedge c_i' \geq 0$. Finally, let φ_f be $\mathbf{G}\neg(\mathit{state} = r)$. It is clear that r is reachable in M iff $\Gamma \not\models \varphi_f$. Since state reachability is undecidable for counter machines [Minsky 1967], part (ii) 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 Deutsch et al. [2009] towards decidable static verification. Theorem 3.4 shows that the results of Deutsch et al. [2009] do not transfer to the new setting. As detailed shortly, 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 services that perform the increment or decrement operations on each counter variable, and allow the run to "feed back" the counter value into the same service (by updating the same counter variable) for an unbounded number of times. Intuitively, this ability allows to simulate recursive computation and 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 and practical relevance of feedback freedom are further discussed after the formal definition.

4.1. 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. The symbolic run is associated to both the artifact system and a specific property to be verified. Intuitively, a symbolic run consists 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 postcondition of the service applied at each transition in the run, and also specify which FO components of the LTL-FO property 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 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 shortly, 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.

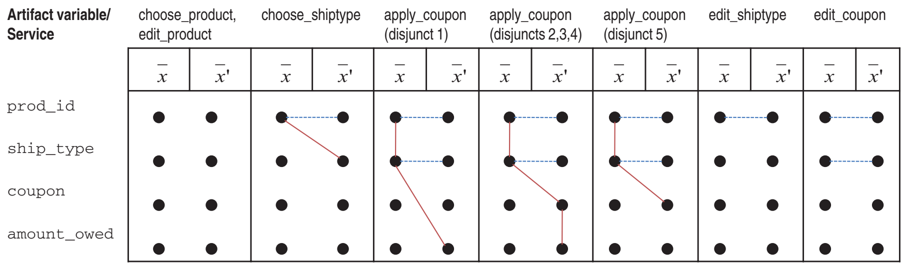
(2) Δ_{φ_f} containing, for each $\sigma \in 2^P$, all disjuncts of the DNF of the formula

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

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 earlier 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 $\neg\varphi$.

4.2. Feedback Freedom

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.



Artifact variable/ Service
choose_product, edit_product
choose_shiptype
apply_coupon (disjunct 1)
apply_coupon (disjuncts 2,3,4)
apply_coupon (disjunct 5)
edit_shiptype
edit_coupon
$\bar{x}$
$\bar{x}'$
prod_id
ship_type
coupon
amount_owed

Fig. 1. Graphs G_ψ and E_ψ for the symbolic transition templates in Example 4.1.

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 (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).

Example 4.1. We illustrate symbolic transition templates and the associated graphs G_ψ, E_ψ, G_ϱ, E_ϱ 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 postcondition for each service into prenex DNF, and add each disjunct as a separate formula to Δ_Σ. The pre- and postconditions 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 shortly and list the others in Appendix A; 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.

$$\begin{aligned}\xi_2\colon\ & \texttt{prod\_id}' = \texttt{prod\_id} \wedge \texttt{ship\_type}' = \texttt{ship\_type}\\ & \wedge\ \texttt{status} = \text{“edit\_coupon”} \wedge \texttt{status}' = \text{“processing”} \wedge\\ & \exists\, t, v, m, s, p, a, w, c, l\ (\\ & \quad \texttt{COUPONS}(\texttt{coupon}', t, v, m, s) \wedge \texttt{PRODUCTS}(\texttt{prod\_id}, p, a, w)\\ & \quad \wedge \texttt{SHIPPING}(\texttt{ship\_type}, c, l) \wedge \neg(t = \text{“free\_shipping”})\\ & \quad \wedge\ p + c \geq m \wedge\ \texttt{amount\_owed}' = p + c - v)\end{aligned}$$

Given the preceding sets Δ_Σ, Δ_{φ_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 depicts the graphs G_ψ and E_ψ for all transition templates. 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. In the case of **apply_coupon**, we indicate which disjunct ξ_i is used.

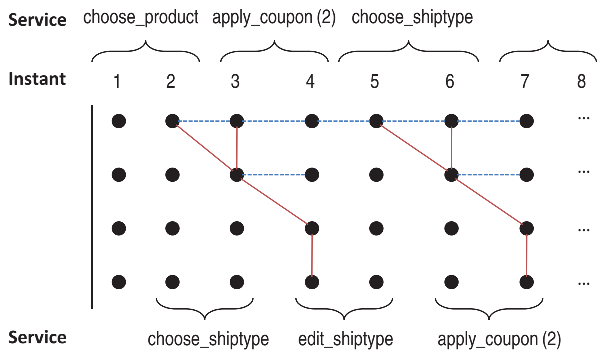


Fig. 2. Computation graph G_ϱ for Example 4.1.

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 postcondition. 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 nonartifact 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 sometimes omit edges in either the G_ψ and E_ψ graphs when these edges belong to the transitive closure of the drawn edges (for instance, we omit the solid edges prod_id − coupon′ or prod_id − amount_owed′ for **apply_coupon**(2,3,4), and prod_id − amount_owed′ for **apply_coupon**(1)). This loss of visual information is irrelevant because, as detailed soon, we are exclusively interested in answering reachability questions in these graphs.

Other services include **edit_product, edit_shiptype**, and **edit_coupon**, whose specification we relegate to Appendix A. 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 reinput 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 reinput 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 2. 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.

We now define feedback freedom. We assume the notation developed earlier.

Definition 4.2. (Γ, φ_f) is *feedback-free* if 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 2, 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

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

Notice that every path from $\texttt{ship\_type}_3$ to $\texttt{ship\_type}_7$ must pass through node $\texttt{prod\_id}_4$, and that

$$\begin{aligned} \textit{span}([\texttt{prod\_id}_4]) &\supseteq [2, 7] \\ &\supseteq \textit{span}([\texttt{ship\_type}_3]) \cup \textit{span}([\texttt{ship\_type}_7]). \end{aligned}$$

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 ($\textit{span}([y]) \supseteq \textit{span}([x_i]) \cup \textit{span}([x_j])$).

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 reconsidering (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.

4.3. Practical Relevance of Feedback Freedom

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

A way to visualize the types of unboundedness represented in business processes is the following. Consider a *call graph* whose nodes are the actual invocations of services during the run (with instantiated input parameters). Draw an edge from task invocation a to task invocation b whenever b takes as input some of a's output, or reads global data modified by a. The call graph is (or can be unfolded into) a tree. Feedback-free processes disallow call trees of unbounded height, but allow unbounded outdegrees of the tree nodes.

Additional evidence for the relevance of the notion of feedback freedom is provided by the fact that it arises naturally in a widely adopted design paradigm for business processes (beyond artifact systems). The paradigm consists in specifying the business process workflow in a modular, hierarchically organized manner. The building blocks of such workflows are atomic tasks and complex tasks (recursively defined as subworkflows involving atomic or complex tasks). The task-subtask relationship is often strictly tree-shaped. We shall refer to such business processes as *hierarchical*. A line of extensive surveys of real-life business processes [Russell et al. 2005a, 2005b, 2006] argues that their majority is hierarchically structured, and in addition exhibits the following control flow restrictions.

—Recursive invocation among tasks is limited to only the case when a task t fails and it raises an exception that is handled by an ancestor a, who decides to restart its descendants on the path leading to t (without changing a's state).
—Whenever a task fails, its descendants also fail. When a task is restarted, its state is reinitialized.

The surveys go as far as to advocate that, to be sensible, a business process specification should conform to the aforesaid restrictions. Let us refer to the resulting business processes as *well-behaved hierarchical*.

There is a natural way to incorporate artifacts into well-behaved hierarchical business processes: atomic tasks correspond to services, and tasks encapsulate their own local artifacts. These artifacts may contain "input" and "output" variables, where input variables are set by the parent task and preserved during the owner task's execution, while output variables are set by the owner task and are visible to the parent task only.

Due to space limitations, we keep the presentation of the model at the informal level and refer the interested reader to Damaggio [2011], where the model is formalized under the name AWE (Artifact Workflows with Exceptions). Remarkably, Damaggio [2011] shows that AWE business processes can be simulated by feedback-free artifact systems[2]! The result relies crucially on the fact that recursion is limited to exception handling, and that restarted tasks forget their prior state (i.e., reinitialize their artifact variables). In the AWE model, it is natural to expect properties to obey the hierarchy and well-behavedness restrictions, in the sense that their FO components only relate artifact variables according to the scoping rules mentioned before. Call such properties *well-behaved*. It follows that pairs of well-behaved hierarchical processes and properties can be simulated as feedback-free pairs of artifact systems and properties. This suggests that feedback freedom is satisfied by naturally arising classes of business processes and properties.

Finally, feedback feedom has proven useful in enabling the application of our verification results to a rich business process model introduced recently by IBM under the name of Guard-Stage-Milestone model (GSM) [Hull et al. 2010]. A natural subclass of the GSM model was shown to be simulated by feedback-free artifact systems, thus rendering our verification technique applicable to this subclass [Damaggio et al. 2011b].

4.4. Testing Feedback Freedom

We next show that testing feedback freedom of (Γ, φ_f) can be reduced to testing emptiness of a nondeterministic finite-state automaton $A_{(\Gamma,\varphi_f)}$. Since a direct construction requires some rather tedious bookkeeping, we describe instead for simplicity a *two-way* alternating automaton $T_{(\Gamma,\varphi_f)}$ whose emptiness is equivalent to feedback freedom. Recall that, in each transition, the head of a two-way automaton may move to the right or to the left of the current position, or may be stationary. An alternating automaton augments nondeterminism with universal and existential states. Acceptance from an existential state occurs if there *exists* a transition to a next state leading to acceptance, whereas acceptance from a universal state occurs if *all* transitions to a next state lead to acceptance. It is well-known that two-way alternating automata can be converted to classical one-way automata, and testing emptiness is PSPACE-complete (see Martens et al. [2008], Ladner et al. [1984], and Birget [1996]).

Intuitively, $T_{(\Gamma,\varphi_f)}$ guesses a violation of feedback freedom for (Γ, φ_f). Recall that a violation consists in a prefix ϱ of a symbolic run in which there are two occurrences x_i, x_j of the same variable x in configurations i and j such that there exists a path from x_i to x_j in G_ϱ so that no variable v occurring along the path has the property that $span([x_i]) \cup span([x_j]) \subseteq span([v])$.

We next describe $T_{(\Gamma,\varphi_f)}$ informally. The alphabet consists of the symbolic transition templates, augmented with several kinds of markers:

—ϵ (emtpy marker);
—x^1 for $x \in \bar{x}$; this identifies one occurrence of x;
—x^2 for $x \in \bar{x}$; this identifies a second occurrence of x;
—$[^1_x$ and $]^1_x$ for $x \in \bar{x}$; this identifies the beginning, respectively, the end of the span of $[x^1]$;
—$[^2_x$ and $]^2_x$ for $x \in \bar{x}$; this identifies the beginning, respectively the end of the span of $[x^2]$.

Formally, the alphabet of $T_{(\Gamma,\varphi_f)}$ consists of pairs of symbolic transition templates and subsets of the these markers. The automaton $T_{(\Gamma,\varphi_f)}$ performs the following tests, carried out sequentially in multiple passes. For an occurrence of a marker α, we denote by $pos(\alpha)$ its position in the word.

[2]Technically, the definition of feedback freedom involves both an artifact system and a property, but it adapts straightforwardly to an isolated artifact system: just consider the trivially true property; see Remark 4.8.

—There is a single occurrence of each of the markers x^i, $[^i_x$, $]^i_x$ for some $x \in \bar{x}$ and $pos([^i_x) \leq pos(x^i) \leq pos(]^i_x)$, $i \in \{1, 2\}$.
—$span([x_i]) = [pos([^i_x), pos(]^i_x)]$; this can be tested by first generating a path in E_ϱ from x in $pos(x^i)$ to some variable in $pos([^i_x)$, and similarly a path from x in $pos(x^i)$ to some variable in $pos(]^i_x)$, and then checking that *every* nondeterministically generated path in E_ϱ originating from x in $pos(x^i)$ never reaches $pos([^i_x) - 1$ or $pos(]^i_x) + 1$. The latter is easily done using universal states.
—There exists a path from x in $pos(x^1)$ to x in $pos(x^2)$ in G_ϱ such that for every variable v along the path, $span([v])$ does not include $[min\{pos([^i_x) \mid i = 1, 2\}, max\{pos(]^i_x) \mid i = 1, 2\}]$. This is checked using an alternation of existential and universal states. Specifically, every time a new v is generated along the path, the following must be tested for *every* path in E_ϱ starting at v: the path does not touch both $min\{pos([^i_x) \mid i = 1, 2\}$ and $max\{pos(]^i_x) \mid i = 1, 2\}]$.

It is easily seen that one can construct a two-way alternating automaton testing the preceding properties, that uses polynomially many states in (Γ, φ_f). However, the automaton uses an alphabet of exponential size with respect to (Γ, φ_f), because there are exponentially many symbolic transition templates. Observe that the size of *each* alphabet symbol is polynomial in (Γ, φ_f).

We have the following.

LEMMA 4.5. *(Γ, φ_f) is feedback-free iff the language accepted by $T_{(\Gamma,\varphi_f)}$ is empty.*

As stated earlier, emptiness of two-way alternating automata can be checked in PSPACE (see Martens et al. [2008], Ladner et al. [1984], and Birget [1996]) with respect to the number of states and the size of the alphabet. Recall that the size of the alphabet of $T_{(\Gamma,\varphi_f)}$ is exponential with respect to (Γ, φ_f). However, because the reduction in Birget [1996] yields an exponential blowup of the input in the number of states but not in the size of the alphabet, it follows that the complexity of testing emptiness of $T_{(\Gamma,\varphi_f)}$ remains PSPACE with respect to (Γ, φ_f).

In summary, we have shown the following.

PROPOSITION 4.6. *Feedback freedom of (Γ, φ_f) can be tested in* PSPACE.

Remark 4.7. 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).

Remark 4.8. The notion of feedback freedom developed earlier applies to an artifact system *together* with a property to be verified. Clearly, one can define a notion of feedback-free artifact system independently of any property (specifically, an artifact system Γ is feedback-free if (Γ, true) is feedback-free). In general, if Γ is feedback-free and φ_f is a property, it is not always the case that (Γ, φ_f) remains feedback-free. However, there are important special cases when this does hold. Such cases are of practical interest, because they allow testing feedback freedom only once for Γ alone, rather than repeatedly for each property to be verified. For example, suppose that each FO component of φ_f contains only one variable (such is the case in Example 4.1). Then it is easy to see that φ_f contributes nothing to the computation graphs of symbolic runs. Thus, if Γ is feedback-free then so is (Γ, φ_f). This approach can be extended to more flexible restrictions on properties that guarantee preservation of feedback freedom of Γ, such as the ones discussed in Section 4.3.

5. VERIFICATION OF FEEDBACK-FREE SYSTEMS WITH ARITHMETIC

Our main result is that model checking LTL-FO properties is decidable if the artifact system together with the property are feedback-free.

THEOREM 5.1. *It is decidable, given an artifact system Γ without dependencies, 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.*

For simplicity of exposition, we first consider in this section the case of artifact systems with arithmetic but no data dependencies. We then extend this result in Section 6 to the presence of dependencies on database relations. Thus, in the remainder of this section we assume that no dependencies are specified on the database.

The proof of Theorem 5.1 requires developing some technical machinery, to which the remainder of this section is dedicated.

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 definitions 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, $Runs_D(\Gamma) = \cup\{Runs_D(\varrho) \mid \varrho$ 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 $\neg\varphi_f$.*

PROOF. Consider (i). Suppose $\rho = \{\rho_i\}_{i \geq 0}$ is a run of Γ on database D. By definition, for each $i \geq 0$ there exists a formula $\delta(\bar{x}_i, \bar{x}_{i+1}) \in \Delta_\Sigma$ so that $D \cup \mathcal{C} \models \delta(\rho_i, \rho_{i+1})$. Also, for each FO component $f(p)$ of $\neg\varphi_f$, either $D \cup \mathcal{C} \models f(p)(\rho_i, \rho_{i+1})$ or $D \cup \mathcal{C} \models \neg f(p)(\rho_i, \rho_{i+1})$. It follows that there exists a formula ψ_i constructed as before from Δ_Σ and Δ_{φ_f} so that $D \cup \mathcal{C} \models \psi_i(\rho_i, \rho_{i+1})$. (For ψ_0, the precondition Π is also taken into account in the obvious way.) Let $\varrho = \{\psi_i\}_{i \geq 0}$. By construction, $\rho \in Runs_D(\varrho)$. The converse is similar.

Part (ii) follows from the observation that for every run $\rho \in Runs_D(\varrho)$, the same FO components of φ_f are satisfied in the i-th configuration of ρ and ϱ. Thus $\rho \models \neg\varphi_f$ iff $\varrho \models \neg\varphi_f$. □

In view of the preceding, 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 up to it, that is, $\{\psi_i\}_{i<j}$. 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 $j > 0$,*

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

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

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 nonequivalent formulas of given quantifier rank over a given vocabulary (e.g., see Libkin [2004]). The number of nonequivalent 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 postconditions of services in Σ.*

The proof (next) 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 nondeterministic, 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.

PROOF. Let η_j be defined as earlier. Recall that service pre- and postconditions may contain $\exists$FO formulas, so the ψ_i may generally contain such subformulas. We assume first for simplicity that the ψ_i are quantifier-free, and deal with the $\exists$FO subformulas later. We denote by $\xi = \bigwedge_{i<j} \psi_i$, so $\eta_j = \exists\bar{z}\xi$. Consider the equivalence relation induced by the equality atoms in η_j on the variables $\{\bar{x}_i\}_{i\leq j}$. We denote the equivalence class of y by $[y]$ (when η_j is understood). By slight abuse, we use such equivalence classes as variables in the fomula we are constructing. Thus, we begin by eliminating all positive equality atoms from η_j and replacing in each nonequality atom each variable y by its equivalence class $[y]$. Let us denote by $[\eta_j]$ the resulting formula. Note that the set of free variables of $[\eta_j]$ is now the set of equivalence classes of variables in $\bar{x}_j$, denoted $[\bar{x}_j]$. It is clear by construction that for every database D, $D\cup\mathcal{C} \models \eta_j(\bar{x}_j)$ iff $D\cup\mathcal{C} \models [\eta_j]([\bar{x}_j])$. We will ensure later that the final formula $\bar{\eta}_j$ has free variables $\bar{x}_j$ rather than $[\bar{x}_j]$.

Note that $[\eta_j]$ is in existential prenex form. We next show that the quantifiers can be pushed inside, resulting in a formula η_j^* of quantifier rank bounded by k^2. To this end, we will use the computation graph associated with a symbolic run (or a prefix thereof). Let us denote by G_j the graph on equivalence classes induced by the computation graph of ξ. More precisely, if there is an edge from x_i to y_n in G_ξ, then there is an edge from $[x_i]$ to $[y_n]$ in G_j. Consider a formula β consisting of the conjunction of a subset of the literals of $[\xi]$ and let G_β be its associated computation graph (so a subgraph of G_j). Let $\bar{y}$ be a subset of the variables of β. The *width* of $\bar{y}$ is defined as

$$w(\bar{y}) \;=\; max\{|\bar{v}| \;\mid \bar{v} \subseteq \bar{x}_i,\; \text{and each } v \in \bar{v} \text{ is in an equivalence class } y \in \bar{y}\}.$$

We prove the statement for formulas $\gamma(\bar{s}) = \exists\bar{y}\beta(\bar{y},\bar{s})$ with free variables $\bar{s}$, such that the restriction of G_β to $\bar{y}$, denoted $G_\beta|\bar{y}$, is connected. More precisely, we show that each such formula can be rewritten with quantifier rank bounded by $k \cdot w(\bar{y})$, where $k = |\bar{x}|$. The proof is by induction on $w(\bar{y})$. If $w(\bar{y}) = 0$ then $\bar{y} = \emptyset$ so $qd(\gamma) = 0$. Now suppose the statement holds for $w(\bar{y}) < n$ and consider $\gamma(\bar{s}) = \exists \bar{y} \beta(\bar{s})$ where $w(\bar{y}) = n$ and $G_\beta|\bar{y}$ is connected. Consider the set S of maximal spans of variables in $\bar{y}$ in ϱ_j (with respect to inclusion). For each $s \in S$, let y_s be a variable in $\bar{y}$ for which $span(y_s) = s$, and let $\bar{y}_{max} = \{y_s \mid s \in S\}$. We note the following.

(i) There is no variable $x \in \bar{x}$ and $i \neq m$ such that $[x_i] \neq [x_m]$ and $[x_i], [x_m] \in \bar{y}_{max}$.

Indeed, suppose there were such $[x_i], [x_m]$. Since $G_\beta|\bar{y}$ is connected, there is a path from $[x_i]$ to $[x_j]$ in $G_\beta|\bar{y}$. Since $(\Gamma, \forall \bar{y} \varphi_f)$ is feedback-free, the path must go through a class $[t] \in \bar{y}$, whose span strictly includes the spans of both $[x_i]$ and $[x_j]$. Since by definition of $\bar{y}_{max}$ the spans of $[x_i]$ and $[x_m]$ are distinct, the inclusion is strict, contradicting the maximality of the spans of $[x_i]$ and $[x_j]$.

Note that, as a consequence of (i), we have the following.

(ii) The size of $\bar{y}_{max}$ is bounded by k.

Now let $\bar{v} = \bar{y} - \bar{y}_{max}$ and consider $G_\beta|\bar{v}$. Let $H_1, \ldots, H_c$ be its connected components. For each $r \in [1, c]$ let $\bar{y}_r$ be the set of nodes of H_r and let β_r be the formula consisting of the conjunction of the literals in β using only variables in $\bar{s} \cup \bar{y}_{max} \cup \bar{y}_r$. We claim the following.

(iii) $\exists \bar{y} \beta$ is equivalent to the formula $\exists \bar{y}_{max}(\bigwedge_{1 \leq r \leq c} \exists \bar{y}_r(\beta_r))$.

To see that (iii) holds, note first that each literal of β belongs to some β_r. Also, $\bar{y} = \bar{y}_{max} \cup \bigcup_{1 \leq r \leq c} \bar{y}_r$. Finally, distinct β_r have no variables in common besides those in $\bar{s} \cup \bar{y}_{max}$.

Now consider a formula $\exists \bar{y}_r \beta_r$. The graph $G_{\beta_r}|\bar{y}_r$ equals H_r so is connected by construction. Consider $w(\bar{y}_r)$. Since $\bar{y}_{max}$ contains variables covering all maximal spans, for each i such that $x \in \bar{x}$ and $[x_i] \in \bar{v}_r$, there exists $v \in \bar{x}$, $v \neq x$, such that $[v_i] \in \bar{y}_{max}$. It follows that $w(\bar{y}_r) \leq w(\bar{y}) - 1$. Thus, we can apply the induction hypothesis and $\exists \bar{y}_r \beta_r$ can be rewritten with quantifier rank at most $k \cdot w(\bar{y}_r)$. It follows that $\exists \bar{y} \beta$ can be rewritten with quantifier rank at most $|\bar{y}_{max}| + k \cdot max\{w(\bar{y}_r) \mid 1 \leq r \leq c\}$. Since by (ii) $|\bar{y}_{max}| \leq k$, this is bounded by $k(1 + (w(\bar{y}) - 1) = k \cdot w(\bar{y})$.

In summary, we have shown that every formula $\exists \bar{y} \beta$ where $G_\beta|\bar{y}$ is connected can be rewritten with quantifier rank at most $k \cdot w(\bar{y})$. Finally, consider $[\eta_j] = \exists [\bar{z}][\xi]$. Similarly to the induction step, consider the connected components $H_1, \ldots, H_c$ of $G_j|[\bar{z}]$. For each $r \in [1, c]$ let $\bar{y}_r$ be the set of nodes of H_r and let ξ_r be the formula consisting of the conjunction of the literals in ξ using only variables in $[\bar{x}_j] \cup \bar{y}_r$. It is clear that $[\eta_j]$ is equivalent to the formula $\bigwedge_{1 \leq r \leq c} \exists \bar{y}_r(\xi_r)$. By construction, the graph $G_{\xi_r}|\bar{y}_r$ is connected for each r, so by the preceding $\exists \bar{y}_r(\xi_r)$ can be rewritten with quantifier rank bounded by $k \cdot w(\bar{y}_r) \leq k^2$ for each r. It follows that $[\eta_j]$ can be rewritten with the same quantifier rank k^2.

Recall that we have assumed that all formulas ψ_i are quantifier-free. If $\exists$FO formulas are used in Γ, the previous construction can be extended as follows. First, associate to each maximal subformula α used in Γ such that $\alpha(\bar{u}) = \exists \bar{v} \gamma(\bar{v}, \bar{u})$, where $\bar{u}$ are the free variables, a new relation symbol R_α of arity $|\bar{u}|$, and replace in Γ each occurrence of $\alpha(\bar{u})$ by $R_\alpha(\bar{u}) \wedge eq(\bar{u})$, where $eq(\bar{u})$ is the conjunction of all equalities among variables in $\bar{u}$, resulting from taking the transitive closure of the equality graph of $\gamma(\bar{v}, \bar{u})$. The preceding construction then yields a CQ$^\neg$ formula of quantifier rank k^2 over the augmented vocabulary. Finally, replace each atom $R_\alpha(\bar{u})$ by $\alpha(\bar{u})$, yielding a CQ$^\neg$ formula of quantifier rank $k^2 + q$.

Finally, to obtain a formula equivalent to η_j, we define $\bar{\eta}_j$ from η_j^* as follows. For each equivalence class $e \in [\bar{x}_j]$ let v_e be an arbitrarily chosen variable in $\bar{x}_j$ such that $v_e \in e$. Let $\bar{\eta}(\bar{x}_j)$ be obtained by substituting v_e for each e in $\eta^*([\bar{x}_j])$ and adding the conjunction of all equalities $\{u = v \mid u, v \in \bar{x}_j, [u] = [v]\}$. The quantifier rank of $\bar{\eta}_j$ remains $k^2 + q$ and $\bar{\eta}_j$ is equivalent to η_j. □

The inherited constraints and some of the constructions described before are illustrated using our running example in Appendix A.3.

5.1. 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 $red(\alpha)$, 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 elaborate briefly on the procedure for constructing $red(\alpha)$ for a CQ$^\neg$ formula α. For each such α we first define a simplified representation of its syntax tree as a forest $\Im(\alpha)$ as follows (a tree consisting of root r and child subtrees $t_1, \ldots, t_n$ is denoted $r[t_1, \ldots, t_n]$):

—$\Im(L) = \{L\}$ for a literal L;
—$\Im(\beta_1 \wedge \beta_2) = \Im(\beta_1) \cup \Im(\beta_2)$;
—$\Im(\exists z(\beta)) = \{z[\Im(\beta)]\}$.

Thus, all internal nodes of $\Im(\alpha)$ are labeled by variables, and the leaves are literals. We define by structural recursion a partial order $\prec$ and then an equivalence relation $\sim$ on CQ$^\neg$ formulas that have the same free variables as follows. First, $\prec$ is defined on trees as follows.

—For literals L_1, L_2, $L_1 \prec L_2$ iff $L_1 = L_2$;
—For trees $z_1[F_1], z_2[F_2]$ (where z_1, z_2 are variables and F_1, F_2 forests) let z be a new variable. Then $z_1[F_1] \prec z_2[F_2]$ iff for each $t_1 \in F_1$ there exists $t_2 \in F_2$ such that $t_1(z_1 \leftarrow z) \prec t_2(z_2 \leftarrow z)$.

For CQ$^\neg$ formulas φ_1 and φ_2 with the same free variables, $\varphi_1 \prec \varphi_2$ if for each $t_1 \in \Im(\varphi_1)$ there exists $t_2 \in \Im(\varphi_2)$ such that $t_1 \prec t_2$. Finally, $\varphi_1 \sim \varphi_2$ iff $\varphi_1 \prec \varphi_2$ and $\varphi_2 \prec \varphi_1$.

Example 5.5. The forest corresponding to

$$Q(y) \wedge \exists x(R(x, y) \wedge P(y) \wedge \exists z(R(y, z) \wedge \neg R(z, z)) \wedge \exists u(R(y, u) \wedge \neg R(u, u)))$$

is

$$[Q(y), x[R(x, y), P(y), z[R(y, z), \neg R(z, z)], u[R(y, u), \neg R(u, u)]]].$$

We note the following property, immediate from the definition.

LEMMA 5.6. *If $\varphi_1 \sim \varphi_2$ then φ_1 and φ_2 are logically equivalent.*

Using the preceding, we define a normal form for CQ$^\neg$ formulas as follows. For each α, the *reduced* formula $red(\alpha)$ is obtained by eliminating multiple equivalent sibling subtrees, that is, retaining only one representative of each equivalence class of $\sim$ among all sibling subtrees.

Example 5.7. In the forest of Example 5.5, the subtrees rooted at variables z and u are $\sim$-equivalent, and a corresponding reduced formula has the forest $[Q(y), x[R(x, y), P(y), z[R(y, z), \neg R(z, z)]]]$.

Note that the construction of a reduced formula is nondeterministic, because of the arbitrary choice of a representative from each equivalence class. However, all the resulting reduced formulas are identical up to renaming of variables.

We observe the following.

LEMMA 5.8. *For $CQ^{\neg}$ formulas φ_1, φ_2, $\varphi_1 \sim \varphi_2$ iff $red(\varphi_1) = red(\varphi_2)$.*

In particular, $\alpha \sim red(\alpha)$ and α is equivalent to $red(\alpha)$.

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

PROOF. By Lemma 3.3 it follows that there exists Γ' with $k + |\bar{y}|$ artifact variables such that $\Gamma \models \forall \bar{y} \varphi_f$ iff $\Gamma' \models \varphi_f$. However, the variables in $\bar{y}$ are used in Γ' in a very limited way, and do not contribute to the quantifier rank of inherited constraints. Indeed, in every inherited constraint η_j there is a unique equivalence class for every $y \in \bar{y}$, which occurs free in η_j. The construction in Lemma 5.4 then yields a rewriting η_j^* of η_j with quantifier rank k^2 over a fixed vocabulary consisting of the relations in $\mathcal{DB}$ and $\mathcal{C}$ used in Γ and φ_f, as well as the new relations R_α for maximal formulas α used in Γ that have existential quantifications (see previous). This in turn yields the hyperexponential bound in k^2. $\square$

5.2. 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 5.10. 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 5.11. *There exists a run of Γ satisfying $\neg\varphi_f$ iff there exists a satisfiable symbolic lasso.*

PROOF. For the *only-if* part, suppose there is a run satisfying $\neg\varphi_f$. By Lemma 5.2, there exists a satisfiable symbolic run $\{\psi_i\}_{i \geq 0}$ whose corresponding sequence $\{\sigma_i\}_{i \geq 0}$ of truth assignments to P is accepted by $B_{\neg\varphi}$. Thus, there exists an accepting run $\{q_i\}_{i \geq 0}$ of $B_{\neg\varphi}$ on $\{\sigma_i\}_{i \geq 0}$. Let r be an accepting state of $B_{\neg\varphi}$ occurring infinitely often in the run.

Since there are finitely many inherited constraints, there exists an infinite subset I of integers such that $red(\eta_j^*)$ is the same for all $j \in I$ and $q_j = r$ for all $j \in I$. A simple pigeonhole argument further shows that there is an infinite subset J of I for which:

—for each $u, v \in \bar{x}$, $[u_{j_1}] = [v_{j_1}]$ iff $[u_{j_2}] = [v_{j_2}]$ for all $j_1, j_2 \in J$, and
—for each $u \in \bar{x}$ and $j_1, j_2 \in J$, $[u_{j_1}] = [u_{j_2}]$ or $[u_{j_1}] \neq [v_{j_2}]$ for all $v \in \bar{x}$.

Now pick arbitrary $j, j+n \in J$. Clearly, $\{\psi_i\}_{i<j+n}$ is satisfiable and is a symbolic lasso.

Consider the *if* part. Let $\lambda = \{\psi_i\}_{i<j+n}$ be a satisfiable symbolic lasso. Let D be an instance of $\mathcal{DB}$ for which $Runs_D(\lambda) \neq \emptyset$. We show that

$$(\dagger) \text{ there exists } \{\rho_i\}_{i\leq j+n} \in Runs_D(\lambda) \text{ such that } \rho_j = \rho_{j+n}.$$

Observe that (†) suffices to establish the *if* part of the lemma. Indeed, if (†) holds then the sequence $\{\rho_i\}_{i\leq j}(\{\rho_i\}_{j<i\leq j+n})^\omega$ is an actual run on D of the symbolic run $\{\psi_i\}_{i\leq j}(\{\psi_i\}_{j<i\leq j+n})^\omega$, which satisfies $\neg\varphi_f$.

Consider (†). Let $\bar{y}$ consist of the variables $y \in \bar{x}$ such that $[y_j] = [y_{j+n}]$ and let $\bar{v} = \bar{x} - \bar{y}$. Consider $\eta_j(\bar{y}_j, \bar{v}_j)$. Intuitively, $\bar{y}$ consist of the variables that are preserved from configuration j to $j+n$, while $\bar{v}_j$ and $\bar{v}_{j+n}$ are only related via $\bar{y}$. This together with the fact that $red(\eta_j^*(\bar{x})) = red(\eta_{j+n}^*(\bar{x}))$ will allow us to generate an actual run with the same configurations at j and $j+n$.

We next make this argument more precise. Recall the construction in the proof of Lemma 5.4. Consider $[\eta_{j+n}]$, with the augmented set of free variables $[\bar{y}], [\bar{v}_j], [\bar{v}_{j+n}]$. Let $\bar{u}$ be its quantified variables. Let $G = G_{j+n}$ and consider η_{j+n}^*. Note first that, because of feedback freedom, $[\bar{v}_j]$ and $[\bar{v}_{j+n}]$ occur in distinct connected components of $G|(\bar{u} \cup [\bar{v}_j] \cup [\bar{v}_{j+n}])$ (indeed, any path connecting a node in $[\bar{v}_j]$ to a node in $[\bar{v}_{j+n}]$ must go through a variable in $[\bar{y}]$). Let H_j consist of the connected components of $G|(\bar{u} \cup [\bar{v}_j] \cup [\bar{v}_{j+n}])$ containing nodes in $[\bar{v}_j]$, and H_{j+n} consist of those containing nodes in $[\bar{v}_{j+n}]$. Let $\chi_j([\bar{y}], [\bar{v}_j])$ and $\chi_{j+n}([\bar{y}], [\bar{v}_{j+n}])$ be the subformulas of $[\eta_{j+n}]$ containing variables in $[\bar{y}]$ and in H_j and H_{j+n}, respectively.

Consider now $[\eta_j]([\bar{y}], [\bar{v}_j])$ and let K_j be the subgraph of G_j consisting of the connected components of $G_j|(\bar{s} \cup [\bar{v}_j])$, that contain nodes in $[\bar{v}_j]$, where $\bar{s}$ are the quantified variables of $[\eta_j]$. Let $\xi_j([\bar{y}], [\bar{v}_j])$ be the subformula of $[\eta_j]$ containing nodes in $[\bar{y}]$ and K_j. As in the proof of Lemma 5.4, one can construct formulas $\chi_j^*([\bar{y}], [\bar{v}_j])$, $\chi_{j+n}^*([\bar{y}], [\bar{v}_{j+n}])$ and $\xi_j^*([\bar{y}], [\bar{v}_j])$. Intuitively, $\xi_j^*([\bar{y}], [\bar{v}_j])$ is the component of the inherited constraint η_j^* constraining $[\bar{v}_j]$ and $[\bar{y}]$. Similarly, $\chi_{j+n}^*([\bar{y}], [\bar{v}_{j+n}])$ plays the same role in the inherited constraint η_{j+n}^*. From the fact that $red(\eta_j^*([\bar{x}])) = red(\eta_{j+n}^*([\bar{x}]))$, it easily follows that $red(\xi_j^*([\bar{x}])) = red(\chi_{j+n}^*([\bar{x}]))$. In particular, $\xi_j^*([\bar{x}]) \sim \chi_{j+n}^*([\bar{x}])$.

Now consider how the formulas $\xi_j^*([\bar{y}], [\bar{v}_j])$ and $\chi_j^*([\bar{y}], [\bar{v}_j])$ differ. Intuitively, χ_j^* adds to ξ_j^* additional constraints imposed by the segment of λ between configuration j and $j+n$. Specifically, it is easily seen that

$$(\ddagger)\chi_j^*([\bar{y}], [\bar{v}_j]) = \xi_j^*([\bar{y}], [\bar{v}_j]) \wedge \mu([\bar{y}], [\bar{v}_j])$$

for some CQ$^\neg$ formula μ.

Consider a run $\rho = \{\rho_i\}_{i\leq j+n} \in Runs_D(\lambda)$. Let μ be the assignment to the variables in λ yielding ρ. We denote by $[\mu]$ the assignment to equivalence classes defined by $[\mu]([y_i]) = \mu(y_i)$ for each $y \in \bar{x}$ and $i \leq j+n$ (this is well-defined because μ must satisfy all formulas in λ, including the equalities). We modify μ as follows, yielding a new assignment ν. The assignment ν is the same as μ for all variables v for which $[v]$ is not in H_{j+n}. From (‡) it follows that $D \cup \mathcal{C} \models \xi_j^*([\mu]([\bar{x}_j]))$. Since $\xi_j^*([\bar{x}]) \sim \chi_{j+n}^*([\bar{x}])$, it follows that $D \cup \mathcal{C} \models \chi_{j+n}^*([\mu]([\bar{x}_j]))$. Let ν be defined on the variables in H_{j+n} by setting $\nu(\bar{x}_{j+n}) = \mu(\bar{x}_j)$ and extending $[\mu]([\bar{x}_j])$ to all variables in H_{j+n} by a choice of witnesses to the quantified variables in χ^*_{j+n} satisfying all quantifier-free subformulas of χ^*_{j+n}. It is clear that $D \cup \mathcal{C} \models \psi_i(\nu(\bar{x}_i, \bar{x}_{i+1}))$ for each $i < j + n$. It follows that $\{\nu(\bar{x}_i)\}_{i \leq j+n}$ is in $Runs_D(\lambda)$. By construction, $\nu(\bar{x}_j) = \nu(\bar{x}_{j+n})$ so the run satisfies (†). □

5.3. Decision Procedure

The previous 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 nondeterministically searches for a satisfiable symbolic lasso as follows.

ALGORITHM 1: Decision Procedure

```
Input: artifact system $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ and LTL-FO property $\varphi_f$ over $\mathcal{A}$,
       such that $(\Gamma, \varphi_f)$ is feedback-free.
Output: YES if and only if $\Gamma \models \varphi_f$.

(1)  flag := 0;
(2)  initialize a symbolic run prefix $\varrho$ to $\{\psi_0\}$, with corresponding truth assignment $\sigma_0$ to the
     propositions of $\varphi$ and set $s$ to some output of Büchi-Next($q_0, \sigma_0$) for some initial state $q_0$ of
     $B_{\neg\varphi}$;
(3)  set $\gamma$ to $red(\eta^*(\varrho))$, where $\eta(\varrho)$ is the inherited constraint for the prefix $\varrho$;
(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 go to step (6) or to
     step (7);
(6)  set $(\bar{s}, \bar{\gamma}) := (s, \gamma)$, flag:= 1, and $\mathcal{R} := \emptyset$;
(7)  non-deterministically generate from $\Delta$ a symbolic transition $\psi$ with corresponding truth
     assignment $\sigma$ to the propositions in $\varphi$;
(8)  set $s$ to Büchi-Next($s, \sigma$), $\gamma := red(\eta^*(\varrho\psi))$, and $\varrho := \varrho.\psi$;
(9)  if $(s, \gamma) \in \mathcal{R}$ then output NO and stop; otherwise, set
     $\mathcal{R} := \mathcal{R} \cup \{(s, \gamma)\}$
(10) 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 (10) 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 formulas $red(\eta^*(\varrho_1.\psi))$ and $red(\eta^*(\varrho_2.\psi))$ are identical up to renaming of variables.

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 (9).

Since there are finitely many reduced inherited constraints for symbolic runs of (Γ, φ_f) this bounds the running time of the aforesaid 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, with 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 $\bar{y} = vars(\xi|_{\mathcal{DB}}) \cap vars(\xi|_{\mathcal{C}})$, that is, 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 nonequality 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.

(‡). 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 nonequality atom, and (b) no tuple of terms appears both in a positive and a negative literal with the same relational symbol. Thus (ii) holds.

This establishes decidability of verification for LTL-FO properties of feedback-free systems with arithmetic, completing the proof of Theorem 5.1 for this case.

5.4. Complexity of the Verification Algorithm

The complexity analysis of the decision procedure just described 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 symbolic run prefix ϱ.

Performing the satisfiability check. This is step 10 of the decision procedure, which consists of:

(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 $\eta(\varrho)|_{\mathcal{C}} \wedge eq$ [Karmarkar 1984], 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 nonequality atom;

(iii) checking satisfiability of $\eta(\varrho)|_{\mathcal{DB}} \wedge eq$. This can be done 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 the next proposition.

PROPOSITION 5.12. *Static verification for feedback-free pairs of artifact systems with arithmetic and LTL-FO properties is decidable in time upper bounded by $h(k^2)$, for some $h \in Hyp$.*

5.5. Subclasses with Improved Upper Bounds

The previous analysis shows that the complexity of the decision procedure is dominated by the number of distinct inherited constraints, which upper bounds the length of the symbolic run. We identify next three subclasses 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 equivalent to some $y \in \bar{y}\}$. By extension, the width of a subgraph of G_j is the width of its set of nodes.

Definition 5.13. 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 5.14. *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 5.14 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 5.15. *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 $h \in Hyp$.*

PROOF. We claim that the quantifier rank of inherited constraints is bounded by k, which immediately implies the upper bound in the proposition's statement.

The proof is similar to the proof of Lemma 5.4, except the induction shows that β can be rewritten with quantifier rank bounded by $fp(\bar{y})$, with fp defined as follows.

In the notation of the proof of Lemma 5.4, define the *footprint* of $\bar{y}$ in $G_\beta|\bar{y}$ as the set of artifact variables whose values appear in the equivalence classes $\bar{y}$: $fp(\bar{y}) = \{x | x \in \bar{x}, y \in \bar{y}, [x_i] \in y \text{ for some } i\}$. Clearly, $|fp(\bar{y})| \leq k$.

We prove by induction on the size of $fp(\bar{y})$ that the formula corresponding to $\exists\bar{y}\beta(\bar{y}, \bar{s})$ can be rewritten with quantifier rank bounded by $|fp(\bar{y})|$. As in the proof of Lemma 5.4, consider $\bar{v} = \bar{y} - \bar{y}_{max}$, and let $H_1, \ldots, H_c$ be the connected components of $G_\beta|\bar{v}$ and $\bar{y}_r$ the set of nodes on H_r, for all $1 \leq r \leq c$.

Now observe that linear propagation ensures that each equivalence class consists of successive values of the same variable, $[y] = \{y_l | l \in span([y])\}$. Also observe that, if $[x_i] \in \bar{y}_{max}$, then for all l and all $1 \leq r \leq c$, $[x_l] \notin \bar{y}_r$ (otherwise the span maximality of $\bar{y}_{max}$ is contradicted). It follows that $fp(\bar{y}_r) \subseteq fp(\bar{y}) - fp(\bar{y}_{max})$, so $|fp(\bar{y}_r)| \leq |fp(\bar{y})| - |fp(\bar{y}_{max})|$, and the induction hypothesis applies to each H_r. □

COROLLARY 5.16. *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 5.15 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 5.17. (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 5.18. *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.*

PROOF. We recall from the proof of Lemma 5.4 the notation G_j for the graph on equivalence classes corresponding to symbolic run prefix $\varrho|_j$. We also extend the notion of span beyond equivalence classes to arbitrary subgraphs of G_j, in the natural way.

We claim that any connected component H of G_j has at most k distinct equivalence classes.

Indeed, given an equivalence class $[e]$ in H, denote with

$$vars([e]) = \{x | x \in \bar{x}, l \in span([e]), x_l \in [e]\},$$

that is, the set of artifact variables whose value belongs to $[e]$ at some step. Observe that acyclicity implies that there is no $i, j \in span(H)$ with $[x_i] \neq [x_j]$ in H. Therefore, the equivalence classes in H have pairwise disjoint $vars$ sets. This implies the claim.

By the claim, the corresponding subformula of the inherited constraint can be written using at most k variables. These are free if the span of their equivalence class includes j, and existentially quantified otherwise.

The number of distinct inherited constraints corresponding to connected components of G_j can be upper bounded as follows. By the claim, the number of distinct literals over k variables is upper bounded by $(2|\mathcal{DB} \cup \mathcal{C}|)^k$ (where $|\mathcal{DB} \cup \mathcal{C}|$ denotes the sum of the arities of the relation in $\mathcal{DB} \cup \mathcal{C}$). The number of distinct conjunctions thereof is bounded by

$2^{2|\mathcal{DB}\cup\mathcal{C}|^k}$. Since the number of distinct choices for the subset of existentially quantified variables is bounded by 2^k, we obtain a bound of $N = 2^{k(2|\mathcal{DB}\cup\mathcal{C}|^k)}$ on the number of distinct inherited constraint subformulas that correspond to connected components.

The proposition now follows from the preceding upper bound and the observation that the independence of connected components allows reusing variables across them. □

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 start 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.

Remark 5.19. Another special case stems from the hierarchical designs discussed informally in Section 4.3, and formalized in the AWE model of Damaggio [2011]. We can think of each task evolving within the context set by its ancestor tasks in the hierarchy. Since the descendant tasks can only read, but not change, the artifact variables of the ancestor tasks, the context variables' span includes the span of the descendant task variables. Moreover, each context variable naturally exhibits linear propagation. If properties are well-behaved as well (in the sense of Section 4.3, sticking to the context hierarchy and scoping rules), then the linear propagation property leads to $h(k)$ distinct inherited constraints, by Proposition 5.15.

6. ADDING 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.

6.1. Relevant Chase Properties

The chase [Abiteboul et al. 1995] 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, if it terminates (which is not guaranteed), its result is a finite instance U satisfying:

(a) U is a model of $\mathcal{I}$ and A (we say that U is a model of $\mathcal{I}$ and A if U satisfies the dependencies $\mathcal{I}$ and there is a homomorphism from A to U), and
(b) U is *universal* for $\mathcal{I}$ and A: that is, it has a homomorphism into every model of $\mathcal{I}$ and A.

We call such U a *universal model* for $\mathcal{I}$ and A.

At every *chase step*, the algorithm identifies a violation of some dependency in $\mathcal{I}$ and modifies A to remove this violation (the modification is minimal in a certain sense, and it possibly introduces new violations). When several violations exist, the choice of the one to remove is nondeterministic, and the sequence of such choices induces a sequence of chase steps, called a *chase sequence*. The sequence is *terminating* if it reaches an instance that satisfies $\mathcal{I}$.

We recall from Deutsch et al. [2008] 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 $\perp$. As soon as a chase step derives $\perp$, the chase sequence terminates, yielding the result $\perp$. 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.

6.2. Verification With Dependencies

We borrow from Meier et al. [2010] 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, Deutsch et al. [2008]), recent research has proposed sufficient syntactic restrictions. Examples include *weak acyclicity* [Fagin et al. 2003], stratification [Deutsch et al. 2008], and the *termination hierarchy* [Meier et al. 2010] 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 10, 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 aforesaid 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.*

PROOF. The proof follows straightforwardly from the fact that the witnesses for satisfiability of $\xi|_{\mathcal{DB}}$ and $\xi|_{\mathcal{C}}$, call them $w_{\mathcal{DB}}$ and $w_{\mathcal{C}}$ respectively, both satisfy $eq(\bar{y}, \bar{c})$. This allows an isomorphism that takes $w_{\mathcal{DB}}$ into $w_{\mathcal{C}}$, and since $\xi_{\mathcal{DB}}$ is preserved under isomorphisms, $w_{\mathcal{C}}$ is also a witness for $\xi_{\mathcal{DB}}$. $\square$

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 Deutsch et al. [2008] 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 $\perp$. As soon as a chase step derives $\perp$, the chase sequence terminates, yielding the result $\perp$. 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

$$\begin{aligned} \delta_{\neq} &: \forall x \forall y \; x = y \wedge x \neq y \rightarrow \perp \\ \delta_{\neg}^{P} &: \forall \bar{x} \; P(\bar{x}) \wedge \neg P(\bar{x}) \rightarrow \perp. \end{aligned}$$

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 $q \in \boldsymbol{CQ}^{\neg}$ 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.*

PROOF. Let's denote the fact that there is a chase sequence s from q to q' with $q \overset{s}{\Rightarrow} q'$.

(i) If $\mathcal{I} \in CT_{\forall\exists}$, then there is a terminating chase sequence s_1 of q. By definition, s_1 is finite and if $q \overset{s_1}{\Rightarrow} q'$, q' satisfies all dependencies in $\mathcal{I}$ (as $\mathcal{I}$ does not mention $\perp$).

Observe that the chase steps with dependencies from $\mathcal{I}_{\mathcal{DB}}$ only introduce $\perp$, which is not mentioned in $\mathcal{I}$. Therefore, chase steps with dependencies from $\mathcal{I}_{\mathcal{DB}}$ cannot enable chase steps with dependencies from $\mathcal{I}$. Also observe that the chase with $\mathcal{I}_{\mathcal{DB}}$ must terminate after at most one step (either no dependency in $\mathcal{I}_{\mathcal{DB}}$ applies, or if one applies, then the chase fails).

Let s_2 be a chase sequence with $\mathcal{I}_{\mathcal{DB}}$ such that $q' \overset{s_2}{\Rightarrow} q''$. By the previous observation, s_2 is terminating, and has length at most 1. If the length is 0, then $q'' = q'$, so q'' satisfies $\mathcal{I} \cup \mathcal{I}_{\mathcal{DB}}$. If the length is 1, then $q'' = \perp$, and the chase fails. Therefore s_1, s_2 is a terminating chase sequence with $\mathcal{I} \cup \mathcal{I}_{\mathcal{DB}}$, thus witnessing that $\mathcal{I} \cup \mathcal{I}_{\mathcal{DB}} \in CT_{\forall\exists}$.

(ii) By (i), there is a terminating chase sequence s of q with $\mathcal{I} \cup \mathcal{I}_d b$. Denote with $s|_i$ the prefix of length i of s, with s_i the i'th chase step, and with q_i the result of the first i chase steps: $q \overset{s|_i}{\Rightarrow} q_i$.

If s does not fail, then the result of s, when viewed as an instance, witnesses satisfiability.

Suppose that the chase fails. Then we prove by contradiction that there is no model that satisfies q and $\mathcal{I}$.

For assume that such a model D exists. Let n be the length of s. Since the chase fails, and failure must occur after the first application of any dependency in $\mathcal{I}_{\mathcal{DB}}$, $s|_{n-1}$ uses only dependencies from $\mathcal{I}$, and s_n must use a dependency $\delta \in \mathcal{I}_{\mathcal{DB}}$. The premise of δ must map homomorphically into q_{n-1}. Call this homomorphism h.

Moreover, let $q = q^- \wedge q^+$, where q^- is the subquery of q consisting of all negative literals, and q^+ the subquery consisting of all positive literals.

Since $\mathcal{I}$ mentions only positive literals, the first $n-1$ chase steps apply only to q^+, so $q^-_{n-1} = q^-$ and $q^+ \overset{s|_{n-1}}{\Rightarrow} q^+_{n-1}$. Recall that the premise of all dependencies in $\mathcal{I}_{\mathcal{DB}}$ consists of a positive literal L and its negation $\neg L$. In particular, it follows that h maps L into q^+_{n-1}, and $\neg L$ into q^-.

Notice that the chase of q^+ is standard, involving a positive conjunctive query and positive dependencies. It is known (see Abiteboul et al. [1995] for instance) that the standard chase preserves universality. In particular, this means that for every $1 \leq i \leq n-1$, q^+_i has a homomorphic mapping into every instance that satisfies q^+ and $\mathcal{I}$. In particular, there is a homomorpism m from q^+_{n-1} into D.

But then D must contain an image of L under $h \circ m$, which leads to a contradiction: since D satisfies all of q, it must satisfy $\neg h \circ m(L)$ and $h \circ m(L)$ simultaneously. $\square$

Remark 6.5. We could have applied the more general decision procedure from Deutsch et al. [2008], for satisfiability of CQ$^\neg$ under a set of dependencies with negated literals. The result in Deutsch et al. [2008] 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 blowup can be avoided by resorting to the standard (nondisjunctive) 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. $\square$

Complexity. The complexity upper bound obtained in the absence of dependencies is virtually unaffected by the presence of sets of dependencies from the termination hierarchy. This holds also for the improved complexity of the subclasses introduced in Section 5.5.

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 a conservative bound for the polynomial's degree being the size of $\mathcal{I}$ [Meier et al. 2010; Deutsch et al. 2008; Fagin et al. 2003]. These works perform a more refined analysis, in essence showing a tighter bound that depends on the longest path in a graph that reflects how a chase step with one dependency can trigger another [Meier et al. 2010; Deutsch et al. 2008; Fagin et al. 2003].

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* [Abiteboul et al. 1995]. It also holds if only $\mathcal{DB}$ is fixed while $\mathcal{I}$ is not, but it consists only of *full* dependencies [Abiteboul et al. 1995]. Equality-generating dependencies allow only equality atoms in the conclusion, and capture as particular case the class of functional dependencies. Full dependencies contain no existentially quantified variables.

Note that in all cases mentioned earlier, the complexity of the chase-based satisfiability test at each step of the symbolic run is polynomial in the size of the inherited constraint and thus the additional computation due to the dependencies is absorbed into the hyperexponential computation we inherit from the dependency-free case. This observation continues to hold for the artifact system subclasses of Section 5.5.

Also note that the definition of feedback freedom is oblivious to dependencies, yielding a dependency-independent bound on the number of syntactically distinct inherited constraints. Under dependencies, the number of inherited constraints actually explored by the verification algorithm can only decrease. The decrease is due to the case when a symbolic run prefix is satisfiable but not $\mathcal{I}$-satisfiable. The continuaton of the symbolic run prefix would be explored by the dependency-oblivious verifier, and pruned by the dependency-aware one.

7. CONCLUSION

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 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 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. We discuss encouraging evidence in support of this claim in Section 4.3, which mentions a widely adopted business process design paradigm that naturally results in feedback freedom, and a subclass of a recent business process model that admits simulation by feedback-free artifact systems.

The results presented here for feedback-free systems can likely be extended in various ways. Decidability results for restricted counter machines (such as single-counter, or reversal bounded multicounters [Ibarra 1978; Demri et al. 2010]) can plausibly transfer to the artifact model, extending verification to systems with more powerful arithmetic. Work in progress also considers lifting the feedback-freedom constraint from the order relation on the numeric domain (thus extending previous results on verification of artifact systems with a totally ordered domain [Deutsch et al. 2009; Segoufin and Torunczyk 2011]). Finally, we plan to consider verification for various extensions of the artifact model. Such extensions allow artifacts to carry richer data than simple tuples, provide richer control mechanisms, and allow multiple artifact instances in a run.

REFERENCES

ABITEBOUL, S., HERR, L., AND DEN BUSSCHE, J. V. 1996. Temporal versus first-order logic to query temporal databases. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'96).* 49–57.

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison Wesley.

ABITEBOUL, S., SEGOUFIN, L., AND VIANU, V. 2009. Static analysis of active xml systems. *ACM Trans. Datab. Syst. 34*, 4.

ABITEBOUL, S., VIANU, V., FORDHAM, B., AND YESHA, Y. 2000. Relational transducers for electronic commerce. *J. Comput. Syst. Sci. 61*, 2, 236–239.

BHATTACHARYA, K., GUTTMAN, R., LYMAN, K., HEATH III, F. F., KUMARAN, S., NANDI, P., WU, F., ATHMA, P., FREIBERG, C., JOHANNSEN, L., AND STAUDT, A. 2005. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Syst. J. 44*, 1, 145–162.

BHATTACHARYA, K., CASWELL, N. S., KUMARAN, S., NIGAM, A., AND WU, F. Y. 2007a. Artifact-Centered operational modeling: Lessons from customer engagements. *IBM Syst. J. 46,* 4, 703–721.

BHATTACHARYA, K., GEREDE, C. E., HULL, R., LIU, R., AND SU, J. 2007b. Towards formal analysis of artifact-centric business process models. In *Proceedings of the International Conference on Business Process Management (BPM'07).* 288–304.

BIRGET, J.-C. 1996. Two-Way automata and length-preserving homomorphisms. *Theory Comput. Syst. 29*, 3, 191–226.

BOJANCZYK, M., MUSCHOLL, A., SCHWNTICK, T., SEGOUFIN, L., AND DAVID, C. 2006. Two-Variable logic on words with data. In *Proceedings of the Annual IEEE Symposium on Logic in Computer Science (LICS'06).* 7–16.

BOUAJJANI, A., HABERMEHL, P., JURSKI, Y., AND SIGHREANU, M. 2007a. Rewriting systems with data. In *Proceedings of the International Symposium on Fundamentals of Computation Theory (FCT'07).* Lecture Notes in Computer Science, vol. 4639, Springer, 1–22.

BOUAJJANI, A., JURSKI, Y., AND SIGHIREANU, M. 2007b. A generic framework for reasoning about dynamic networks of infinite-state processes. In *Proceedings of the International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'07).* Lecture Notes in Computer Science, vol. 4424, Springer, 690–705.

BOUAJJANI, A., HABERMEHL, P., AND MAYR, R. 2003. Automatic verification of recursive procedures with one integer parameter. *Theor. Comput. Sci. 295*, 85–106.

BOUYER, P. 2002. A logical characterization of data languages. *Inf. Process. Lett. 84*, 2, 75–85.

BOUYER, P., PETIT, A., AND THÉRIEN, D. 2003. An algebraic approach to data languages and timed languages. *Inf. Comput. 182*, 2, 137–162.

BURKART, O., CAUCAL, D., MOLLER, F., AND STEFFEN, B. 2001. Verification of infinite structures. In *Handbook of Process Algebra*, Elsevier Science, 545–623.

CALVANESE, D., GIACOMO, G. D., HULL, R., AND SU, J. 2009. Artifact-Centric workflow dominance. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC)/Service Wave*. 130–143.

DAMAGGIO, E. 2011. Verification of business process specifications with arithmetic and data dependencies. Ph.D. thesis, University of California San Diego.

DAMAGGIO, E., DEUTSCH, A., AND VIANU, V. 2011a. Artifact systems with data dependencies and arithmetic. In *Proceedings of the International Conference on Database Theory (ICDT'11).*

DAMAGGIO, E., HULL, R., AND VACULIN, R. 2011b. On the equivalence of incremental and fixpoint semantics for business entities with guard-stage-milestone lifecycles. In *Proceedings of the International Conference on Business Process Management (BPM'11).*

DEMRI, S. AND LAZIĆ, R. 2006. LTL with the freeze quantifier and register automata. In *Proceedings of the Annual IEEE Symposium on Logic in Computer Science (LICS'06).* 17–26.

DEMRI, S., LAZIĆ, R., AND SANGNIER, A. 2008. 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).* 490–504.

DEMRI, S., LAZIĆ, R., AND SANGNIER, A. 2010. Model checking memoryful linear-time logics over one-counter automata. *Theor. Comput. Sci. 411*, 22-24, 2298–2316.

DEUTSCH, A., NASH, A., AND REMMEL, J. B. 2008. The chase revisited. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'08).* 149–158.

DEUTSCH, A., HULL, R., PATRIZI, F., AND VIANU, V. 2009. Automatic verification of data-centric business processes. In *Proceedings of the International Conference on Database Theory (ICDT'09).* 252–267.

DEUTSCH, A., SUI, L., AND VIANU, V. 2007. Specification and verification of data-driven Web applications. *J. Comput. Syst. Sci. 73*, 3, 442–474.

DEUTSCH, A., SUI, L., VIANU, V., AND ZHOU, D. 2006. Verification of communicating data-driven Web services. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'06)*. 90–99.

DONG, G., HULL, R., KUMAR, B., SU, J., AND ZHOU, G. 1999. A framework for optimizing distributed workflow executions. In *Proceedings of the International Workshop on Database Programming Languages (DBPL'99)*. 152–167.

EMERSON, E. A. 1990. Temporal and modal logic. In *Handbook of Theoretical Computer Science, vol. B: Formal Models and Semantics*, J. V. Leeuwen, Ed., North-Holland/MIT Press, 995–1072.

FAGIN, R. 1982. Horn clauses and database dependencies. *J. ACM 29*, 4, 952–985.

FAGIN, R., KOLAITIS, P. G., MILLER, R. J., AND POPA, L. 2003. Data exchange: Semantics and query answering. In *Proceedings of the International Conference on Database Theory (ICDT'03)*. 207–224.

FRITZ, C., HULL, R., AND SU, J. 2009. Automatic construction of simple artifact-based business processes. In *Proceedings of the International Conference on Database Theory (ICDT'09)*. 225–238.

GEREDE, C. E., BHATTACHARYA, K., AND SU, J. 2007. Static analysis of business artifact-centric operational models. In *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*.

GEREDE, C. E. AND SU, J. 2007. Specification and verification of artifact behaviors in business process models. In *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07)*.

GLUSHKO, R. AND MCGRATH, T. 2005. *Document Engineering: Analyzing and Designing Documents for Business Informatics and Web Services*. MIT Press, Cambridge, MA.

HULL, R., DAMAGGIO, E., FOURNIER, F., GUPTA, M., HEATH, F. T., HOBSON, S., LINEHAN, M. H., MARADUGU, S., NIGAM, A., SUKAVIRIYA, P., AND VACULIN, R. 2010. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proceedings of the International Workshop on Web Services and Formal Methods (WS-FM'10)*. 1–24.

HULL, R., LLIRBAT, F., KUMAR, B., ZHOU, G., DONG, G., AND SU, J. 2000. Optimization techniques for data-intensive decision flows. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'00)*. 281–292.

HULL, R., LLIRBAT, F., SIMON, E., SU, J., DONG, G., KUMAR, B., AND ZHOU, G. 1999. Declarative workflows that support easy modification and dynamic browsing. In *Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration*.

IBARRA, O. H. 1978. Reversal-Bounded multicounter machines and their decision problems. *J. ACM 25*, 1, 116–133.

JURDZINSKI, M. AND LAZIĆ, R. 2007. Alternation-Free modal mu-calculus for data trees. In *Proceedings of the Annual IEEE Symposium on Logic in Computing Science (LICS'07)*. 131–140.

KARMARKAR, N. 1984. A new polynomial-time algorithm for linear programming. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'84)*. 302–311.

KUMARAN, S., LIU, R., AND WU, F. Y. 2008. On the duality of information-centric and activity-centric models of business processes. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAISE'08)*.

KUMARAN, S., NANDI, P., HEATH, T., BHASKARAN, K., AND DAS, R. 2003. ADoc-Oriented programming. In *Proceedings of the Symposium on Applications and the Internet (SAINT'03)*. 334–343.

KÜSTER, J., RYNDINA, K., AND GALL, H. 2007. Generation of bpm for object life cycle compliance. In *Proceedings of the 5th International Conference on Business Process Management (BPM'07)*.

LADNER, R. E., LIPTON, R. J., AND STOCKMEYER, L. J. 1984. Alternating pushdown and stack automata. *SIAM J. Comput. 13*, 1, 135–155.

LAZIĆ, R., NEWCOMB, T., OUAKNINE, J., ROSCOE, A., AND WORRELL, J. 2007. Nets with tokens which carry data. In *Proceedings of the International Conference on Applications and Theory of Petri Nets (ICATPN'07)*. Lecture Notes in Computer Science, vol. 4546, Springer, 301–320.

LIBKIN, L. 2004. *Elements of Finite Model Theory*. Springer.

LIU, R., BHATTACHARYA, K., AND WU, F. Y. 2007. Modeling business contexture and behavior using business artifacts. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAISE'07)*. Lecture Notes in Computer Science, vol. 4495, Springer.

MARTENS, W., NEVEN, F., AND GYSSENS, M. 2008. Typechecking top-down xml transformations: Fixed input or output schemas. *Inf. Comput. 206*, 7, 806–827.

MARTIN, D., BURSTEIN, M., HOBBS, J., LASSILA, O., MCDERMOTT, D., MCILRAITH, S., NARAYANAN, S., PAOLUCCI, M., PARSIA, B., PAYNE, T., SIRIN, E., SRINIVASAN, N., AND SYCARA, K. 2003. OWL-S: Semantic markup for Web services. W3C member submission.

MCILRAITH, S. A., SON, T. C., AND ZENG, H. 2001. Semantic Web services. *IEEE Intell. Syst. 16*, 2, 46–53.

MEIER, M., SCHMIDT, M., WEI, F., AND LAUSEN, G. 2010. Semantic query optimization in the presence of types. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'10).* 111–122.

MINSKY, M. L. 1967. *Computation: Finite and Infinite Machines*. Prentice-Hall, Upper Saddle River, NJ.

NANDI, P. AND KUMARAN, S. 2005. Adaptive business objects – A new component model for business integration. In *Proceedings of the International Conference on Enterprise Information Systems.* 179–188.

NARAYANAN, S. AND MCILRAITH, S. 2002. Simulation, verification and automated composition of Web services. In *Proceedings of the International World Wide Web Conference (WWW'02).*

NEVEN, F., SCHWENTICK, T., AND VIANU, V. 2004. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic 5*, 3, 403–435.

NIGAM, A. AND CASWELL, N. S. 2003. Business artifacts: An approach to operational specification. *IBM Syst. J. 42*, 3, 428–445.

PNUELI, A. 1977. The temporal logic of programs. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'77).* 46–57.

RUSSELL, N., HOFSTEDE, A. H. M. T., AND EDMOND, D. 2005a. Workflow resource patterns: Identification, representation and tool support. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*. Lecture Notes in Computer Science, vol. 3520, Springer, 216–232.

RUSSELL, N., TER HOFSTEDE, A. H. M., EDMOND, D., AND VAN DER AALST, W. M. P. 2005b. Workflow data patterns: Identification, representation and tool support. In *Proceedings of the 25th International Conference on Conceptual Modeling (ER'05).* Springer.

RUSSELL, N., VAN DER AALST, W. M. P., AND TER HOFSTEDE, A. H. M. 2006. Workflow exception patterns. In *Proceedings of the 18th Conference on Advanced Information Systems Engineering (CAiSE'06).* Springer, 288–302.

SEGOUFIN, L. AND TORUNCZYK, S. 2011. Automata based verification over linearly ordered data domains. In *International Symposium on Theoretical Aspects of Computer Science (STACS'11).*

SPIELMANN, M. 2003. Verification of relational transducers for electronic commerce. *J. Comput. Syst. Sci. 66*, 1, 40–65. Extended abstract in *ACM Symposium on Principles of Database Systems (PODS'00).*

WANG, J. AND KUMAR, A. 2005. A framework for document-driven workflow systems. In *Business Process Management*, 285–301.

ZHAO, X., SU, J., YANG, H., AND QIU, Z. 2009. Enforcing constraints on life cycles of business artifacts. In *Proceedings of the IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE'09).* 111–118.

Received October 2011; accepted January 2012